UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Industrial

Estudio comparativo de algoritmos para el modelado del contacto de la rueda de un vehículo ferroviario con el carril

Trabajo Fin de Máster

Máster Universitario en Ingeniería Industrial

AUTOR/A: Pons Torres, Borja

Tutor/a: Baeza González, Luis Miguel

CURSO ACADÉMICO: 2023/2024

“A mi familia y a mi pareja por su apoyo y ayuda incondicional,

a mi tutor Luis Baeza por su ayuda y paciencia,

a mis amigos por estar ahí siempre,

a mi psicólogo

y a un amigo muy especial”

RESUMEN

La simulación de la dinámica asociada al transporte ferroviario constituye una herramienta esencial para el diseño tanto de vehículos como de la vía, permitiendo abordar problemas tan importantes como el confort vibratorio, la seguridad de marcha o el ruido de rodadura. Los modelos mecánicos empleados en los programas de simulación deben integrar teorías del contacto rueda-carril, las cuales relacionan las fuerzas transmitidas entre los sólidos con las variables cinemáticas del contacto. Las distintas teorías disponibles en la literatura suelen tratar por separado el problema del contacto normal y el tangencial, diferenciándose según la dirección de las tensiones que se estudian, ya sea normales al plano de contacto o contenidas en él. Pero sobre todo se caracterizan por su complejidad matemática y computacional, y su capacidad para reproducir las condiciones reales que se dan en el contacto. A pesar del incremento creciente de la capacidad informática, el coste computacional representa el mayor obstáculo para que las teorías más precisas sean implementadas de forma generalizada por los paquetes informáticos que son empleados por la industria ferroviaria.

Este trabajo analiza y compara diferentes algoritmos para resolver el problema del contacto normal rueda-carril en situaciones no hercianas, es decir, sin recurrir a las hipótesis de Hertz que se limitan a geometrías de contacto aproximables a cuádricas. El objetivo principal es comparar estos métodos evaluando su precisión y eficiencia computacional, con el fin de identificar cuál de ellos ofrece un mejor rendimiento. Para este proyecto, se implementaron en programas informáticos tres algoritmos para resolver el problema del contacto normal: el método NORM del software CONTACT, una aproximación basada en distribuciones de tensiones polinómicas o elípticas utilizando el método NORM como base, y un tercer algoritmo que emplea el método de gradiente conjugado con restricciones acotadas. Con el fin de establecer una independencia con la herramienta de cálculo, todos los métodos fueron programados en Fortran.

El caso de estudio corresponde con un eje montado centrado en la vía sin ángulo de lazo. El perfil de rueda es un S1002, y el del carril un UIC60 inclinado 1/40. Entre los resultados se incluye resultados directos de los modelos (distribuciones de presiones en el contacto, fuerza normal de contacto), comparativas de los resultados con respecto al método CONTACT original (errores relativos y absolutos máximos de la distribución de tensiones), y diversas estimaciones del coste computacional (tiempo de cálculo, eficiencia computacional en comparación con el método CONTACT, número de iteraciones necesarias, memoria empleada por los programas).

De los resultados de este trabajo se concluye que la combinación de NORM con la aproximación elíptica o polinómica de las distribuciones de tensiones es 78 veces más rápida que CONTACT original y 2 veces más rápida que el método que emplea el gradiente conjugado con restricciones acotadas. En cuanto al error cometido, ambos métodos proporcionan resultados muy parecidos a la solución de referencia, si bien el método basado en el gradiente conjugado reduce el error prácticamente a cero. En el método que utiliza bases elípticas o polinómicas el error depende de la naturaleza de la base, siendo del 0.0259% para distribuciones elípticas, y del 0.0171% para polinómicas de grado 4.

Palabras Clave: problema de contacto normal, contacto rueda-carril, ferrocarriles, algoritmos de resolución, NORM, CONTACT, BCCG(K), MATLAB, Fortran, optimización del rendimiento.

RESUM

La simulació de la dinàmica associada al transport ferroviari constitueix una eina essencial per al disseny tant de vehicles com de la via, permetent abordar problemes tan importants com el confort vibratori, la seguretat de marxa o el soroll de rodadura. Els models mecànics emprats en els programes de simulació han d'integrar teories del contacte roda-rail, les quals relacionen les forces transmeses entre els sòlids amb les variables cinemàtiques del contacte. Les diferents teories disponibles en la literatura solen tractar per separat el problema del contacte normal i el tangencial, diferenciant-se segons la direcció de les tensions que s'estudien, ja siga normals al pla de contacte o contingudes en ell. Però sobretot es caracteritzen per la seua complexitat matemàtica i computacional, i la seua capacitat per a reproduir les condicions reals que es donen en el contacte. A pesar de l'increment creixent de la capacitat informàtica, el cost computacional representa el major obstacle perquè les teories més precises siguen implementades de manera generalitzada pels paquets informàtics que són emprats per la indústria ferroviària.

Aquest treball analitza i compara diferents algoritmes per a resoldre el problema del contacte normal roda-rail en situacions no hertzianes, és a dir, sense recórrer a les hipòtesis de Hertz que es limiten a geometries de contacte aproximables a quàdriques. L'objectiu principal és comparar aquests mètodes avaluant la seua precisió i eficiència computacional, amb la finalitat d'identificar quin d'ells ofereix un millor rendiment. Per a aquest projecte, es van implementar en programes informàtics tres algoritmes per a resoldre el problema del contacte normal: el mètode NORM del programari CONTACT, una aproximació basada en distribucions de tensions polinòmiques o el·líptiques utilitzant el mètode NORM com a base, i un tercer algoritme que empra el mètode de gradient conjugat amb restriccions acotades. Amb la finalitat d'establir una independència amb l'eina de càlcul, tots els mètodes van ser programats en Fortran.

El cas d'estudi correspon amb un eix muntat centrat en la via sense angle de llaç. El perfil de roda és un S1002, i el del rail un UIC60 inclinat 1/40. Entre els resultats s'inclouen resultats directes dels models (distribucions de pressions en el contacte, força normal de contacte), comparatives dels resultats respecte al mètode CONTACT original (errors relatius i absoluts màxims de la distribució de tensions), i diverses estimacions del cost computacional (temps de càlcul, eficiència computacional en comparació amb el mètode CONTACT, nombre d'iteracions necessàries, memòria emprada pels programes). Dels resultats d'aquest treball es conclou que la combinació de NORM amb l'aproximació el·líptica o polinòmica de les distribucions de tensions és 78 vegades més ràpida que CONTACT original i 2 vegades més ràpida que el mètode que empra el gradient conjugat amb restriccions acotades. Quant a l'error comés, tots dos mètodes proporcionen resultats molt semblants a la solució de referència, si bé el mètode basat en el gradient conjugat redueix l'error pràcticament a zero. En el mètode que utilitza bases el·líptiques o polinòmiques l'error depèn de la naturalesa de la base, sent del 0.0259% per a distribucions el·líptiques, i del 0.0171% per a polinòmiques de grau 4.

Paraules Clau: problema de contacte normal, contacte roda-rail, ferrocarrils, algoritmes de resolució, NORM, CONTACT, BCCG(K), MATLAB, Fortran, optimització del rendiment.

ABSTRACT

The railway dynamics simulation is an essential tool for the design of both vehicles and tracks, addressing key issues such as vibration comfort, running safety, and rolling noise. The mechanical models used in simulation programs must incorporate wheel-rail contact theories, which relate the forces transmitted between solids with the kinematic variables of the contact. The different theories in the literature typically address the normal and tangential contact problems separately, distinguishing between the direction of the tractions under study, whether normal to the contact plane or contained within it. However, they are mainly characterized by their mathematical and computational complexity, as well as their ability to reproduce the real contact conditions. Despite the growing increase in computing power, computational cost remains the biggest obstacle preventing the widespread implementation of more accurate theories in the software packages used by the railway industry.

This work analyzes and compares different algorithms for solving the wheel-rail normal contact problem in non-Hertzian cases, i.e., without adopting Hertz's assumptions, which are limited to quadric contact surfaces. The primary objective is to compare these methods by evaluating their accuracy and computational efficiency to identify which one offers the best performance. For this project, three algorithms were implemented in software to solve the normal contact problem: the NORM method from the CONTACT software, an approach based on polynomial or elliptical traction distributions using the NORM method as a base, and a third algorithm that implements the enhanced bound-constrained conjugate gradient. To ensure independence from the calculation tool, all methods were programmed in Fortran.

The case study corresponds to a centered wheelset on the track with no yaw angle. The wheel profile is an S1002, and the rail profile is a UIC60 inclined at 1/40. The results include direct outputs from the models (contact traction distributions, total normal contact force), comparisons with the original CONTACT method (maximum relative and absolute errors in the traction distribution), and various estimates of computational cost (calculation time, computational efficiency compared to the CONTACT method, number of iterations needed, memory needed by the programs).

From the results of this work, it is concluded that combining NORM with the elliptical or polynomial approach is 78 times faster than the original CONTACT and 2 times faster than the method using the conjugate gradient. Regarding the error committed, both methods provide results very close to the reference solution, although the conjugate gradient-based method reduces the error to nearly zero. In the method that uses elliptical or polynomial bases, the error depends on the nature of the base, being 0.0259% for elliptical distributions, and 0.0171% for polynomial ones of degree 4.

Keywords: normal contact problem, wheel-rail contact, railways, resolution algorithms, NORM, CONTACT, BCCG(K), MATLAB, Fortran, performance optimization.

DOCUMENTOS CONTENIDOS EN EL TFM

LISTA DE FIGURAS 6
LISTA DE TABLAS 7
AGRADECIMIENTOS 8
FINANCIACIÓN 9
MEMORIA 10
PRESUPUESTO 109
ANEXOS 113

LISTA DE FIGURAS

Figura 1. Secuencia de resolución del problema de contacto rueda-carril (Baeza, 2022). 16
Figura 2. Pseudodeslizamiento (Baeza, 2022). 16
Figura 3. Relación de la Fuerza tangencial con la velocidad del punto de contacto en un modelo de contacto 2D (Baeza, 2022). 17
Figura 4. Contacto conforme (izquierda) y no-conforme (derecha) (Baeza, 2022). 18
Figura 5. Perfil S1002 (eje centrado en vía) sobre carril UIC 60 inclinado 1/20 (Baeza, 2022). 18
Figura 6. Sistema de referencia empleado en el contacto (Baeza, 2022). 19
Figura 7. Configuraciones: Referencia, no-deformada y deformada (Baeza, 2022). 22
Figura 8. Configuración de referencia (verde), no-deformada (azul) y deformada (rojo), para el carril y la rueda (Baeza, 2022). 23
Figura 9. Configuración de referencia (verde), no-deformada (azul) y deformada (rojo), para el carril y la rueda (Baeza, 2022). 24
Figura 10. Puntos de contacto de la rueda y el carril (Baeza, 2022). 25
Figura 11. Spin de la superficie de contacto (Baeza, 2022). 26
Figura 12. Cálculo de la velocidad asociada a la configuración indeformada $\boldsymbol{w}$ (Baeza, 2022). 27
Figura 13. Propiedades de cuasiidentidad elástica (Baeza, 2022). 28
Figura 14. Discretización espacial de CONTACT. En rojo aparece el área real de contacto y en gris la estimada por CONTACT (Baeza, 2022). 30
Figura 15. Ejemplo de la distribución de tensiones normales calculadas con CONTACT (Baeza, 2022). 30
Figura 16. Teoría de contacto según Hertz (Sichani, 2016). 34
Figura 17. Contacto según la Teoría Lineal de Kalker (San-Segundo Cordero, 2019). 38
Figura 18. Variación del error relativa con el número de iteraciones (Vollebregt, 2014). 49
Figura 19. Definición de los índices de posición en Fortran para el método NORM clásico. 59
Figura 20. Iteraciones del Gradiente Conjugado (Apuntes de métodos matemáticos, 2019). 63
Figura 21. Ejemplo de área de contacto con la que determinar $\boldsymbol{Q}$ y $\boldsymbol{B}$ (Giner Navarro, Gómez Bosch, Alonso, & Baeza, 2023). 72
Figura 22. Área potencial de contacto del caso de estudio empleado (elaboración propia). 88
Figura 23. Forma del área de contacto (roja) ilustrada por la matriz $\boldsymbol{P}$ tras la resolución del problema normal (elaboración propia). 90
Figura 24. Número de iteraciones hasta resolver el problema normal frente al grado utilizado en la aproximación polinómica del algoritmo NORM+E/P+CG (elaboración propia). 102
Figura 25. Errores absolutos reportados por Giner Navarro et al. (2023): (a) Aproximación elíptica de las tensiones (para un desplazamiento lateral del eje montado $y = 0$); (b) Aproximación polinómica de las tensiones (para un ángulo nulo de guiñada en el eje montado y $g = 4$). 104
Figura 26. Error relativo de la fuerza normal para diferentes desplazamientos del eje montado según Giner Navarro et al. (2023): (a) Aproximación elíptica; (b) Aproximación polinómica con $g = 4$. 104

LISTA DE TABLAS

Tabla 1. Coeficientes de Hertz ($K1/K2 < 1$) (Iwnicki, 2006). 36
Tabla 2. Coeficientes de Hertz para $\theta = 0°$ hasta $180°$ (Iwnicki, 2006). 36
Tabla 3. Valores de los coeficientes de deslizamiento y giro según Kalker (San-Segundo Cordero, 2019). 40
Tabla 4. Errores relativos de los algoritmos respecto a la Fuerza Normal Total obtenida. NORM+E y NORM+P(g=2) indican que la variante NORM+E/P opera con la aproximación elíptica o la polinómica de grado 2, respectivamente. 84
Tabla 5. FLOPs de las operaciones básicas (Golub & Van Loan, 2013). 91
Tabla 6. Tiempos de ejecución: tiempo de CPU de los algoritmos (en segundos) para todas las ejecuciones. 93
Tabla 7. Tiempos de ejecución: Tiempo medio de CPU, eficiencia computacional, número total de iteraciones y número de elementos en el área potencial. 94
Tabla 8. Estimación del número de FLOPs de los algoritmos implementados. 95
Tabla 9. Tiempos de ejecución: eficiencia computacional del método de la columna respecto al método de la fila. 96
Tabla 10. Precisión de los resultados: errores relativos y absolutos. El superíndice $*$ denota que se trata del error máximo. 97
Tabla 11. Gestión de la memoria: Memoria usada, reservada, excesos y memoria temporal. 98
Tabla 12. Comparativa de funcionamiento de los métodos que aplican el Gradiente Conjugado (CG). ES y PS señala si la matriz de coeficientes es exactamente simétrica o prácticamente simétrica, respectivamente. 102
Tabla 13. Medición de horas de trabajo realizadas por el Ingeniero industrial. 111
Tabla 14. Medición de horas de trabajo realizadas por el Ingeniero Senior. 111
Tabla 15. Coste total de la mano de obra. 111
Tabla 16. Coste del Software y los equipos electrónicos. 112
Tabla 17. Presupuesto base de licitación. 112

AGRADECIMIENTOS

La realización de este Trabajo de Fin de Máster ha sido posible gracias al apoyo y la colaboración de numerosas personas a las que deseo expresar mi más profundo y sincero agradecimiento.

En primer lugar, quiero manifestar mi especial gratitud al profesor Luis Baeza González, cuyo liderazgo y experiencia han sido fundamentales durante todo este proceso. Su destacada trayectoria como catedrático e investigador en el ámbito ferroviario, así como su labor en el Instituto de Investigación en Ingeniería Mecánica y Biomecánica (I2MB) de la Universitat Politècnica de València, han aportado una perspectiva invaluable a este trabajo. Agradezco enormemente su orientación experta, su constante disponibilidad y la confianza depositada en mí, que han sido motores esenciales para superar los retos encontrados y alcanzar los objetivos propuestos.

A mis compañeros y amigos, gracias por estar siempre presentes brindando apoyo y compartiendo este camino conmigo. Vuestras palabras de aliento, colaboraciones y momentos compartidos han enriquecido tanto el proceso académico como el personal, convirtiendo cada obstáculo en una oportunidad de aprendizaje conjunto.

Deseo también expresar mi agradecimiento a mi psicólogo, cuyo acompañamiento ha sido clave para mantener el equilibrio y la perseverancia necesarios durante este periodo. Sus consejos y apoyo han contribuido significativamente a mi crecimiento personal y profesional.

A mi familia y a mi pareja, mi más profundo reconocimiento por su amor incondicional y su constante respaldo. Su comprensión, paciencia y ánimo han sido el sostén que me ha impulsado a seguir adelante incluso en los momentos más desafiantes. Este logro es reflejo de vuestro cariño y confianza inquebrantables.

Finalmente, a ese amigo tan especial que ha estado a mi lado en cada paso de este recorrido, gracias por tu presencia constante y tu apoyo desinteresado. Tu amistad ha sido una fuente de fortaleza y motivación que ha iluminado este camino. Espero que estés bien, allí donde estés.

A todos vosotros, gracias por formar parte de este viaje y por contribuir de manera tan significativa a la culminación de este proyecto. Este éxito es compartido y no habría sido posible sin vuestra invaluable contribución.

FINANCIACIÓN

Este trabajo ha sido posible gracias al generoso apoyo financiero proporcionado por la Agencia Estatal de Investigación. Quisiera expresar mi más sincero agradecimiento tanto a la Universitat Politècnica de València como al responsable científico, el profesor Luis Baeza González, por la confianza depositada en mí y por el respaldo económico brindado, los cuales han sido fundamentales para el inicio de esta investigación.

La financiación otorgada por la Agencia Estatal de Investigación ha permitido emprender este primer paso dentro del proyecto general destinado al desarrollo de modelos integrales para la simulación de la interacción dinámica del vehículo con el desvío, enmarcado en el plan de Proyectos I+D+i (Convocatoria 2020). El presente TFM, titulado ″Estudio comparativo de algoritmos de resolución del problema de contacto normal rueda-carril en ferrocarriles″, marca el inicio de este ambicioso proyecto, representando el primer hito en la ejecución de esta investigación.

Este contrato está asociado al proyecto PID2020-118013RB-C21, financiado por el Ministerio de Ciencia e Innovación (MCIN) y la Agencia Estatal de Investigación (AEI) bajo el código 10.13039/501100011033.

MEMORIA

ÍNDICE DE LA MEMORIA

1. INTRODUCCIÓN 13

1.1. Motivación 13
1.2. Objetivos del proyecto 13
1.3. Finalidad 14

2. CONTEXTO TEÓRICO 15

2.1. Introducción 15
2.2. Contacto rueda-carril 15
2.2.1. Introducción 15
2.2.2. Contacto de sólidos elásticos en rodadura 16
2.2.3. Fundamentos de la Mecánica de Contacto 18
2.2.3.1. Conformidad 18
2.2.3.2. Modelo elástico 19
2.2.3.2.1. Sistema de referencia 19
2.2.3.2.2. Formulación del problema elástico 19
2.2.3.2.3. Coeficientes de influencia para carga uniforme sobre un semiespacio 20
2.2.4. Modelo cinemático 22
2.2.4.1. Cinemática del solido elástico 22
2.2.4.2. Cinemática del contacto normal 23
2.2.4.3. Cinemática del contacto tangencial 24
2.2.4.3.1. Cálculo de los Pseudodeslizamientos 25
2.2.4.3.2. Cálculo del spin 26
2.2.4.3.3. Cálculo de la velocidad asociada a la configuración indeformada 27
2.2.5. Cuasiidentidad elástica 27
2.2.6. Teorías de contacto rueda carril 28
2.2.6.1. Introducción 28
2.2.6.2. Clasificación de teorías según Kalker 29
2.2.6.3. Método CONTACT 29
2.2.6.3.1. Discretización Espacial 30
2.2.6.3.2. Hipótesis de Cuasiidentidad Elástica 30
2.2.6.3.3. NORM 31
2.2.6.3.4. TANG 32
2.2.6.4. Modelo normal de Hertz 33
2.2.6.4.1. Suposiciones del modelo de Hertz 34
2.2.6.4.2. Parámetros y variables del problema 34
2.2.6.4.3. El modelo de Hertz en aplicaciones ferroviarias 37
2.2.6.5. Teoría lineal de Kalker 38
2.2.6.6. Algoritmo FASTSIM 40
2.3. Fundamentos teóricos de los algoritmos a implementar 43
2.3.1. Introducción 43
2.3.2. NORM de CONTACT 43
2.3.3. Aproximación de la distribución de tensiones en el contacto mediante una base elíptica o polinómica. .. 45
2.3.4. Método del gradiente conjugado con restricciones acotadas 48
2.4. Conclusión 52

3. METODOLOGÍA 54

3.1. Introducción 54
3.2. Implementación de los algoritmos 54

3.2.1. Objetivos de la implementación 54
3.2.2. Estrategias de optimización y criterios en la implementación 55
3.2.3. Metodología de implementación 56
3.2.4. Implementación de los algoritmos 57
3.2.4.1. Introducción 57
3.2.4.2. Preámbulo 58
3.2.4.2.1. Gestión de la memoria 59
3.2.4.2.2. Eliminación de Gauss con sustitución regresiva y pivotaje parcial 60
3.2.4.2.3. Método del Gradiente Conjugado 62
3.2.4.3. NORM 65
3.2.4.3.1. Implementación 65
3.2.4.3.2. Variantes implementadas 69
3.2.4.4. NORM + E/P 69
3.2.4.4.1. Implementación de la distribución elíptica de tensiones 70
3.2.4.4.2. Implementación de la distribución polinómica de tensiones 73
3.2.4.4.3. Particularidades 75
3.2.4.4.4. Variantes implementadas 76
3.2.4.5. BCCG(K) 76
3.2.4.6. Generalidades 82
3.3. Validación de la implementación 83
3.4. Conclusiones 84

4. EJECUCIÓN Y BENCHMARKING DE LOS ALGORITMOS 86

4.1. Introducción 86
4.2. Hardware utilizado 86
4.3. Escenario de contacto 87
4.3.1. Parámetros del material 87
4.3.2. Discretización y geometría del área potencial de contacto 87
4.4. Ejecución de los algoritmos 88
4.4.1. Argumentos de entrada 88
4.4.2. Salidas de los algoritmos 89
4.5. Extracción de datos para el análisis de los algoritmos 90
4.6. Análisis comparativo de los algoritmos (*benchmarking*) 92
4.6.1. Introducción 92
4.6.2. Tiempo de ejecución 93
4.6.3. Precisión de los resultados 96
4.6.4. Gestión de la memoria 98
4.7. Conclusiones 99

5. DISCUSIÓN DE LOS RESULTADOS 101

6. CONCLUSIONES Y FUTUROS PROYECTOS 105

7. BIBLIOGRAFÍA 107

1. INTRODUCCIÓN

1.1. Motivación

A lo largo del proceso hacia la obtención del Título Universitario de Graduado en Ingeniería en Tecnologías Industriales y el de Ingeniero Superior, a través del Máster Universitario en Ingeniería Industrial, se ha desarrollado una sólida capacidad para observar, razonar y resolver problemas en diversos ámbitos de la ingeniería. Este periodo académico, junto con la experiencia adquirida a través de la colaboración en proyectos de investigación en la UPV, específicamente en el AI2 (Instituto Universitario de Automática e Informática Industrial) y el I2MB (Instituto de Ingeniería Mecánica y Biomecánica), ha permitido al autor definir sus objetivos futuros y descubrir sus verdaderas pasiones.

Este Trabajo de Fin de Máster (en adelante, TFM) se enfoca en tres áreas clave que reflejan los principales intereses y proyecciones de futuro del autor: la programación de algoritmos y modelos para simulación y resolución de problemas mecánicos; la ingeniería mecánica; y la investigación. Con este proyecto, se busca combinar y profundizar en estos intereses a través de un desafío concreto: analizar el rendimiento de varios algoritmos para la resolución del problema de contacto normal rueda-carril sin rozamiento en ferrocarriles.

El proyecto se desarrolla bajo la dirección y asesoramiento del Sr. Luis Baeza González, catedrático e investigador de renombre en el campo ferroviario. El Sr. Baeza, destacado miembro del Instituto de Investigación en Ingeniería Mecánica y Biomecánica (I2MB) y profesor del Departamento de Ingeniería Mecánica y Materiales de la Universitat Politècnica de València, es una figura clave en el estudio de la dinámica ferroviaria. Su vasta experiencia y sus contribuciones al conocimiento en este ámbito han sido fundamentales para la realización y el éxito de este trabajo académico.

1.2. Objetivos del proyecto

El objetivo principal de este proyecto es implementar diversos algoritmos, en lenguaje Fortran, para la resolución del problema de contacto normal entre rueda y carril sin rozamiento, y realizar un análisis exhaustivo para comparar su rendimiento. Este análisis permitirá evaluar los algoritmos en términos de tiempo de ejecución, precisión y eficiencia global, con el fin de identificar las soluciones más competitivas desde el punto de vista computacional. Este enfoque permite profundizar en las áreas clave del proyecto, que combinan la programación de algoritmos, la ingeniería mecánica aplicada al sector ferroviario y la investigación.

Más concretamente, los objetivos específicos del proyecto son los siguientes:

- Implementar en lenguaje Fortran los siguientes algoritmos para el modelado y resolución del contacto normal sin rozamiento entre rueda y carril:
    1. El método NORM del software CONTACT (Kalker, 1990).
    2. Una aproximación basada en distribuciones de tensiones polinómicas o elípticas utilizando el método NORM como base (Giner Navarro, Gómez Bosch, Alonso, & Baeza, 2023).

3. El método *Enhanced Bound-Constrained Conjugate Gradient* o gradiente conjugado con restricciones acotadas (Vollebregt, 2013).

- Evaluar el rendimiento individual de cada algoritmo, optimizando el código para mejorar la eficiencia y reducir el tiempo de ejecución, de manera que los algoritmos sean competitivos en términos computacionales durante el posterior benchmarking.
- Llevar a cabo un benchmarking exhaustivo de los algoritmos implementados, comparando su rendimiento en términos de eficiencia (tiempo de CPU y uso de recursos) y eficacia (precisión de los resultados obtenidos), con el objetivo de identificar las soluciones óptimas y posibles puntos de mejora.
- Contribuir al desarrollo tecnológico en la ingeniería ferroviaria mediante soluciones más eficientes y menos demandantes en recursos computacionales, promoviendo la sostenibilidad en el sector.
- Explorar nuevas metodologías y enfoques innovadores para abordar la resolución del contacto normal entre la vía y las ruedas del tren, aportando avances en el campo de la dinámica ferroviaria.

1.3. Finalidad

El objetivo principal de este TFM es comparar y analizar exhaustivamente las prestaciones computacionales de los algoritmos propuestos para la resolución del problema de contacto normal sin rozamiento entre la rueda del tren y la vía, mediante un proceso de benchmarking. Para ello, se evaluará cada algoritmo en términos de eficacia (precisión de los resultados) y eficiencia (tiempo de CPU y uso de recursos), utilizando implementaciones optimizadas en Fortran, asegurando que las soluciones sean competitivas desde el punto de vista computacional.

Además, el proyecto busca profundizar en la comprensión teórica del contacto rueda-carril, con especial énfasis en el contacto normal, explicando las hipótesis y fundamentos que sustentan los algoritmos implementados. Este contexto teórico será clave para entender su aplicabilidad en problemas prácticos relacionados con la dinámica ferroviaria.

A lo largo del desarrollo, se analizará cómo las diferencias en las estrategias de implementación afectan el rendimiento computacional de los algoritmos. Este análisis permitirá no solo identificar las fortalezas y debilidades de cada enfoque, sino también optimizar las soluciones para adaptarlas a las necesidades específicas de las aplicaciones ferroviarias.

Finalmente, el proyecto ofrecerá una visión integral y actualizada sobre la modelización y resolución del problema de contacto rueda-carril, reflejando el conocimiento adquirido por el autor a lo largo del proceso. Con ello, se espera contribuir de manera significativa al avance en esta área de investigación, especialmente en el desarrollo de soluciones más eficientes y precisas para su aplicación en la ingeniería ferroviaria.

2. CONTEXTO TEÓRICO

2.1. Introducción

Tras haber presentado la motivación, los objetivos y la finalidad del trabajo en el capítulo anterior, este capítulo se centrará en los fundamentos teóricos del contacto rueda-carril, un aspecto clave en la dinámica ferroviaria.

Se abordarán los principios fundamentales del contacto de sólidos elásticos en rodadura y la mecánica de contacto, con especial énfasis en la conformidad y el modelo elástico utilizado para describir la interacción entre la rueda y el carril. Posteriormente, se analizará el comportamiento cinemático del contacto, diferenciando las componentes normal y tangencial. Se prestará particular atención al cálculo de los pseudodeslizamientos, el spin y las velocidades asociadas, elementos esenciales para comprender los movimientos relativos en la zona de contacto.

El capítulo también revisará el concepto de cuasiidentidad elástica, y se introducirá la clasificación de Kalker, que organiza las distintas teorías sobre el contacto rueda-carril. A partir de esta clasificación, se profundizará en las teorías más relevantes, como el método CONTACT, en sus versiones NORM y TANG, así como el modelo normal de Hertz, la teoría lineal y el algoritmo FASTSIM, todos ellos fundamentales para describir con precisión esta interacción.

Finalmente, se expondrán los fundamentos de los algoritmos que se implementarán en este trabajo, centrados en la resolución del problema de contacto normal. Entre ellos se encuentran el método NORM, la aproximación basada en distribuciones de tensiones elípticas o polinómicas, y el Gradiente Conjugado con restricciones acotadas, que servirán como base para el análisis numérico de los capítulos posteriores.

2.2. Contacto rueda-carril

2.2.1. Introducción

El problema de contacto entre la rueda y el carril se modela mediante las Teorías de Contacto, formulaciones que relacionan las magnitudes cinemáticas de la rueda con las fuerzas transmitidas en contacto con la vía. Las Teorías de Contacto son un componente esencial (aunque no el único) en el modelado dinámico de un vehículo ferroviario, permitiendo el estudio de cuestiones como la estabilidad, el comportamiento en curvas, la respuesta a irregularidades en la vía, la interacción entre el vehículo y la vía, etc.

La resolución del problema de contacto se lleva a cabo siguiendo la siguiente secuencia de pasos (que siguen el esquema de la Figura 1):

1. Se determinan las posiciones y velocidades relativas al carril del punto de contacto, a partir de las coordenadas generalizadas del eje montado.
2. Se resuelve el problema normal de contacto, determinando el área de contacto entre la rueda y el carril y la distribución de tensiones normales existente.

3. Se procede a la resolución del problema tangencial de contacto, calculando la distribución de tensiones tangenciales en el contacto.
4. Se integran las tensiones en el área de contacto para calcular las fuerzas resultantes.

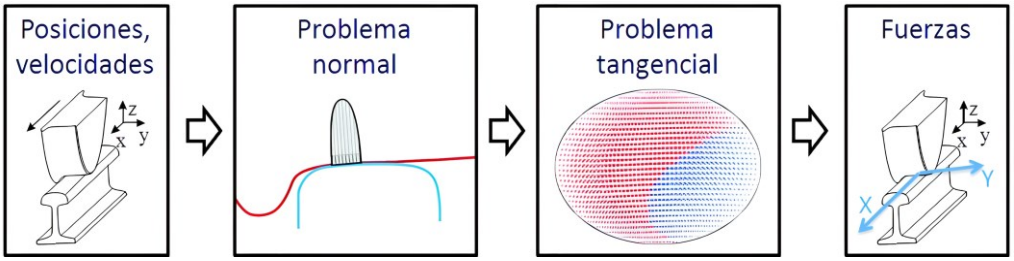

Figura 1. Secuencia de resolución del problema de contacto rueda-carril (Baeza, 2022).

Los principales avances en este ámbito se deben al profesor de la Universidad Técnica de Delft, Joost Jacques Kalker (1933-2006). El método CONTACT es uno de los modelos desarrollados por Kalker. Su sólido fundamento físico y la adopción de hipótesis avanzadas hacen que este modelo se utilice, a su vez, para validar otros modelos. El único inconveniente es el alto costo computacional que conllevan sus cálculos, lo que ha limitado su implementación en simulaciones hasta la actualidad.

Cabe destacar que el problema de contacto rueda-carril es complejo de abordar, lo cual ha condicionado también su difusión junto a la reducida claridad con que los investigadores de esta área han publicado los resultados. En este apartado, se pretende dar unas pinceladas teóricas que permitan entender al lector los fundamentos sobre los que este trabajo se desarrolla.

Para el desarrollo de esta contextualización teórica, se han tomado como referencia principal las notas proporcionadas por Baeza, L. (2022) sobre el contacto rueda-carril. No obstante, se han consultado y revisado otros documentos relevantes para complementar y enriquecer lo expuesto.

2.2.2. Contacto de sólidos elásticos en rodadura

El fenómeno del contacto en rodadura puede entenderse suponiendo una rueda girando a una velocidad angular ω sobre una superficie plana, mientras su centro se desplaza a una velocidad lineal V. Dado que la rueda es un sólido elástico, se formará un área de contacto aproximadamente elíptica entre ella y el plano de apoyo. A efectos ilustrativos y para facilitar la comprensión, en la rueda de la Figura 2 a) se disponen regularmente líneas radiales.

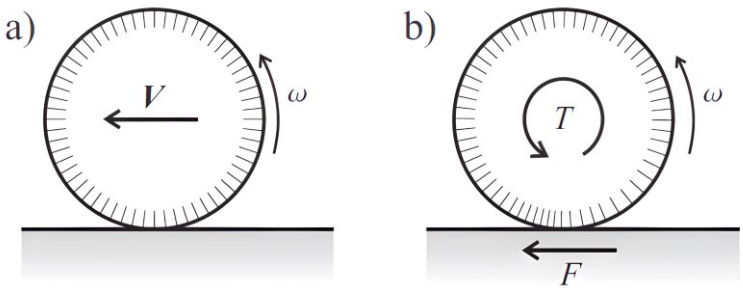

Figura 2. Pseudodeslizamiento (Baeza, 2022).

Si se aplica un par T, como ilustra la Figura 2 b), aparecerá una fuerza de fricción F en el punto de contacto a modo de resistencia a la rodadura. Debido a las tensiones tangenciales presentes en la zona de contacto, las líneas radiales se aproximan entre ellas delante del punto de contacto y se separan detrás del mismo. Al avanzar la rueda, el material alrededor de la zona de contacto pasa de estar comprimido a estar traccionado, lo que provoca la separación progresiva de las líneas radiales dentro del área de contacto. Esta deformación forma zonas locales de deslizamiento y adhesión.

Mayoritariamente, las partículas del cuerpo entran en la zona de contacto sin previo deslizamiento relativo y tienden a sufrirlo a medida que avanzan por el área de contacto, salen de ella y se separan. Esto genera un área de contacto en la que conviven una zona de adhesión (sin deslizamiento relativo entre las partículas de los cuerpos en contacto) y otra de deslizamiento. Este fenómeno se conoce como pseudodeslizamiento (del francés *pseudoglissement* y *creepage* en inglés), y se traduce macroscópicamente en una velocidad no nula del punto de contacto. Cuanto mayor es la fuerza transmitida, mayor es la deformación que se genera en el contacto, el área de deslizamiento y la velocidad aparente (o velocidad de pseudodeslizamiento) del punto de contacto. El límite de fuerza trasmitida se encuentra en μN, situación en la que la totalidad del área de contacto presentaría deslizamiento.

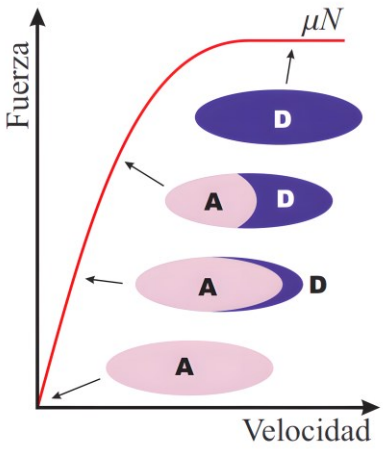

Figura 3. Relación de la Fuerza tangencial con la velocidad del punto de contacto en un modelo de contacto 2D (Baeza, 2022).

La relación entre la velocidad de pseudodeslizamiento y la fuerza tangencial transmitida en el contacto se muestra en la Figura 3. Cuando la fuerza tangencial en el contacto es nula se cumple la condición de rodadura sin deslizamiento y el punto de contacto tiene velocidad nula, lo que implica que todos los puntos del área de contacto están en condición de adhesión (zona A). A medida que la fuerza tangencial aumenta en la zona de contacto, la velocidad de pseudodeslizamiento crece y se desarrolla una zona de deslizamiento en la parte posterior del área de contacto (zona D). Puede observarse como la curva tiende al límite de fuerza tangencial μN, situación en la que toda la zona de contacto presenta deslizamiento.

Las Teorías de Contacto pueden entenderse como un sistema en el que los parámetros cinemáticos del contacto actúan como entrada, mientras que las fuerzas resultantes transmitidas en el contacto rueda-carril son la salida. Estas teorías formulan leyes, como la mostrada en la Figura 3,

que permiten vincular las fuerzas tangenciales generadas en el contacto con la velocidad de pseudodeslizamiento. La dificultad aparece al ser necesario considerar el problema en sus tres dimensiones, pues existen fuerzas y velocidades de pseudodeslizamiento tanto en la dirección de rodadura, como en la dirección lateral.

2.2.3. Fundamentos de la Mecánica de Contacto

2.2.3.1. Conformidad

En el caso del contacto rueda-carril, pueden originarse dos tipos de contacto: el conforme y el no conforme. El contacto conforme se produce cuando el tamaño del área de contacto es similar al de los radios de curvatura de las superficies en contacto. En el caso del contacto no-conforme, los radios de curvatura son bastante mayores que las dimensiones del área de contacto. Estas condiciones de contacto pueden observarse en la Figura 4.

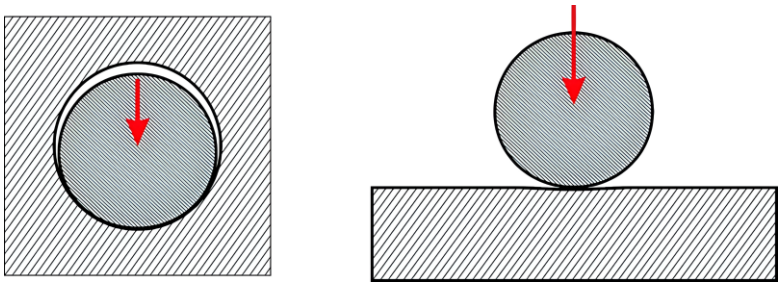

Figura 4. Contacto conforme (izquierda) y no-conforme (derecha) (Baeza, 2022).

Al observar el contacto entre la rueda y el carril, como se muestra en la Figura 5, se puede deducir que el contacto es no conforme, ya que el área de contacto suele estar entre 1 y 4 cm. Aunque esto puede deducirse fácilmente, existen situaciones en las que se pueden dar condiciones de conformidad, especialmente cuando el eje se desplaza lateralmente y el contacto ocurre, por ejemplo, entre la pestaña y la banda de rodadura en el perfil de la rueda. El contacto conforme sólo puede ser modelado con precisión mediante el método de los Elementos Finitos (MEF).

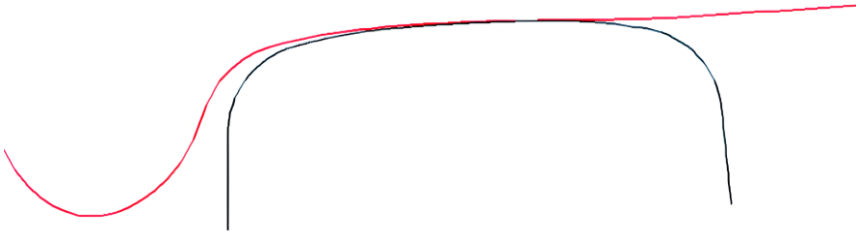

Figura 5. Perfil S1002 (eje centrado en vía) sobre carril UIC 60 inclinado 1/20 (Baeza, 2022).

Las principales implicaciones de un contacto no conforme son:

- Debido a que las superficies en contacto son prácticamente paralelas, es posible definir una dirección normal y un plano tangente en la zona de contacto.

- Cuando un punto de la rueda entra en contacto con un punto del carril bajo una carga normal específica, ambos puntos estarán alineados a lo largo de una dirección perpendicular al contacto.
- Si las propiedades mecánicas de los materiales son similares, se cumplen las condiciones de cuasiidentidad elástica (véase el punto 2.2.5). Esto permite aislar el problema normal de contacto del problema tangencial de contacto.
- Se puede suponer que los sólidos se comportan elásticamente como semiespacios infinitos, es decir, como medios elásticos que se extienden indefinidamente por debajo de la superficie de contacto.

2.2.3.2. Modelo elástico

Se considerará el modelo de contacto no conforme, donde los sólidos se comportan elásticamente como semiespacios infinitos. Para modelar este problema elástico, es necesario establecer un sistema de referencia que permita definir las variables involucradas en el problema.

2.2.3.2.1. Sistema de referencia

Como sistema de referencia se toma un sistema inercial $\boldsymbol{X_1X_2X_3}$ que se desplaza a la misma velocidad del vehículo, denotada como V. El origen del sistema de referencia se ubica en el punto teórico de contacto, es decir, donde se supone que ocurriría el contacto entre los perfiles de la rueda y el carril en el caso de que estos fuesen rígidos. El eje X_1 está alineado con la dirección de rodadura, mientras que el eje X_3 es perpendicular al plano de contacto. Aunque resulta crucial definir correctamente la dirección del eje X_1, la orientación del eje X_3 en sentido positivo ascendente se considera arbitraria, como se ilustra en la Figura 6.

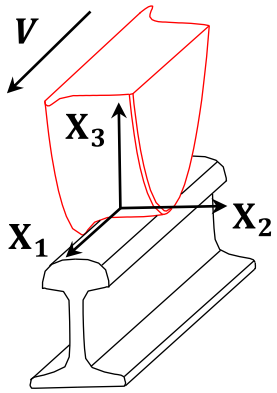

Figura 6. Sistema de referencia empleado en el contacto (Baeza, 2022).

2.2.3.2.2. Formulación del problema elástico

Se define $p_j(x_1, x_2)$ como la tensión en el punto (x_1, x_2), según la dirección X_j. Asimismo, se define $u_j(x_1, x_2)$ como el campo de desplazamientos correspondientes a las deformaciones originadas en el contacto.

El problema normal permite calcular el área de contacto a partir de las tensiones p_3 y los desplazamientos u_3, ambos normales al plano de contacto. Por otro lado, el problema tangencial se

relaciona con las tensiones y desplazamientos p_1, p_2, u_1 y u_2. Las magnitudes contenidas en el plano tangencial se denotarán como $p_\tau, u_\tau \; y \; x_\tau$ (donde τ toma los valores 1 o 2). Además, se concibe innecesaria la consideración de puntos fuera del plano de contacto, por lo que se asume $x_3 = 0$. De este modo, se puede definir $\boldsymbol{x} = (x_1, x_2)$ y, en consecuencia, $p_j = p_j(\boldsymbol{x})$ y $u_j = u_j(\boldsymbol{x})$.

La relación entre el campo de tensiones y el de deformaciones se fundamenta en la aplicación de la Teoría de la Elasticidad. Generalmente, los modelos elásticos que relacionan tensiones y deformaciones en la zona de contacto son utilizados para determinar los coeficientes de influencia $A_{ij}(\boldsymbol{x}, \boldsymbol{y})$. Estos coeficientes se definen como el desplazamiento ocasionado en el punto $\boldsymbol{x}$ en la dirección i cuando se aplica una tensión unitaria en la dirección j del punto $\boldsymbol{y}$. Así, mediante la aplicación del principio de superposición, el campo de desplazamientos en la zona de contacto puede calcularse a partir de la integral extendida sobre los límites del área de contacto ∂C:

$$u_i(\boldsymbol{x}) = \sum_{j=1}^{3} \int_{\partial C} A_{ij}(\boldsymbol{x}, \boldsymbol{y}) \cdot p_j(\boldsymbol{y}) \cdot dS(\boldsymbol{y}) \tag{1}$$

En la realidad, existe una cierta interacción entre los problemas tangencial y normal. Sin embargo, debido a las propiedades de cuasiidentidad elástica, no es relevante estudiar la influencia de las tensiones normales sobre los desplazamientos tangenciales, ni viceversa. Como resultado, la formulación del campo de desplazamientos normales resulta:

$$u_3(\boldsymbol{x}) = \int_{\partial C} A_{33}(\boldsymbol{x}, \boldsymbol{y}) \cdot p_3(\boldsymbol{y}) \cdot dS(\boldsymbol{y}) \tag{2}$$

y para el problema tangencial:

$$u_\tau(\boldsymbol{x}) = \sum_{j=1}^{2} \int_{\partial C} A_{\tau j}(\boldsymbol{x}, \boldsymbol{y}) \cdot p_j(\boldsymbol{y}) \cdot dS(\boldsymbol{y}) \tag{3}$$

Nótese que $\int_{\partial C} dS$ denota la integral extendida al área de contacto.

2.2.3.2.3. Coeficientes de influencia para carga uniforme sobre un semiespacio

Cuando se da la condición de no conformidad en el contacto entre la rueda y el carril, los desplazamientos que ocurren cerca del área de contacto son similares a los que se generarían en un semiespacio infinito. La ecuación (1) particularizada para un semiespacio infinito se conoce como *integral de Boussinesq-Cerruti*, y tiene soluciones analíticas para ciertos tipos de distribuciones de carga. Un ejemplo de estos casos es un rectángulo sometido a una presión constante.

Considérese un semiespacio infinito que ocupa el espacio donde $x_3 < 0$. En este medio, existe un rectángulo de dimensiones $2a \times 2b$ sometido a una tensión constante p_3. Este rectángulo está centrado en el origen ($\boldsymbol{x} = 0$), y las dimensiones $2a$ y $2b$ corresponden a las direcciones $\boldsymbol{X_1}$ y $\boldsymbol{X_2}$,

respectivamente. El desplazamiento en la dirección $\boldsymbol{X_3}$ en la superficie del semiespacio infinito se calcula mediante la expresión:

$$\begin{aligned} u_3(x_1,x_2) = \frac{p_3(1-\nu^2)}{\pi E}\Bigg[&(x_1+a)\log\left(\frac{(x_2+b)\sqrt{(x_1+a)+(x_2+b)}}{(x_2-b)\sqrt{(x_1+a)+(x_2-b)}}\right)+\\ &+\ (x_2+b)\log\left(\frac{(x_1+a)\sqrt{(x_1+a)+(x_2+b)}}{(x_1-a)\sqrt{(x_1-a)+(x_2+b)}}\right)+\\ &+\ (x_1-a)\log\left(\frac{(x_2-b)\sqrt{(x_1-a)+(x_2-b)}}{(x_2+b)\sqrt{(x_1-a)+(x_2+b)}}\right)+\\ &+\ (x_2-b)\log\left(\frac{(x_1-a)\sqrt{(x_1-a)+(x_2-b)}}{(x_1+a)\sqrt{(x_1+a)+(x_2-b)}}\right)\Bigg], \end{aligned} \tag{4}$$

donde E es el módulo de Young, ν el coeficiente de Poisson, y el operador $\log$ corresponde al logaritmo natural.

Supóngase ahora el caso en que, en lugar de aplicar una carga normal, se aplican tensiones tangenciales sobre el rectángulo: p_1 en la dirección $\boldsymbol{X_1}$ y p_2 en la dirección $\boldsymbol{X_2}$. El desplazamiento según $\boldsymbol{X_1}$ en la superficie del semiespacio se calcula mediante:

$$u_1(x_1,x_2) = \frac{p_1(1+\nu)}{2\pi E}\big(\mathcal{J}_1+\mathcal{J}_2(1-\nu)\big)+\frac{p_2(1+\nu)\nu}{2\pi E}\mathcal{J}_4, \tag{5}$$

donde:

$$\begin{aligned} \mathcal{J}_1 = (x_2+b)\cdot\log\left(\frac{(x_1+a)\sqrt{(x_1+a)^2+(x_2+b)^2}}{(x_1-a)\sqrt{(x_1-a)^2+(x_2+b)^2}}\right)+(x_2-b)\cdot\\ \cdot\log\left(\frac{(x_1-a)\sqrt{(x_1-a)^2+(x_2-b)^2}}{(x_1+a)\sqrt{(x_1+a)^2+(x_2-b)^2}}\right), \end{aligned} \tag{6}$$

$$\begin{aligned} \mathcal{J}_2 = (x_1+a)\cdot\log\left(\frac{(x_2+b)\sqrt{(x_1+a)^2+(x_2+b)^2}}{(x_2-b)\sqrt{(x_1+a)^2+(x_2-b)^2}}\right)+(x_1-a)\cdot\\ \cdot\log\left(\frac{(x_2-b)\sqrt{(x_1-a)^2+(x_2-b)^2}}{(x_2+b)\sqrt{(x_1-a)^2+(x_2+b)^2}}\right), \end{aligned} \tag{7}$$

$$\begin{aligned} \mathcal{J}_4 = \sqrt{(x_1-a)^2+(x_2+b)^2}-\sqrt{(x_1+a)^2+(x_2+b)^2}-\sqrt{(x_1-a)^2+(x_2-b)^2}\\ +\sqrt{(x_1+a)^2+(x_2-b)^2}. \end{aligned} \tag{8}$$

El cálculo de $u_2(x_1, x_2)$ a partir de las ecuaciones (6), (7) y (8) es directo. Estas fórmulas son útiles para obtener las relaciones constitutivas correspondientes al contacto rueda-carril a través del método CONTACT.

2.2.4. Modelo cinemático

2.2.4.1. Cinemática del solido elástico

Para modelar la cinemática del contacto, se hace uso del concepto de *"configuración"*, comúnmente empleado en Mecánica del Medio Continuo (Mase, Mase, & Smelser, 2010). Este concepto es útil de cara a modelar tanto los desplazamientos como las velocidades de puntos en sólidos flexibles. Se toman tres configuraciones: referencia, deformada y no-deformada (véase la Figura 7).

La configuración de referencia se refiere a una posición arbitraria y fija (independiente del tiempo) del sólido no deformado. Usualmente, corresponde a aquella que ocupa el sólido sin deformar en el instante $t = 0$, aunque también puede definirse una posición que el sólido no puede ocupar físicamente.

La configuración deformada se asocia al espacio que ocupa el sólido flexible en un instante específico t. Entre la configuración de referencia y la deformada, se define una configuración intermedia llamada configuración no-deformada. Esta configuración no-deformada, al igual que la deformada, varía con el tiempo y se caracteriza porque la transformación de la configuración deformada a la no-deformada implica un pequeño desplazamiento.

Es importante destacar que la noción de configuración permite un análisis detallado de cómo los sólidos flexibles responden a fuerzas y desplazamientos en diferentes momentos. Al establecer una configuración de referencia y compararla con las configuraciones deformada y no deformada, se pueden estudiar las propiedades mecánicas y de deformación del material, así como su comportamiento bajo diversas condiciones de carga. Este enfoque es fundamental para comprender y predecir el comportamiento de materiales y estructuras en ingeniería y física aplicada.

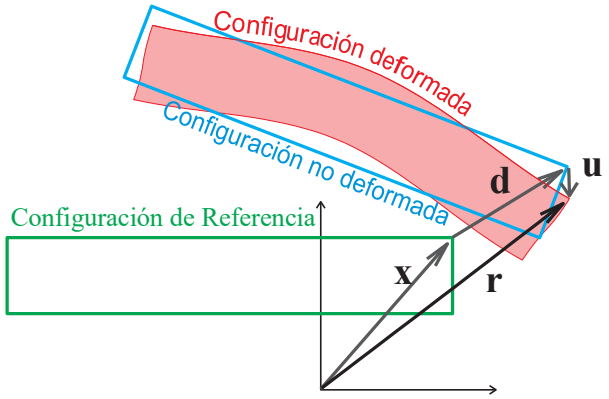

Figura 7. Configuraciones: Referencia, no-deformada y deformada (Baeza, 2022).

Siguiendo el esquema de la Figura 7, el vector posición de una partícula material que en la configuración de referencia se encuentra en la posición $\boldsymbol{x}$, se calcula de tal que:

$$\boldsymbol{r} = \boldsymbol{x} + \boldsymbol{d} + \boldsymbol{u}, \tag{9}$$

donde $\boldsymbol{d}$ corresponde al vector que describe el desplazamiento de sólido rígido desde la posición de referencia, y $\mathbf{u}$ es el vector de desplazamiento asociado a las deformaciones. Es fundamental considerar que los vectores $\boldsymbol{r}$, $\boldsymbol{d}$ y $\boldsymbol{u}$ dependen tanto del punto en la configuración de referencia $\boldsymbol{x}$ como del tiempo t.

El planteamiento anterior se fundamenta en la existencia de métodos específicos para modelar tanto la cinemática de cuerpos rígidos (ángulos de Euler, ecuación del campo de velocidades, …) como los pequeños desplazamientos correspondientes a deformaciones (elasticidad lineal, vibraciones, …). Estos métodos se utilizan para definir los vectores $\boldsymbol{d}$ y $\boldsymbol{u}$.

Además, la expresión (9) también se utiliza derivada con respecto al tiempo, obteniendo:

$$\dot{\boldsymbol{r}} = \dot{\boldsymbol{d}} + \dot{\boldsymbol{u}}. \tag{10}$$

2.2.4.2. Cinemática del contacto normal

Siguiendo la metodología descrita antes, se han establecido las configuraciones representadas en la Figura 8 para la rueda y el carril:

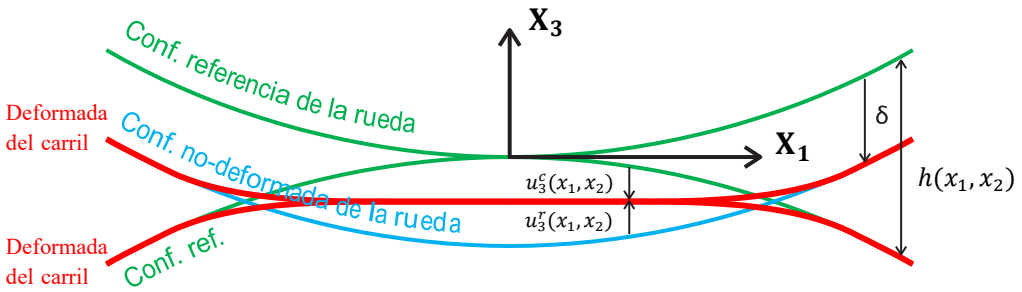

Figura 8. Configuración de referencia (verde), no-deformada (azul) y deformada (rojo), para el carril y la rueda (Baeza, 2022).

1. **Configuraciones de referencia:** Estas configuraciones corresponden a las posiciones de ambos sólidos (rueda y carril) supuestos rígidos y sin deformar cuando se tocan en un único punto, denominado punto teórico de contacto. Posicionado en el centro de este punto teórico de contacto, se ubica el sistema de referencia $\boldsymbol{X_1X_2X_3}$ asociado al carril, conforme al criterio establecido en el apartado 2.2.3.2.1. Se define una función $h(x_1, x_2)$ que mide la distancia entre las superficies de la rueda y del carril en la dirección $\boldsymbol{X_3}$. Esta función, conocida como función distancia entre superficies indeformadas, depende de la geometría de las superficies de la rueda y el carril, así como de su posición relativa.
2. **Configuraciones no deformadas:** En esta configuración la rueda se mueve como un sólido rígido y el carril permanece fijo. Por lo tanto, la configuración de referencia del carril y la no deformada de este, coinciden. En este caso, interesa el desplazamiento de la rueda en

dirección normal al plano de contacto, llamado interpenetración entre superficies no deformadas (o *approach*), representado por la variable δ.

3. **Configuraciones deformadas:** Esta configuración se refiere a la posición final del contacto entre la rueda y el carril. En esta etapa se deben considerar los desplazamientos asociados a las deformaciones en la dirección $\boldsymbol{X_3}$, tanto en la rueda (u_3^r) como en el carril (u_3^c).

Según la Tercera Ley de Newton, la distribución de tensiones en la rueda p_3^r será igual en magnitud, pero opuesta en signo a la distribución de tensiones en el carril p_3^c. Dado que tanto el carril como la rueda responden de la misma manera ante la misma acción (ya que, elásticamente, ambos se comportan como semiespacios infinitos), los desplazamientos resultantes de las deformaciones en la rueda u_3^r y en el carril u_3^c también serán iguales, pero con signos opuestos. Entonces, las magnitudes correspondientes al carril se pueden representar con u_3 y p_3 ($u_3 = u_3^c = -u_3^r$, y $p_3 = p_3^c = -p_3^r$). El esquema que detalla los desplazamientos de los puntos en contacto se presenta en la Figura 9.

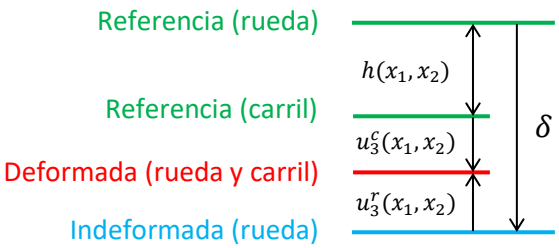

Figura 9. Configuración de referencia (verde), no-deformada (azul) y deformada (rojo), para el carril y la rueda (Baeza, 2022).

Es importante tener en cuenta que la variable δ siempre se define como positiva, por lo que, según el sistema de referencia utilizado, u_3^c debe tener valores negativos. De acuerdo con el esquema de la figura anterior, se cumple que:

$$\delta = h(x_1, x_2) - 2u_3(x_1, x_2). \tag{11}$$

Esta ecuación es válida únicamente en los puntos dentro del área de contacto, ya que en cualquier otro caso la interpenetración sería nula.

2.2.4.3. Cinemática del contacto tangencial

Considérense los puntos del carril P_c y de la rueda P_r en contacto, como se muestra en la Figura 10. La diferencia de velocidades entre estos puntos se denomina deslizamiento relativo entre ambas superficies, o velocidad de deslizamiento local $\boldsymbol{s}$:

$$\boldsymbol{s} = \dot{\boldsymbol{r}}^r - \dot{\boldsymbol{r}}^c, \tag{12}$$

donde r^c y r^r son los vectores de posición de los puntos en contacto, es decir, P_c y P_r.

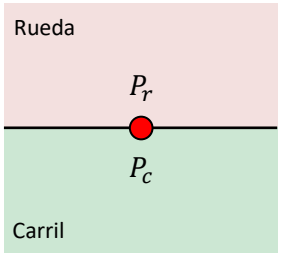

Figura 10. Puntos de contacto de la rueda y el carril (Baeza, 2022).

Según la ecuación (10), la velocidad de los cuerpos puede expresarse a través del movimiento de sólido rígido y los desplazamientos asociados a las deformaciones como:

$$\boldsymbol{s} = (\boldsymbol{w} + \dot{\boldsymbol{u}}^r) - \dot{\boldsymbol{u}}^c, \tag{13}$$

donde $\boldsymbol{w}$ es la velocidad asociada a la configuración indeformada.

Considerando que los desplazamientos debidos a las deformaciones en el carril son iguales, pero de signo opuesto a los de a la rueda, se establece que:

$$u_\tau = u_\tau^c = -u_\tau^r. \tag{14}$$

Así, la ecuación para el deslizamiento local s_τ es:

$$s_\tau = w_\tau - 2\dot{u}_\tau\,, \qquad para\ \tau = 1{,}2. \tag{15}$$

El cálculo de $\boldsymbol{w}$ se formula a partir de las expresiones de las velocidades de pseudodeslizamiento y de spin.

2.2.4.3.1. Cálculo de los Pseudodeslizamientos

Asumiendo que el eje montado es un cuerpo rígido, se puede calcular la velocidad asociada a la configuración no deformada del punto teórico de contacto. Este cálculo depende del tipo de coordenadas utilizadas. Usando coordenadas cartesianas, la velocidad indeformada del punto nominal de contacto $\boldsymbol{v}^O$ se obtiene mediante la ecuación del campo de velocidades:

$$\boldsymbol{v}^O = \boldsymbol{v}^G + \boldsymbol{\omega}^{eje} \times \boldsymbol{r}^{GO}, \tag{16}$$

donde $\boldsymbol{v}^G$ es la velocidad del centro de masas del eje montado, $\boldsymbol{\omega}^{eje}$ la velocidad angular del eje montado y $\boldsymbol{r}^{GO}$ el vector de posición que conecta el centro de masas con el punto teórico de contacto. En esta ecuación, el símbolo × denota el producto vectorial. Las velocidades definidas aquí se toman como absolutas e incluyen, además de los movimientos de avance, los movimientos laterales y verticales, así como los giros de roll y lazo, entre otros.

El vector $\boldsymbol{v}^G$ debe representarse en el sistema de coordenadas $\boldsymbol{X_1X_2X_3}$. Por lo tanto, la componente v_3^O será nula si se usa un modelo de contacto basado en restricciones cinemáticas, o muy pequeña en el caso de emplear una formulación de penalti. Las componentes v_1^O y v_2^O se interpretan

como velocidades de pseudodeslizamiento en las direcciones longitudinal y lateral, respectivamente. Con frecuencia, se emplean las magnitudes reducidas:

$$\xi_\tau = \frac{v_\tau^O}{V}, \tag{17}$$

donde V representa la velocidad del vehículo. Los valores de v_τ^O son aproximadamente el 0,1% de la velocidad del vehículo.

2.2.4.3.2. Cálculo del spin

En la Figura 11 a) se muestran los perfiles en contacto, el área de contacto (marcada en rojo) y tres radios de diferentes posiciones de la huella: r_a, r_b y r, donde r corresponde al radio en el punto teórico de contacto. Debido a la conicidad de la rueda, se cumple que $r_a > r > r_b$.

En una situación sin pseudodeslizamiento (es decir, $V - \omega \cdot r = 0$), $V - \omega \cdot r_a$ y $V - \omega \cdot r_b$ no serían iguales a cero. Esto sugiere que el punto con radio r_a presenta una velocidad lineal hacia el interior de la figura, mientras que el punto con radio r_b se mueve hacia el exterior. Como resultado, la superficie de contacto de la rueda gira sobre la del carril, fenómeno conocido como *spin*.

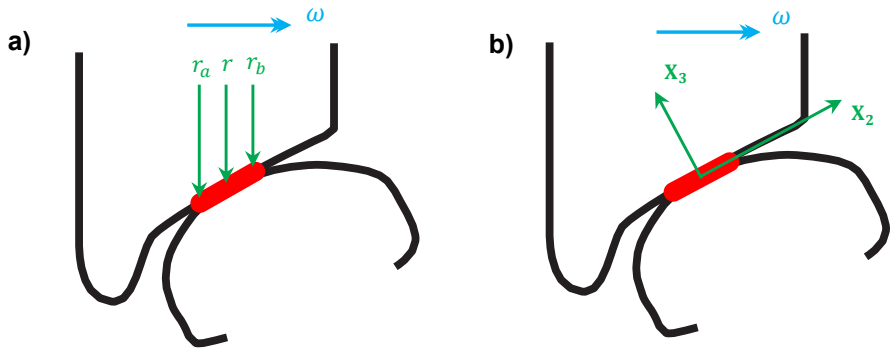

Figura 11. Spin de la superficie de contacto (Baeza, 2022).

El cálculo de la velocidad angular de giro de la superficie de la rueda sobre la del carril, puede deducirse mediante la Figura 11 b). La velocidad angular de spin ω^{sp}, se determina como el producto escalar entre la velocidad angular del eje y el versor normal unitario asociado al eje $\boldsymbol{X_3}$:

$$\omega^{sp} = \boldsymbol{\omega}^{eje} \cdot \boldsymbol{i}_3. \tag{18}$$

El spin se origina principalmente debido a la conicidad de la rueda, y sería nulo si las ruedas fueran completamente cilíndricas. Así, el spin es la componente normal al contacto del vector de velocidad angular absoluta del eje montado ($\boldsymbol{\omega}^{eje}$).

Al igual que con la velocidad de pseudodeslizamiento, el spin se normaliza con respecto a la velocidad del vehículo:

$$\xi_{sp} = \frac{\omega^{sp}}{V}, \tag{19}$$

A diferencia de ξ_1 y ξ_2 , ξ_{sp} tiene unidades de rad/m.

2.2.4.3.3. Cálculo de la velocidad asociada a la configuración indeformada

En la Figura 12 se esquematiza el área de contacto (zona amarilla), el sistema $\boldsymbol{X_1X_2X_3}$, las velocidades de pseudodeslizamiento (v_1^O y v_2^O) y el spin (ω^{sp}). Se representa también el punto P de coordenadas x_1 y x_2, para el cual se va a calcular la velocidad asociada a la configuración no deformada $\boldsymbol{w}$.

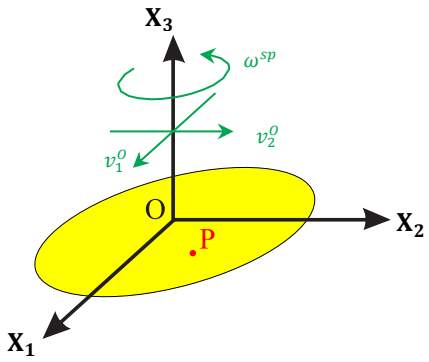

Figura 12. Cálculo de la velocidad asociada a la configuración indeformada $\boldsymbol{w}$ (Baeza, 2022).

Aplicando la ecuación del campo de velocidades, las componentes de $\boldsymbol{w}$ serán:

$$\begin{aligned} w_1 &= v_1^0 - \omega^{spin} x_2 = V\xi_1 - V\xi_{sp}x_2 \\ w_2 &= v_2^0 + \omega^{spin} x_1 = V\xi_2 + V\xi_{sp}x_1 \,. \end{aligned} \tag{20}$$

2.2.5. Cuasiidentidad elástica

La condición de cuasiidentidad elástica se satisface cuando dos cuerpos elásticos reaccionan de manera idéntica ante las mismas fuerzas, mostrando los mismos desplazamientos bajo las mismas condiciones de carga. Para que esta condición se cumpla, es necesario que ambos cuerpos se comporten como semiespacios infinitos y que posean las mismas propiedades mecánicas, tales como el módulo de Young y el coeficiente de Poisson.

Las características de la cuasiidentidad elástica incluyen:

- **Desplazamientos normales independientes de las tensiones tangenciales existentes:** En el contexto del problema de contacto normal, los desplazamientos normales no se formulan considerando las tensiones tangenciales. Esto significa que las fuerzas tangenciales aplicadas no alteran los desplazamientos perpendiculares al contacto.
- **Desplazamientos tangenciales independientes de las tensiones normales existentes:** De manera similar, en el problema de contacto tangencial, las tensiones normales no se incluyen en la formulación de los desplazamientos tangenciales. Esto implica que las fuerzas normales no afectan los desplazamientos paralelos en el contacto.
- **Desacoplamiento de los problemas Normal y Tangencial:** Las condiciones tangenciales no inciden sobre el problema normal de contacto. Esto permite resolver primero el problema

normal para determinar el área de contacto y, posteriormente, abordar el problema tangencial utilizando la información obtenida. Este enfoque secuencial simplifica el proceso de resolución.

Para ilustrar estas propiedades, se puede utilizar un esquema conceptual de los dos cuerpos en contacto, como el de la Figura 13. Supóngase que, al aplicar el principio de liberación entre ambos cuerpos, solo se transmite la fuerza F, tal como muestra la Figura 13 b). Las deformaciones normales al contacto debidas a F se representan en la Figura 13 c). Dado que ambos cuerpos se comportan elásticamente de manera idéntica, se deforman en formas conjugadas. Esto significa que F no modifica la distribución de tensiones normales en el contacto.

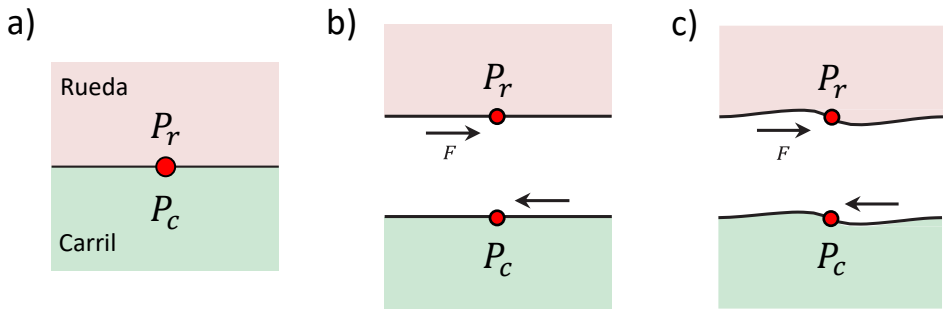

Figura 13. Propiedades de cuasiidentidad elástica (Baeza, 2022).

En resumen, la cuasiidentidad elástica permite el desacoplamiento de los problemas normal y tangencial en el análisis de contacto rueda-carril, facilitando el cálculo de las tensiones y desplazamientos involucrados.

2.2.6. Teorías de contacto rueda carril

2.2.6.1. Introducción

El contacto entre rueda y carril es un aspecto fundamental en la dinámica de los vehículos ferroviarios, ya que influye directamente en la estabilidad, la capacidad de tomar curvas, la respuesta del vehículo a las irregularidades de la vía, etc. Estas interacciones no solo afectan a la seguridad y el confort de los pasajeros, sino que también desempeñan un papel crucial en la eficiencia energética y en el desgaste de los componentes del sistema ferroviario. Por esta razón, es esencial contar con modelos precisos que permitan predecir el comportamiento de este contacto bajo diferentes condiciones operativas.

Para modelar y analizar estas interacciones, se han desarrollado diversas teorías de contacto que permiten relacionar las magnitudes cinemáticas de la rueda con las fuerzas transmitidas en el punto de contacto. Estas teorías ofrecen una comprensión detallada de cómo las fuerzas y los desplazamientos se distribuyen en la zona de contacto, lo que es vital para optimizar el diseño y el mantenimiento de los vehículos y la infraestructura ferroviaria.

En esta sección se revisan las teorías de contacto más reconocidas, incluyendo el método CONTACT, la teoría de Hertz, la teoría lineal y el algoritmo FASTSIM. Estas metodologías han sido ampliamente utilizadas y han demostrado ser efectivas en la resolución del problema de contacto.

2.2.6.2. Clasificación de teorías según Kalker

En la literatura se han publicado diversas metodologías para resolver el problema de contacto entre la rueda y el carril. Kalker propuso una clasificación de estas teorías basándose en distintos factores. A continuación, se presenta un resumen de dicha clasificación.

- **Teorías exactas:** Modelan la relación entre las tensiones y los desplazamientos en el contacto utilizando Elementos Finitos o considerando que los sólidos se comportan como semiespacios infinitos.
- **Teorías simplificadas:** La relación entre tensiones y desplazamientos en el contacto se modela bajo hipótesis simplificadas. Un ejemplo de ello es el modelo de Winkler, en el que los desplazamientos en un punto dependen únicamente de las tensiones en ese punto.
- **Teorías dinámicas:** Tienen en cuenta los efectos inerciales asociados al contacto, aunque hasta el momento no se ha desarrollado ninguna teoría concreta en este ámbito.
- **Teorías cuasiestáticas:** Desprecian los efectos inerciales asociados al contacto, centrando el análisis en condiciones cuasiestáticas.
- **Teorías estacionarias:** No considera la influencia del tiempo en el proceso de contacto; la distribución de tensiones depende únicamente de las velocidades de pseudodeslizamiento.
- **Teorías no estacionarias:** Incorporan la dependencia del tiempo en el proceso de contacto.
- **Teorías bidimensionales:** Se enfocan exclusivamente en estudiar las direcciones $\boldsymbol{X_1}$ y $\boldsymbol{X_3}$.
- **Teorías tridimensionales:** Consideran el caso real, abordando las tres direcciones ortogonales en el análisis del contacto.

Además de la clasificación general de las teorías de contacto rueda-carril, es fundamental destacar los modelos hercianos, basados en la teoría de Hertz, los cuales son ampliamente utilizados para resolver el problema de contacto normal. Estos modelos tienen una gran relevancia en la industria ferroviaria debido a su capacidad para modelar con precisión tanto la distribución de tensiones como el área de contacto entre la rueda y el carril.

A continuación, se detallará el método CONTACT, seguido de una revisión de otros modelos importantes como el modelo normal de Hertz, la teoría lineal y el algoritmo FASTSIM. Aunque algunos de estos modelos no serán utilizados directamente en este trabajo, proporcionan un contexto esencial para comprender el estado del arte en la simulación de contacto ferroviario.

2.2.6.3. Método CONTACT

CONTACT es un método no Herciano, exacto, cuasiestático, no estacionario y tridimensional, diseñado para abordar con alta precisión tanto el problema de contacto normal como el tangencial. Este enfoque resulta especialmente útil para la simulación y análisis detallado de la dinámica del contacto rueda-carril, lo que justifica su elección en este estudio.

El método CONTACT se basa en una discretización espacial del área de contacto, similar a la empleada en Elementos Finitos. Específicamente, CONTACT es una variación del Método de Elementos de Contorno.

2.2.6.3.1. *Discretización Espacial*

Establecido el sistema de referencia $\boldsymbol{X_1X_2X_3}$ y el plano de contacto $x_3 = 0$, el *área potencial de contacto* se define como una región rectangular contenida en el plano de contacto que incluye, al menos, todos los puntos del área de contacto (véase la Figura 14). El área potencial se discretiza en una malla regular con elementos rectangulares de dimensiones $2a \times 2b$.

La hipótesis central del método CONTACT consiste en suponer que la distribución de tensiones en cada elemento de la malla es constante, tal como se muestra en la Figura 15. De este modo, la tensión p_j^I se define como la tensión en el I-ésimo elemento de la malla en la dirección j. Asimismo, se definen, respectivamente, u_j^I, h^I y ω^I como los desplazamientos, distancias indeformadas y velocidad asociada a la configuración indeformada, calculados en el centro del elemento I-ésimo elemento de la malla. Además, x_1^I y x_2^I denotan las coordenadas del I-ésimo elemento.

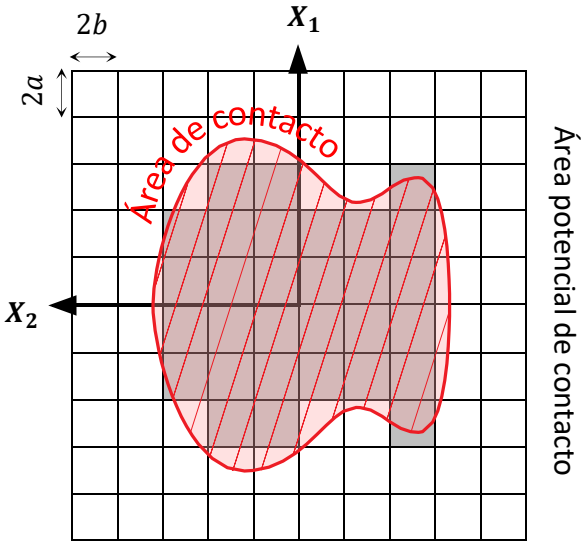

Figura 14. Discretización espacial de CONTACT. En rojo aparece el área real de contacto y en gris la estimada por CONTACT (Baeza, 2022).

2.2.6.3.2. *Hipótesis de Cuasiidentidad Elástica*

Asumiendo que se cumplen las condiciones de cuasiidentidad elástica, es posible desacoplar el problema normal del tangencial. Así, el método CONTACT se descompone en dos procedimientos: NORM y TANG, que calculan respectivamente los problemas normal y tangencial.

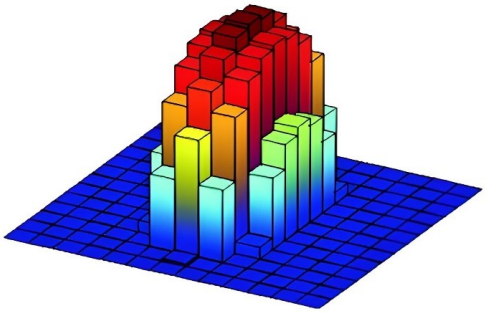

Figura 15. Ejemplo de la distribución de tensiones normales calculadas con CONTACT (Baeza, 2022).

2.2.6.3.3. NORM

El método CONTACT para el problema de contacto normal, también conocido como NORM, asume que el rectángulo J-ésimo en el área potencial de contacto está sometido a una presión normal, p_3^J. De acuerdo con la Teoría de la Elasticidad Lineal, el desplazamiento en el I-ésimo elemento de la malla se puede expresar mediante:

$$u_3^I = D_{I3}^{J3} p_3^J \,, \tag{21}$$

donde D_{I3}^{J3} es el coeficiente de influencia elástica. Su valor se determina mediante la ecuación (4), sustituyendo x_1 por $x_1^I - x_1^J$, y x_2 por $x_2^I - x_2^J$, y tomando $p_3 = 1$.

Aplicando el principio de superposición, la ecuación constitutiva que permite calcular los desplazamientos debidos a todas las cargas en el contacto resulta:

$$u_3^I = \sum_J D_{I3}^{J3} p_3^J \,, \tag{22}$$

extendiéndose el sumatorio a todos los elementos del área de contacto.

Al substituir la expresión anterior en la ecuación de la cinemática (11), se obtiene:

$$\delta = h^I - 2 \sum_J D_{I3}^{J3} p_3^J \,. \tag{23}$$

Al desarrollar dicha ecuación para todo elemento I del área de contacto, se obtiene un sistema lineal de ecuaciones donde las tensiones normales son las incógnitas. El desafío consiste en identificar qué elementos del área potencial de contacto forman parte realmente del área de contacto (es decir, determinar cuáles son los elementos sombreados en la Figura 14).

Para resolver el problema normal, se siguen los siguientes pasos:

1. **Suposición inicial**: Se supone que todos los elementos del área potencial de contacto pertenecen al área de contacto real.
2. **Se resuelve el sistema de ecuaciones (23).**
3. **Análisis de resultados:** Si existen elementos con tensiones positivas (elementos pegados), se concluye que estos elementos no forman parte del área de contacto real y, por lo tanto, se eliminan del área de contacto.
4. **Iteración:** Si se encuentran elementos con tensiones positivas en el paso anterior, se repite el paso 2 con la nueva área de contacto ajustada.
5. **Convergencia:** El proceso iterativo continúa hasta que no se encuentren elementos con tensiones positivas, momento en el cual se considera que se ha alcanzado la solución al problema normal de contacto.

2.2.6.3.4. TANG

El método CONTACT para el problema tangencial, también denominado TANG, requiere igualmente una ecuación constitutiva. Para obtenerla, es necesario calcular los coeficientes $D_{I\tau}^{J\varsigma}$, que describen el desplazamiento del elemento I en la dirección tangencial τ cuando en el elemento J se aplica una tensión tangencial en la dirección ς.

El cálculo de estos coeficientes es análogo al de los coeficientes de influencia elástica D_{I3}^{J3}, pero empleando la ecuación (5). La expresión resultante es:

$$u_\tau^I = \sum_J \left(D_{I\tau}^{J1} p_1^J + D_{I\tau}^{J2} p_2^J \right). \tag{24}$$

Al aplicar esta formulación a la cinemática tangencial (15) y discretizarla según el método CONTACT, se obtiene:

$$s_\tau^I = w_\tau^I - 2\dot{u}_\tau^I. \tag{25}$$

Si se calcula mediante diferencias finitas la derivada de los desplazamientos debidos a las deformaciones, se obtiene:

$$s_\tau^I = w_\tau^I - 2\frac{u_\tau^I - u_\tau^{I_0}}{\Delta t}, \tag{26}$$

donde Δt representa un pequeño incremento de tiempo, y $u_\tau^{I_0}$ los desplazamientos en el elemento I de la malla. Estos desplazamientos están asociados a la distribución de tensiones calculada en un instante previo $t - \Delta t$, aplicada sobre una malla desplazada con respecto a la actual una distancia en X_1 equivalente a $V\Delta t$. Cabe destacar que el método TANG es dependiente del tiempo y, en esencia, actúa como un algoritmo de integración para un sistema de ecuaciones diferenciales ordinarias.

Al sustituir las ecuaciones (24) y (20) en la ecuación de la cinemática tangencial (26), se obtiene:

$$\begin{aligned} s_1^I \Delta t &= w_1^I \Delta t - 2\sum_J \left(D_{I1}^{J1} p_1^J + D_{I1}^{J2} p_2^J \right) + 2u_1^{I_0} \\ s_2^I \Delta t &= w_2^I \Delta t - 2\sum_J \left(D_{I2}^{J1} p_1^J + D_{I2}^{J2} p_2^J \right) + 2u_2^{I_0}. \end{aligned} \tag{27}$$

Para cada elemento I de la malla en contacto (donde $p_3^I < 0$), se formularán las dos ecuaciones anteriores. El elemento I estará en estado de deslizamiento si la velocidad de deslizamiento local no es nula ($\|s^I\| > 0$), lo cual ocurre cuando las tensiones tangenciales alcanzan el límite de fricción:

$$p_3^I \mu = \sqrt{p_1^I + p_2^I}, \tag{28}$$

donde μ es el coeficiente de fricción entra la rueda y el carril.

Si el elemento I se encuentra en condiciones de deslizamiento, s_1^I y s_2^I se consideran incógnitas y sus tensiones tangenciales se calculan en función de la velocidad de deslizamiento local como sigue[1]:

$$p_1^I = -p_3^I \mu \frac{s_1^I}{\sqrt{s_1^I + s_2^I}}, \tag{29}$$

$$p_2^I = -p_3^I \mu \frac{s_2^I}{\sqrt{s_1^I + s_2^I}}. \tag{30}$$

En los elementos donde existe adhesión, la velocidad de deslizamiento local es cero ($\|s^I\| = 0$) y las incógnitas serán las tensiones tangenciales p_1^I y p_2^I.

El desafío principal es ahora determinar en qué elementos del área de contacto existe deslizamiento y en cuáles se cumplen las condiciones de adhesión. Para resolver este problema, el método TANG también sigue un proceso iterativo:

1. **Hipótesis inicial de adhesión:** Se asume que todos los elementos del área de contacto están en adhesión. Se definen como nulas todas las velocidades de deslizamiento local s_τ^I , y se definen las tensiones tangenciales p_τ^I como incógnitas.
2. **Se resuelve el sistema de ecuaciones asociado (27).**
3. **Análisis de resultados:** Los elementos del área de adhesión en los que la tensión tangencial calculada exceda el límite de fricción ($p_3^I \mu < \sqrt{p_1^I + p_2^I}$) se considerarán elementos en deslizamiento.
4. **Iteración:** Si en el paso anterior se identifican elementos con tensiones tangenciales superiores al límite de fricción, se repite el ciclo empezando por el paso 2. En los elementos en adhesión, las incógnitas serán las tensiones tangenciales p_1^I y p_2^I (y $s_1^I = s_2^I = 0$). En los elementos que experimenten deslizamiento, las incógnitas serán las velocidades de deslizamiento local s_1^I y s_2^I, y las tensiones tangenciales se calcularán en función de la velocidad de deslizamiento local, mediante las ecuaciones (29) y (30).
5. **Convergencia:** El proceso iterativo continúa hasta que no se identifiquen elementos donde la tensión tangencial calculada supere el límite de fricción, lo que indicará que se ha encontrado la solución al problema tangencial.

2.2.6.4. Modelo normal de Hertz

El método de contacto normal desarrollado por Heinrich Hertz aborda el problema del contacto elástico entre dos cuerpos. Hertz se centró en el análisis de la deformación de dos cuerpos en contacto y su influencia en la interferencia óptica. El modelo considera el contacto normal sin fricción entre dos cuerpos que se comportan, desde el punto de vista elástico, como semiespacios infinitos y cuyas superficies son cuádricas.

[1] Nótese que esta ecuación establece que, en condiciones de deslizamiento, la tensión tangencial (cuyo módulo es μp_3^I) tiene la misma dirección, pero sentido opuesto, a la velocidad de deslizamiento local.

En este modelo, el área de contacto resultante es elíptica y la distribución de las tensiones normales sigue una forma parabólica o semi-elipsoidal (como muestra la Figura 16). El algoritmo de cálculo de Hertz es preciso siempre que se cumpla la condición geométrica de que las superficies sean cuádricas. La formulación del método es analítica, excepto en la resolución de ciertas integrales elípticas que se encuentran tabuladas. Este método es un ejemplo de técnica semianalítica, pues se sirve de una combinación de métodos analíticos y numéricos para llegar a la solución.

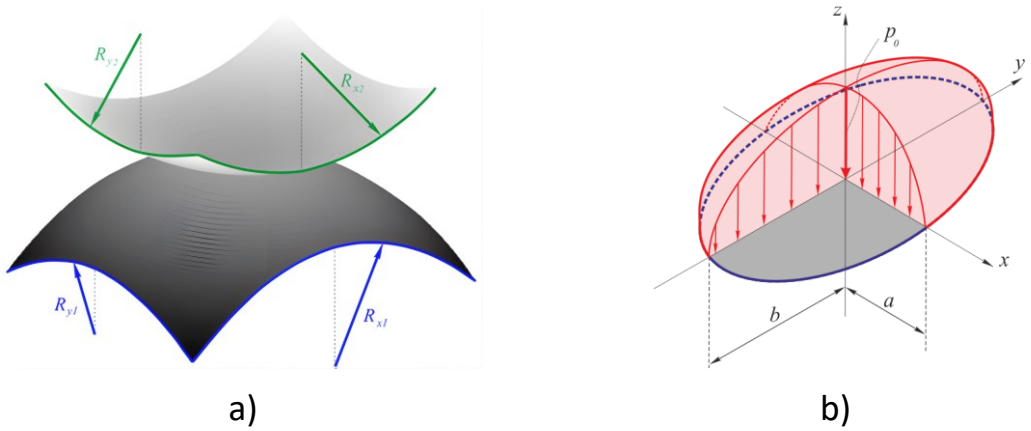

Figura 16. Teoría de contacto según Hertz (Sichani, 2016).

A continuación, se exponen las suposiciones que hacen aplicable el modelo normal de Hertz y los parámetros que este utiliza para modelar el problema normal.

2.2.6.4.1. Suposiciones del modelo de Hertz

Las condiciones esenciales para aplicar el modelo de contacto de Hertz son:

- **Elasticidad lineal, homogeneidad e isotropía del material:** Los cuerpos en contacto deben ser homogéneos, tener propiedades isotrópicas, y exhibir un comportamiento elástico lineal bajo la aplicación de cargas. Además, se asume que las deformaciones son infinitesimales.
- **Los cuerpos se tratan como semiespacios infinitos en las cercanías del contacto:** Esto implica que las dimensiones del área de contacto deben ser mucho menores que las dimensiones de los cuerpos, y también menores que los radios relativos de las superficies en contacto. En este contexto, el contacto se considera no conforme, y se supone que el área de contacto es plana.
- **Superficies cuádricas:** En la zona próxima al contacto, las superficies de los cuerpos se consideran aproximables a cuádricas, lo que indica que las curvaturas de las superficies son constantes.
- **Contacto sin fricción:** Se presupone que las superficies de los cuerpos son perfectamente lisas y no existe fricción entre ellas.

2.2.6.4.2. Parámetros y variables del problema

La aplicación de la teoría de Hertz a la resolución del problema de contacto parte de la definición de los siguientes parámetros:

- R_1: radio de la superficie de la rueda medido en el plano $\boldsymbol{X_1X_3}$. Se considera $R_1 > 0$ para superficies convexas. Habitualmente, R_1 se refiere al radio de la rueda, cumpliendo $R_1 > 0$.
- R_2: radio de la superficie del carril en el plano $\boldsymbol{X_1X_3}$. Se supone normalmente infinito, salvo en bancos de rodillos.
- R_1': radio de la superficie de la rueda medido en el plano $\boldsymbol{X_2X_3}$. Se considera $R_1' > 0$ para superficies convexas. Generalmente, R_1' corresponde al radio de curvatura del perfil de la rueda y es negativo para un perfil desgastado.
- R_2': radio de la superficie del carril según el plano $\boldsymbol{X_2X_3}$. Se considera $R_2' > 0$ para superficies convexas. Generalmente, R_1' corresponde al radio de curvatura del perfil del carril y $R_2' > 0$.
- ν_r, ν_c: coeficientes de Poisson de la rueda y el carril, respectivamente.
- E_r, E_c: módulo de Young de la rueda y el carril, respectivamente.
- F: fuerza normal de contacto aplicada.

Una vez definidos estos parámetros, se calcula el módulo de elasticidad reducido y los parámetros de curvatura (Sichani, 2016):

- Módulo de elasticidad reducido:

$$\frac{1}{E^*} = \frac{1-\nu_r^2}{E_r} + \frac{1-\nu_c^2}{E_c}, \tag{31}$$

- Parámetros de curvatura:

$$K_1 = \frac{1}{2}\left(\frac{1}{R_1} + \frac{1}{R_2}\right), \tag{32}$$

$$K_2 = \frac{1}{2}\left(\frac{1}{R_1'} + \frac{1}{R_2'}\right), \tag{33}$$

Los parámetros K_1 y K_2 encapsulan la información sobre las curvaturas en las direcciones $\boldsymbol{X_1}$ y $\boldsymbol{X_2}$, respectivamente.

Empleando las fórmulas anteriores, Hertz propone el cálculo de los semiejes de la elipse de contacto y la interpenetración como:

$$a = m \cdot \sqrt[3]{\frac{3F}{4E^*(K_1+K_2)}}, \tag{34}$$

$$b = n \cdot \sqrt[3]{\frac{3F}{4E^*(K_1+K_2)}}, \tag{35}$$

$$\delta = r \cdot \sqrt[3]{\left(\frac{3F}{4E^*}\right)^2 (K_1 + K_2)}\,, \tag{36}$$

donde a y b son los semiejes de la elipse de contacto en las direcciones longitudinal y lateral, respectivamente; δ es la interpenetración; y m, n y r son coeficientes geométricos que dependen de la relación entre K_1 y K_2, generalmente obtenidos de tablas preestablecidas. Estos coeficientes, se determinan a partir de tablas precalculadas (como las Tabla 1 y Tabla 2) haciendo uso del parámetro θ, que se define como:

$$\cos\theta = \frac{|K_1 - K_2|}{K_1 + K_2}, \tag{37}$$

Tabla 1. Coeficientes de Hertz ($K_1/K_2 < 1$) *(Iwnicki, 2006).*

$\theta°$	90	80	70	60	50	40	30	20	10	0
$g = n/m$	1	0,7916	0,6225	0,4828	0,3652	0,2656	0,1806	0,108	0,047	0
m	1	1,128	1,285	1,486	1,754	2,136	2,731	3,816	6,612	∞
n	1	0,8927	0,8	0,7171	0,6407	0,5673	0,4931	0,4122	0,311	0
r	1	0,9932	0,9726	0,9376	0,8867	0,8177	0,7263	0,6038	0,428	0

Tabla 2. Coeficientes de Hertz para $\theta = 0°$ *hasta* $180°$ *(Iwnicki, 2006).*

θ°	0	5	10	30	60	90	120	150	170	175	180
K_1/K_2	0	0,0019	0,0077	0,0717	0,3333	1	3	13,93	130,6	524,6	∞
$b/a = n/m$	0	0,0212	0,047	0,1806	0,4826	1	2,072	5,538	21,26	47,2	∞
m	∞	11,238	6,612	2,731	1,486	1	0,7171	0,4931	0,311	0,2381	0
r	0	0,2969	0,428	0,7263	0,9376	1	0,9376	0,7263	0,428	0,2969	0

Tras la definición de los semiejes, y teniendo en cuenta la forma de la distribución de presiones ilustrada, se puede formular la presión en cualquier punto del área elíptica de contacto como:

$$p(x_1, x_2) = p_0 \sqrt{1 - \left(\frac{x_1}{a}\right)^2 - \left(\frac{x_2}{b}\right)^2}\,, \tag{38}$$

donde p_0 es la presión máxima en el centro del área de contacto según la teoría de Hertz, y está relacionada con la fuerza F preestablecida por la siguiente expresión:

$$p_0 = \frac{3F}{2\pi ab}. \tag{39}$$

Como alternativa, si se conoce de antemano δ en lugar de la fuerza F, las ecuaciones (34) y (35) se escribirían como:

$$a = m \cdot \sqrt{\frac{\delta}{r(K_1 + K_2)}}, \tag{40}$$

$$b = n \cdot \sqrt{\frac{\delta}{r(K_1 + K_2)}}, \tag{41}$$

$$F = \frac{4E^*}{2}\sqrt{\frac{1}{A+B}\left(\frac{\delta}{r}\right)^3}. \tag{42}$$

2.2.6.4.3. El modelo de Hertz en aplicaciones ferroviarias

Aunque el modelo de contacto de Hertz proporciona una solución precisa y semianalítica para el contacto elástico entre cuerpos con superficies cuádricas, es importante considerar sus limitaciones y su aplicabilidad en diferentes contextos. Este modelo es especialmente útil en aplicaciones como el contacto rueda-carril en la ingeniería ferroviaria. Las suposiciones de Hertz, como la forma elíptica del área de contacto y la distribución parabólica de las tensiones normales, se ajustan bien a las condiciones que se dan en el contacto normal. Sin embargo, suposiciones como la elasticidad lineal y el tratamiento de los cuerpos como semiespacios infinitos, pueden no cumplirse en ciertos escenarios modernos.

A pesar de su precisión y utilidad, el modelo de Hertz se basa en una serie de supuestos que no siempre se cumplen en aplicaciones reales. Por ejemplo, Hertz asume que los materiales en contacto presentan un comportamiento elástico lineal, una condición que puede no satisfacerse en aplicaciones ferroviarias modernas, donde las cargas por eje son considerablemente altas. Aunque la presión de contacto puede superar el límite elástico del material, no siempre induce flujo plástico superficial. De hecho, el flujo plástico suele darse por debajo de la superficie y la alcanza al aumentar la carga, especialmente en ausencia de tensiones tangenciales. Estudios en el contacto rueda-carril han demostrado que este comportamiento puede reducir las presiones de contacto y expandir el área de contacto.

Otra suposición del modelo de Hertz es la condición de semiespacio, que implica que las dimensiones del área de contacto son mucho menores que las dimensiones de los cuerpos en contacto. Si bien esta suposición es válida en el contacto entre la banda de rodadura de la rueda y el carril, su validez es cuestionable en el caso del contacto entre la pestaña y el carril, particularmente en curvas cerradas. Además, el modelo de Hertz no toma en cuenta los efectos de fricción, lo que limita su aplicabilidad en situaciones donde la fricción es un factor determinante.

A pesar de estas limitaciones, el modelo de Hertz sigue siendo una herramienta fundamental para el análisis preliminar de problemas de contacto elástico en situaciones que cumplen con sus supuestos básicos. En escenarios más complejos, donde las superficies son rugosas, las deformaciones plásticas son significativas o la fricción juega un papel importante, es recomendable utilizar métodos numéricos más avanzados que permitan una representación más precisa y realista de las condiciones de contacto.

2.2.6.5. Teoría lineal de Kalker

En 1967 Joost Jacques Kalker formuló una teoría en la que postulaba que la relación entre las fuerzas de contacto tangenciales y los pseudodeslizamiento era de carácter lineal. Según Kalker, cuando una partícula circula en una dirección paralela a la rodadura y entra en la región de contacto, se genera una tracción debido al reducido deslizamiento, la cual se distribuye uniformemente en la parte delantera del contacto. A medida que la partícula abandona esta superficie, la tracción disminuye gradualmente hasta desaparecer por completo, como se ilustra en la Figura 17.

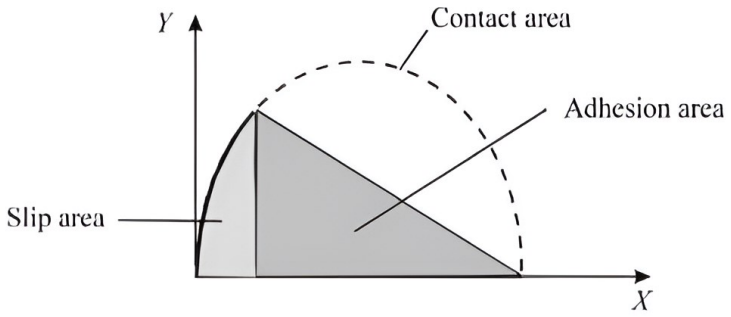

Figura 17. Contacto según la Teoría Lineal de Kalker (San-Segundo Cordero, 2019).

Para describir matemáticamente este comportamiento, Kalker desarrolló un conjunto de expresiones lineales que permiten cuantificar las fuerzas de contacto en función del pseudodeslizamiento. Las expresiones que describen estas fuerzas tangenciales son las siguientes:

$$F_x = -f_{33} \cdot \xi_x \,, \tag{43}$$

$$F_y = -f_{11} \cdot \xi_y - f_{12} \cdot \psi \,, \tag{44}$$

$$M_z = f_{12} \cdot \xi_y - f_{22} \cdot \psi \,, \tag{45}$$

donde:

- F_x es la fuerza debida al pseudodeslizamiento longitudinal.
- F_y es la fuerza debida al pseudodeslizamiento lateral.
- M_z es el momento debido al pseudodeslizamiento de giro.
- ξ_x es el pseudodeslizamiento longitudinal.
- ξ_y es el pseudodeslizamiento lateral.
- ψ es el pseudodeslizamiento de giro.

- f_{11}, f_{12}, f_{22} y f_{33} son los coeficientes de pseudodeslizamiento.

  Kalker define los coeficientes de pseudodeslizamiento como:

$$f_{11} = (a \cdot b) \cdot G \cdot C_{22}\,, \tag{46}$$

$$f_{12} = (a \cdot b)^{3/2} \cdot G \cdot C_{23}\,, \tag{47}$$

$$f_{22} = (a \cdot b)^{2} \cdot G \cdot C_{33}\,, \tag{48}$$

$$f_{33} = (a \cdot b) \cdot G \cdot C_{11}\,, \tag{49}$$

  tal que:

  - a y b son los semiejes del área de contacto según las direcciones de rodadura y lateral, respectivamente.
  - G es el módulo de cizalladura combinado para los materiales de la rueda y el carril, dado por:

$$G = \frac{2G_w G_r}{G_w + G_r}\,. \tag{50}$$

C_{11}, C_{22}, C_{23} y C_{33} son los coeficientes de deslizamiento y de giro cuyos valores están tabulados (La teoría lineal de Kalker es aplicable únicamente cuando los pseudodeslizamientos longitudinal, lateral y de giro son mínimos. En estas condiciones, la zona de deslizamiento es reducida, lo que permite asumir que el área de adherencia abarca por completo la región de contacto. No obstante, para escenarios con mayores pseudodeslizamientos, Kalker desarrolló la *Teoría Exacta*, que implementó en el software CONTACT. Esta teoría se caracteriza por su alta precisión, aunque el coste computacional es elevado. Con el fin de reducir dicho coste, formuló la *Teoría Simplificada*, implementada en FASTSIM, la cual es más eficiente, aunque con un margen de error del 10-15% en comparación con CONTACT.

  - Tabla 3) y dependen únicamente del módulo de rigidez (G) y del coeficiente de Poisson (ν) combinados para los materiales de la rueda y el carril. El coeficiente de Poisson combinado se calcula como:

$$\nu = \frac{G(G_w \nu_r + G_r \nu_w)}{2G_w G_r}\,, \tag{51}$$

    donde G_w y G_r son los módulos de cizalladura de la rueda y el carril, respectivamente. Asimismo, ν_w y ν_r se refieren a los coeficientes de Poisson de la rueda y el carril, respectivamente.

La teoría lineal de Kalker es aplicable únicamente cuando los pseudodeslizamientos longitudinal, lateral y de giro son mínimos. En estas condiciones, la zona de deslizamiento es reducida, lo que permite asumir que el área de adherencia abarca por completo la región de contacto. No

obstante, para escenarios con mayores pseudodeslizamientos, Kalker desarrolló la *Teoría Exacta*, que implementó en el software CONTACT. Esta teoría se caracteriza por su alta precisión, aunque el coste computacional es elevado. Con el fin de reducir dicho coste, formuló la *Teoría Simplificada*, implementada en FASTSIM, la cual es más eficiente, aunque con un margen de error del 10-15% en comparación con CONTACT.

Tabla 3. Valores de los coeficientes de deslizamiento y giro según Kalker (San-Segundo Cordero, 2019).

	C_{11}			C_{22}			$C_{23} = -C_{32}$			C_{33}		
g	$\sigma = 0$	$1/4$	$1/2$	$\sigma = 0$	$1/4$	$1/2$	$\sigma = 0$	$1/4$	$1/2$	$\sigma = 0$	$1/4$	$1/2$
0.0	$\pi^2/4(1-\sigma)$			$\pi^2/4 = 2,47$			$\pi\sqrt{g}/3$	-	-	$\pi^2/16(1-\sigma)g$		
						a/b						
0,1	2,51	3,31	4,85	2,51	2,52	2,53	0,33	0,47	0,73	6,42	8,28	11,70
0,2	2,59	3,37	4,81	2,59	2,63	2,66	0,48	0,60	0,81	3,46	4,27	5,66
0,3	2,68	3,44	4,80	2,68	2,75	2,81	0,61	0,72	0,89	2,49	2,96	3,72
0,4	2,78	3,53	4,82	2,78	2,88	2,98	0,72	0,82	0,98	2,02	2,32	2,77
0,5	2,88	3,62	4,83	2,88	3,01	3,14	0,83	0,93	1,07	1,74	1,93	2,22
0,6	2,98	3,72	4,91	2,98	3,14	3,31	0,93	1,03	1,18	1,56	1,68	1,86
0,7	3,09	3,81	4,97	3,09	3,28	3,48	1,03	1,14	1,29	1,43	1,50	1,60
0,8	3,19	3,91	5,05	3,19	3,41	3,65	1,13	1,25	1,40	1,34	1,37	1,42
0,9	3,29	4,01	5,12	3,29	3,54	3,82	1,21	1,36	1,51	1,27	1,27	1,27
						b/a						
1	3,40	4,12	5,20	3,40	3,67	3,98	1,33	1,47	1,63	1,21	1,19	1,16
0,9	3,51	4,22	5,30	3,51	3,81	4,16	1,44	1,59	1,77	1,16	1,11	1,04
0,8	3,65	4,36	5,42	3,65	3,99	4,39	1,58	1,75	1,94	1,10	1,04	0,95
0,7	3,82	4,54	5,58	3,82	4,21	4,67	1,76	1,95	2,18	1,05	0,97	0,85
0,6	4,06	4,78	5,80	4,06	4,50	5,04	2,01	2,23	2,50	1,01	0,89	0,75
0,5	4,37	5,10	6,11	4,37	4,90	5,56	2,35	2,62	2,96	0,96	0,82	0,65
0,4	4,84	5,57	5,57	4,84	5,48	6,31	2,88	3,24	3,70	0,91	0,75	0,55
0,3	5,57	6,34	7,34	5,57	6,40	7,51	3,79	4,32	5,01	0,87	0,67	0,45
0,2	6,96	7,78	8,82	6,96	8,14	9,79	5,72	6,63	7,89	0,83	0,60	0,34
0,1	10,70	11,70	12,90	10,70	12,80	16,00	12,20	14,60	18,00	0,80	0,53	0,23

2.2.6.6. Algoritmo FASTSIM

La teoría simplificada de Kalker establece una relación lineal entre la deformación superficial y la tracción, de modo que:

$$u = Lp, \tag{52}$$

donde $\boldsymbol{u} = (u_1, u_2, u_3)$ es el vector de deformación, $\boldsymbol{p} = (p_1, p_2, p_3)$ es el vector de tracciones y L la matriz de coeficientes de flexibilidad.

El uso de la teoría simplificada en la dirección normal, $u_z = Lp$, para contactos hertzianos carece de sentido, pues proporciona soluciones poco precisas. La solución analítica precisa la proporciona el propio modelo normal de Hertz. Sin embargo, aplicar la teoría simplificada al problema tangencial permite simplificar y acelerar la estimación de la distribución de esfuerzos tangenciales. Así es como surgió el algoritmo FASTSIM.

FASTSIM (Kalker, 2000) es un algoritmo diseñado para calcular la tracción tangencial en el contacto, particularmente cuando φ (*spin creepage*, o pequeño deslizamiento de la rueda debido al spin) no es despreciable. FASTSIM está específicamente ideado para ser un método rápido de simulación, proporcionando soluciones aproximadas pero eficientes, lo que lo hace ideal para aplicaciones donde el tiempo de cálculo es esencial.

En términos generales, el algoritmo se basa en descomponer la zona de contacto entre la rueda y el carril en una malla rectangular o cuadrada y calcular las fuerzas tangenciales en cada punto de esta malla utilizando relaciones simplificadas basadas en el deslizamiento y la deformación elástica. De forma más concreta, técnica y de acuerdo con lo anterior, la formulación del método parte de una expresión similar a la ecuación (13), definiendo el deslizamiento como:

$$\boldsymbol{s} = \boldsymbol{c} + \frac{\boldsymbol{u} - \boldsymbol{u}'}{k} = \boldsymbol{c} + \frac{L(\boldsymbol{p} - \boldsymbol{p}')}{k}\,, \tag{53}$$

donde $\boldsymbol{c} = \dot{\boldsymbol{x}}^r - \dot{\boldsymbol{x}}^c$, conocido como *creep* o deslizamiento rigido, es la diferencia de las velocidades entre dos partículas de las superficies en contacto, según su configuración indeformada. $\boldsymbol{p} = (p_1[(x_1, x_2), t], p_2[(x_1, x_2), t])^T$ el esfuerzo tangencial actual y $\boldsymbol{p}' = \boldsymbol{p}[(\boldsymbol{x} + \boldsymbol{v}k), t - k]$ la tracción tangencial en el instante previo. Además, en este contexto, k es el intervalo de tiempo considerado.

Si se sigue a una partícula a lo largo de su trayectoria, es posible asumir que en el borde delantero $a(x_2, t)$ se da $\boldsymbol{p} = 0$. Si se trabaja ahora por inducción, suponiendo conocido $\boldsymbol{p}'$, se tiene que:

$$\boldsymbol{p} = \boldsymbol{p}' + (k/L)[s - c]\,, \tag{54}$$

donde $\boldsymbol{s}$ se configura como la incógnita a determinar.

Dado que $\boldsymbol{c}$ es conocido, se asume tentativamente $\boldsymbol{s} = 0$. Con esta suposición, se puede obtener:

$$\boldsymbol{p}_H \approx \boldsymbol{p}(x)_{s=0} = \boldsymbol{p}' - \left(\frac{k}{L}\right)\boldsymbol{c}\,. \tag{55}$$

Esto permite determinar el módulo $|\boldsymbol{p}_H|$ y compararlo con el parámetro g, definido como el límite de tracción tangencial para un punto $(x_1, x_2) \in C$.

- Si $\boldsymbol{p}_H \leq g$, se establece $\boldsymbol{p} = \boldsymbol{p}_H$. Esta situación implica que $\boldsymbol{s} = 0$ y, según la ley de Coulomb, se tiene adhesión en el contacto o, en caso de que se tenga la igualdad, se cumple la ley en su estado límite.
- Si $\boldsymbol{p}_H > g$, se establece $\boldsymbol{p} = \left(\frac{g}{|\boldsymbol{p}_H|}\right)\boldsymbol{p}_H$. Entonces, $|\boldsymbol{p}| = g$ y se define el deslizamiento como:

$$\boldsymbol{s} = \boldsymbol{c} + \left(\frac{L}{k}\right)(\boldsymbol{p} - \boldsymbol{p}') = \left(\frac{L}{k}\right)(-\boldsymbol{p}_H + \boldsymbol{p}) = -\left(\frac{L}{k}\right)\boldsymbol{p}_H\left(1 - \left[\frac{g}{|\boldsymbol{p}_H|}\right]\right), \tag{56}$$

de tal forma que $\boldsymbol{s}$ está en dirección opuesta a $\boldsymbol{p}$ según la ley de Coulomb.

Partiendo de la formulación anterior y considerando condiciones de estacionariedad, de tal forma que $p(x,t) = p(x)$, pueden definirse, los siguientes parámetros característicos:

- $\boldsymbol{x} = (x_1, x_2)$ coordenadas de los elementos del área de contacto.
- $\boldsymbol{x} + k\boldsymbol{v} = (x_1 + q, x_2)$ coordenadas de los elementos del área de contacto en un instante posterior.
- m, el número de elementos en el área según la dirección X_1.
- n, el número de elementos en el área según la dirección X_2.
- $V = |\boldsymbol{v}|$, la velocidad de movimiento en la dirección de rodadura, considerada siempre positiva.
- L, el coeficiente de flexibilidad. Es un valor positivo.
- $g = g(x_1, x_2)$, el límite de tracción tangencial admisible por un punto $(x_1, x_2) \in C$.
- $\boldsymbol{c}(x_1, x_2)$, el deslizamiento rígido en un punto $(x_1, x_2) \in C$.
- $a(x_2)$, el borde delantero.
- $-a(x_2)$, el borde trasero.
- a y b, los semiejes de la elipse de contacto.
- $\boldsymbol{p_H}$ y $\boldsymbol{p}$, tracciones tangenciales, incógnita del problema.
- $\boldsymbol{F}$, la fuerza tangencial total, incógnita del problema.

Una vez definidos los parámetros propios del problema, el algoritmo FASTSIM seguiría la siguiente secuencia:

1. **Inicio del programa y del bucle de $\boldsymbol{x_2}$:** Se establece $r = \frac{2b}{n}; x_2 = b - \frac{r}{2}; \boldsymbol{F} = 0$.
2. **Inicio del bucle de $\boldsymbol{x_1}$:** $q = \frac{2a(x_2)}{m}; x = a(x_2) - q; p = p(x_1 + q, x_2) = 0$.
3. **Actualización de $\boldsymbol{p}'$:** Dado que se ha encontrado $\boldsymbol{p}$, se actualiza $\boldsymbol{p}' = \boldsymbol{p}$
4. **Formación de $\boldsymbol{p_H}$:** Siendo $\left(x_1 + \frac{q}{2}\right)$ el punto medio entre x_1 y $x_1 + q$, se define:

$$\boldsymbol{p} = \boldsymbol{p}' - \left[\frac{q}{VL}\right]\boldsymbol{c}\left(x_1 + \frac{q}{2}, x_2\right). \tag{57}$$

5. **Evaluación de la tensión tangencial obtenida:**

- Si $|\boldsymbol{p}| > g$ entonces se define $\boldsymbol{p} = \left(\frac{g}{|\boldsymbol{p}|}\right)\boldsymbol{p}$.
- Si $|\boldsymbol{p}| \leq g$, se tienen condiciones de adhesión y $\boldsymbol{p}(x_1, x_2) = \boldsymbol{p} = \boldsymbol{p_H}$.

6. **Rellenar $\boldsymbol{p}(\boldsymbol{x_1}, \boldsymbol{x_2})$ y actualizar $\boldsymbol{F}$:**

- $\boldsymbol{p}(x_1, x_2) = \boldsymbol{p}$.
- $\boldsymbol{F} = \boldsymbol{F} + q \cdot r \cdot \boldsymbol{p}$.

7. **Comprobación del bucle para $\boldsymbol{x_1}$:** se calcula $x_1 = x_1 - q$. si $x_1 > -a(x_2)$ se vuelve al paso 3.
8. **Comprobación del bucle para $\boldsymbol{x_2}$:** se calcula $x_2 = x_2 - r$. si $x_2 > -b$ se vuelve al paso 2.
9. **Finalización:** Si los anteriores pasos no generan otra iteración, termina el algoritmo.

Como puede observarse, el algoritmo FASTSIM es una herramienta poderosa para el análisis rápido de problemas de contacto rueda-carril, especialmente útil en aplicaciones donde se requiere una solución computacionalmente eficiente. Aunque es una aproximación, su velocidad y simplicidad lo hacen invaluable para situaciones donde el tiempo de cálculo es crítico.

2.3. Fundamentos teóricos de los algoritmos a implementar

2.3.1. Introducción

En esta sección se presentan los fundamentos teóricos de los algoritmos implementados en este trabajo. Es importante destacar que el objetivo principal de dichos algoritmos es resolver el problema de contacto normal, aunque las hipótesis y metodologías empleadas varían según el método aplicado. Posteriormente, en el apartado 3, se detalla cómo se ha adaptado esta teoría al contexto de la programación, junto como la metodología seguida para su implementación.

El trabajo de programación integra tres métodos para la resolución y cálculo del problema normal, cuya eficiencia computacional será evaluada posteriormente. Estos métodos se basan en las siguientes propuestas:

- El método NORM del software CONTACT en su forma pura.
- Una aproximación basada en distribuciones de tensiones polinómicas o elípticas, utilizando el método NORM como base, propuesta por Giner Navarro et al. (2023).
- El método del gradiente conjugado con restricciones acotadas. Es una combinación de una técnica de conjunto activo (*active set*[2]) con el método del gradiente conjugado optimizado mediante las fórmulas de Polak-Ribière. Esta propuesta, publicada por Vollebregt (2013), da lugar a un método conocido como *Bound-Constrained Conjugate Gradient,* o más específicamente, *Enhanced BCCG(K).* Por simplicidad, también se refiere a este como BCCG(K).

2.3.2. NORM de CONTACT

El procedimiento NORM de CONTACT permite resolver el problema normal de contacto entre la rueda y el carril de manera más precisa, considerando aspectos complejos que otros modelos simplificados, como el de Hertz, no pueden abordar adecuadamente.

[2] *Active Set Algorithm*: Método de optimización que se utiliza para resolver problemas con restricciones, especialmente en situaciones donde ciertas restricciones pueden volverse activas o inactivas durante el proceso de optimización. NORM es un algoritmo de esta tipología.

En particular, este método facilita la resolución de configuraciones geométricas complejas en la zona de contacto. Esto se logra mediante la discretización de la denominada *área potencial de contacto*, un área rectangular definida en las superficies de contacto, que es lo suficientemente grande como para contener todos los puntos de contacto reales. Esta área se subdivide en pequeñas subáreas, lo que permite un análisis detallado de la distribución de presión normal. Por esta razón, CONTACT se considera una variación del Método de Elementos de Contorno.

Recuérdese que el método NORM, según lo explicado en el apartado 2.2.6.3.3, conduce a la expresión:

$$\delta = h^I - 2\sum_J D_{I3}^{J3} p_3^J \,. \tag{58}$$

Partiendo de esta y expandiéndola para todo elemento I del área potencial de contacto, se obtiene el siguiente sistema lineal de ecuaciones en forma matricial:

$$\boldsymbol{\delta} = \boldsymbol{h} - 2\boldsymbol{D}\boldsymbol{p_d}, \tag{59}$$

donde $\boldsymbol{D}$ es la matriz de coeficientes de influencia D_{I3}^{J3} calculados según la ecuación (4), sustituyendo x_1 por $x_1(I) - x_1(1)$ y x_2 por $x_2(J) - x_2(1)$, y tomando $p_3 = 1$; $\boldsymbol{h}$ el vector que contiene la distancia entre las superficies indeformadas en contacto de la rueda y el carril; $\boldsymbol{p_d}$ el vector de tensiones normales, incógnitas del problema normal; y $\boldsymbol{\delta}$ el vector que contiene la interpenetración de la rueda en el carril o *approach.* La dimensión de las matrices y vectores está sujeta al número total de elementos empleados para definir el área potencial de contacto.

De acuerdo con las suposiciones anteriores, y teniendo en cuenta que la interpenetración es nula en el área de contacto, al agrupar términos se obtiene el sistema:

$$\boldsymbol{h} = \boldsymbol{A}\boldsymbol{p_d}, \tag{60}$$

donde $\boldsymbol{A} = 2\boldsymbol{D}$ es la matriz de coeficientes de influencia del problema, de dimensión $N_e \times N_e$, con $N_e = N_1 \cdot N_2$ (siendo N_1 y N_2 el número de elementos de contacto en la dirección de rodadura y en la dirección lateral, respectivamente). $\boldsymbol{A}$ es una matriz *Toeplitz*[3], cuyas entradas a_{ij} dependen de la posición relativa $\boldsymbol{x}^I - \boldsymbol{x}^J$ de los elementos en el área potencial de contacto[4].

Los parámetros que intervienen de forma habitual en el problema son:

- E, el módulo de Young de los materiales constitutivos de las dos superficies en contacto.
- ν, el coeficiente de Poisson de los materiales constitutivos.

[3] Matriz Toeplitz: Matriz en la que cada elemento a lo largo de una diagonal es constante, es decir, los elementos en cada diagonal descendente de izquierda a derecha son iguales.

[4] Nótese que $\boldsymbol{x}$ denota el conjunto de coordenadas (x_1, x_2) de un punto concreto.

- N_1, el número de elementos en la dirección de rodadura X_1.
- N_2, el número de elementos en la dirección lateral X_2.
- $\boldsymbol{h} = h(x_1, x_2)$, la distancia indeformada de cada punto del área potencial de contacto.
- x_1 y x_2, las coordenadas longitudinales y laterales, respectivamente, de los elementos del área potencial de contacto.
- a y b, los semianchos de los elementos en las direcciones X_1 y X_2, respectivamente.

Una vez conocidas las variables y el sistema de ecuaciones asociado al problema, la resolución mediante el algoritmo se realiza siguiendo la siguiente secuencia de pasos:

1. **Suposición inicial**: Se asume que todos los elementos del área potencial de contacto forman parte del área de contacto.
2. **Se resuelve el sistema de ecuaciones (60)**.
3. **Análisis de resultados:** Si se identifican elementos con tensiones nulas o negativas (elementos pegados), se concluye que estos elementos no forman parte del área de contacto real y se eliminan del área de contacto.
4. **Iteración:** Si se encuentran elementos con tensiones nulas o negativas en el paso anterior, se repite el paso 2 con la nueva área de contacto ajustada.
5. **Convergencia:** El proceso iterativo continúa hasta que no se detecten elementos con tensiones positivas, momento en el cual se considera que se ha encontrado la solución al problema normal de contacto.

Es importante destacar que se descartan los elementos con tensiones negativas debido a la configuración del sistema de ecuaciones. No obstante, la secuencia de pasos presentada sigue siendo equivalente a la presentada en el apartado 2.2.6.3.3.

Mediante el algoritmo NORM y los parámetros correspondientes, obtenidos empírica o analíticamente, es posible determinar:

- $\boldsymbol{P} = p_d(x_1, x_2)$, la matriz que contiene la distribución de tensiones normales en el contacto.
- N, la fuerza normal aplicada.
- x_1^p y x_2^p, las coordenadas del centro de presiones según las direcciones longitudinal y lateral, respectivamente.
- El área de contacto real en función de la forma de la distribución de tensiones.

2.3.3. Aproximación de la distribución de tensiones en el contacto mediante una base elíptica o polinómica.

El algoritmo propuesto por Giner Navarro et al. (2023) supone que la distribución de la tracción en la dirección $\boldsymbol{X_1}$ (en cada tira de la malla) sigue una forma aproximadamente elíptica. Esto permite formular la distribución en cada tira longitudinal a través de una base que dependa de un número reducido de parámetros. De esta manera, la tracción normal p_{es} en el elemento e-ésimo de la tira s puede aproximarse mediante una elipse de la siguiente forma:

$$p_{es} \approx \lambda_{0s} + \lambda_{1s}\sqrt{1-\left(\frac{x_{es}}{L_s}\right)^2}, \tag{61}$$

donde x_{es} es la coordenada en la dirección de rodadura del elemento e-ésimo en la tira s del área de contacto, L_s es la semilongitud de la tira en la dirección de rodadura X_1, y λ_{gs} son los coeficientes de orden g asociados a la tira s.

El método se sirve del mismo sistema de referencia $\boldsymbol{X_1X_2X_3}$ y parte del mismo sistema matricial (60). De manera similar, se asume que el área potencial de contacto se divide en una malla regular de rectángulos, en los cuales la tracción normal se considera constante en cada elemento. Asimismo, al emplear el método NORM, el problema de contacto normal se resuelve teniendo en cuenta las tracciones obtenidas en cada iteración y excluyendo los elementos que presenten tracciones nulas o negativas, ya que no se consideran parte del área de contacto real.

Para identificar los elementos que no deben formar parte del área de contacto, la ecuación (61) permite obtener una solución que genera un valor de tracción negativo en esos elementos, ubicados en el borde del área de contacto, utilizando el parámetro λ_{0s}. En la iteración final del algoritmo NORM, este valor constante tiende a ser prácticamente cero, lo que sirve como indicativo de que se ha alcanzado la solución correcta. Según este enfoque, el vector de tracción $\boldsymbol{p_d}$ se puede expresar como:

$$\boldsymbol{p_d} \approx \boldsymbol{Q\lambda}, \tag{62}$$

donde el vector $\boldsymbol{\lambda}$ contiene los coeficientes λ_{gs}, y $\boldsymbol{Q}$ es una matriz que depende de las coordenadas de la malla x_{es} y de las longitudes de las tiras L_s. Dado que el área de contacto se determina en cada cálculo, x_{es}, L_s , y con ello $\boldsymbol{Q}$, son conocidos. Es importante destacar que el $\boldsymbol{\lambda}$ es de dimensión $2N_s \times 1$ y $\boldsymbol{Q}$ de dimensión $N_e \times N_S$, donde N_e y N_S son el número total de elementos en la malla y el número de tiras que componen el área de contacto, respectivamente.

Sustituyendo (62) en (60), y premultiplicando el resultado por $\boldsymbol{Q}^T$, se obtiene:

$$\boldsymbol{Q}^T\boldsymbol{h} = \boldsymbol{Q}^T\boldsymbol{A}\boldsymbol{Q\lambda}, \tag{63}$$

que permite obtener, a partir de las ecuaciones (62) y (63), la siguiente expresión:

$$\boldsymbol{p_d} = \boldsymbol{Q}(\boldsymbol{Q}^T\boldsymbol{A}\boldsymbol{Q})^{-1}(\boldsymbol{Q}^T\boldsymbol{h}). \tag{64}$$

Con lo anterior, el problema reduce el número de incógnitas de N_e (en el problema original) a $2N_s$, lo que favorece la eficiencia computacional.

Aunque el uso de la aproximación elíptica permita reducir la complejidad del problema, este método presenta limitaciones al intentar representar con exactitud el área de contacto cuando el ángulo de guiñada del eje montado ψ no es nulo. En estas circunstancias, el plano $\boldsymbol{X_2X_3}$ deja de ser un plano de simetría para la distribución de la tracción normal. Debido a la simetría inherente de la

base elíptica (61), solo es posible generar distribuciones de tracción simétricas, lo cual no es suficiente para captar adecuadamente la asimetría causada por el ángulo de guiñada.

Para resolver este problema, Giner Navarro et al. (2023) propone utilizar una distribución de tensiones polinómicas en lugar de elípticas. En esta nueva formulación, la tracción normal p_{es} del elemento e-ésimo en la tira s de la malla se aproxima mediante un polinomio de grado g. Este polinomio se expresa como una suma de términos, donde los coeficientes ρ_{js} corresponden a los distintos órdenes j del polinomio:

$$p_{es} \approx \sum_{j=0}^{g} \rho_{js} x_{es}^{j} . \tag{65}$$

Aquí, x_{es} es nuevamente la coordenada en la dirección de rodadura para el elemento e-ésimo en la tira s. Con esta nueva aproximación, se sigue un procedimiento similar al aplicado con la base elíptica, lo que permite expresar el vector de tracción $\boldsymbol{p_d}$ de la siguiente forma:

$$\boldsymbol{p_d} \approx \boldsymbol{B\rho}, \tag{66}$$

donde $\boldsymbol{\rho}$ es el vector que contiene los coeficientes ρ_{js} del polinomio para cada tira de la malla y $\boldsymbol{B}$ una matriz que depende de las coordenadas de la malla x_{es} y del grado del polinomio g. Siguiendo un proceso análogo al descrito anteriormente para la base elíptica, se obtiene un sistema de ecuaciones de la forma:

$$\boldsymbol{B^T h} = \boldsymbol{B^T AB\rho}, \tag{67}$$

donde $\boldsymbol{B^T}$ es la transpuesta de la matriz de $\boldsymbol{B}$. Este sistema se resuelve para obtener los coeficientes polinómicos $\boldsymbol{\rho}$ que describen la distribución de la tracción normal. Aunque la introducción de una base polinómica aumenta la dimensión del problema a $(g+1)N_s$, esta sigue siendo considerablemente menor que la del problema original, lo que permite una resolución más eficiente.

Finalmente, la distribución de la tracción normal se calcula mediante la siguiente expresión:

$$\boldsymbol{p_d} = \boldsymbol{B}(\boldsymbol{B^T AB})^{-1}(\boldsymbol{B^T h}). \tag{68}$$

En resumen, ambas propuestas, la base elíptica y la polinómico, son eficaces para resolver el problema de contacto entre rueda y carril, ya que reducen significativamente el número de incógnitas mejorando la eficiencia computacional. El modelo elíptico es más simple y eficiente para situaciones simétricas, pero presenta limitaciones cuando existe ángulo de guiñada. Por otro lado, el modelo polinómico, aunque es más complejo, ofrece una mayor precisión en escenarios asimétricos.

Definidas las aproximaciones de las tensiones mediante las bases correspondientes, la resolución de los sistemas (63) y (67) permite obtener $\boldsymbol{\lambda}$ y $\boldsymbol{\rho}$, respectivamente. Conocidas estas

incógnitas, el vector $\boldsymbol{p_d}$ se determina utilizando las ecuaciones (62) o (66), según la base empleada. A partir de este punto, se descartan los elementos con tensiones negativas de acuerdo con el método NORM, y se definen las sucesivas áreas de contacto, repitiendo el procedimiento de aproximación hasta converger a la solución.

2.3.4. Método del gradiente conjugado con restricciones acotadas

E.A.H. Vollebregt es autor de varios *papers* relativos a un método numérico conocido como *Bound-Constrained Conjugate Gradient (BCCG)* (Vollebregt, 2014; Vollebregt, 2013). El método del gradiente conjugado con restricciones acotadas es una técnica diseñada para abordar problemas de programación cuadrática convexa (QP) sujetos a restricciones de contorno. Estos problemas son frecuentes en áreas como la mecánica de contacto elástico, donde es fundamental que las soluciones respeten límites físicos, como la *no negatividad* de las tensiones de contacto. Este tipo de problemas suele expresarse mediante la formulación matemática:

$$\min_{\boldsymbol{x}} \phi(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{x}, \qquad sujeto\ a\ \boldsymbol{x} \geq 0. \tag{69}$$

donde $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ es una matriz simétrica definida positiva, $\boldsymbol{b} \in \mathbb{R}^n$ el vector de términos independientes y $\boldsymbol{x} \in \mathbb{R}^n$ el vector de incógnitas. En el contexto de los problemas de contacto elástico, la matriz $\boldsymbol{A}$ suele ser una matriz grande, dispersa y cuyos elementos son no negativos, lo que añade complejidad al problema.

De forma concreta, el método se basa en el uso combinado de un algoritmo de conjunto activo (*active set*) y el gradiente conjugado (CG), lo que resulta en un algoritmo muy eficiente computacionalmente y que permite resolver, bajo adaptación, el problema normal de contacto definido según la expresión (60).

Varios autores han combinado técnicas de a*ctive set* con el gradiente conjugado para abordar el problema de contacto. Un enfoque directo consiste en utilizar el método del gradiente conjugado como una iteración interna para resolver los sistemas reducidos que se surgen en el proceso de resolución. Por otro lado, Polonsky y Keer (1999) desarrollaron un enfoque más integrado, proponiendo un algoritmo que opera en un solo bucle, denominado *Extended Conjugate Gradient*. Este algoritmo también ha sido adoptado, con algunas modificaciones, por Sainsot y Lubrecht (2011).

No obstante, en las diversas propuestas existentes en la literatura para la resolución de problemas cuadráticos (QPs) mediante el uso del gradiente conjugado, se identifican dos limitaciones claras:

1. **Cautela excesiva en la expansión del *active set*:** Los métodos tradicionales tienden a ser extremadamente cuidadosos al actualizar el conjunto activo, es decir, las variables que se encuentran en los límites. Este enfoque conservador puede resultar en un número elevado de iteraciones, prolongando el tiempo necesario para alcanzar la solución óptima.
2. **Reinicios frecuentes del CG:** Cada vez que el a*ctive set* se actualiza, los métodos tradicionales suelen requerir el reinicio del gradiente conjugado. Este reinicio interrumpe el proceso

iterativo, provocando la pérdida de la información acumulada en las iteraciones anteriores y ralentizando considerablemente la convergencia del método.

Algunas diferencias esenciales entre las estrategias evaluadas por E.A.H. Vollebregt, se ilustran en la Figura 18. En ella se muestra la convergencia de los diferentes métodos para un determinado problema de contacto.

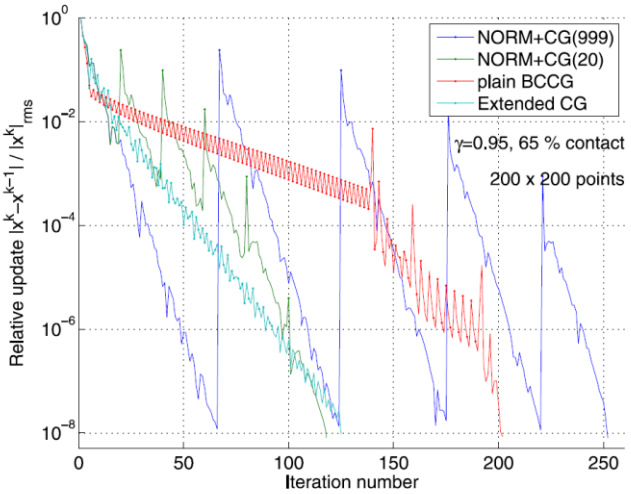

Figura 18. Variación del error relativa con el número de iteraciones (Vollebregt, 2014).

De esta figura puede deducirse que el enfoque directo NORM+CG(999) emplea un número bastante elevado de iteraciones para resolver cada sistema reducido antes de pasar al siguiente. El uso de soluciones aproximadas mediante algoritmos de resolución numéricos permite ahorrar mucho trabajo, aunque bajo cierto compromiso: si el CG se reinicia con demasiada frecuencia, afectará negativamente a la tasa de convergencia. Por último, el *Extended Conjugate Gradient* verifica las restricciones en cada iteración sirviéndose de un enfoque heurístico para evitar el reinicio del proceso de resolución del Gradiente Conjugado. Esto lo convierte en el más eficiente junto a NORM+CG(20).

El método *Extended CG* funciona de manera bastante efectiva. Una ventaja en comparación con NORM+CG es que no depende de un número de iteraciones que es distinto según el problema que se está resolviendo. No obstante, su tasa de convergencia es lineal, en contraste con la convergencia *superlineal* de otros métodos. Además, el enfoque heurístico requiere un segundo producto matriz-vector por iteración, lo que aproximadamente duplica la cantidad de trabajo por iteración con respecto al método NORM+CG.

Un problema identificado por Sainsot & Lubrecht (2011) es que el método del gradiente conjugado extendido tiende a estancarse cuando se buscan resultados muy precisos. Además, al combinar el CG con una estrategia de conjunto activo, como por ejemplo anulando los elementos negativos $p_i^k < 0$ de las iterados del CG, se podrían introducir ciclos[5] innecesarios. Este riesgo también

[5] Se entiende por ciclos al hecho de que el algoritmo active set, tras un cierto tiempo, vuelva a un punto que ya había visitado antes, de modo que no se encuentra la solución y comienza a repetirse sin final.

está presente en el algoritmo NORM (CONTACT) de Kalker (1990), donde se agregan o eliminan múltiples restricciones simultáneamente. Este peligro es real, como se demuestra en Vollebregt (2013) mediante una matriz 3×3. Esta situación ha llevado a numerosos autores a desarrollar algoritmos en los que se pueda garantizar la convergencia (Moré & Toraldo, 2006; Hager & Zhang, 1991). En general, se emplean técnicas complejas para manejar múltiples restricciones simultáneamente, asegurando así que en cada paso se logre garantizar la disminución en los valores $\phi(\boldsymbol{p}^k)$ de la función a minimizar.

Vollebregt sostiene que, en el contexto del problema de contacto, es posible utilizar procedimientos mucho más simples, ya que prácticamente cualquier técnica puede emplearse sin riesgo de introducir ciclos ni de comprometer significativamente la convergencia del algoritmo. Esto se debe, según Vollebregt, a la positividad de todos los elementos de la matriz $\boldsymbol{A}$, lo que contribuye a la estabilidad del proceso. A partir de esta premisa, Vollebregt presenta el algoritmo llamado *Bound-Constrained Conjugate Gradient* (BCCG) combinando tres elementos clave:

- El principal componente es la utilización de una proyección simple, $p_i^k = \max(0, \tilde{p}_i^k)$, para mantener la *factibilidad* de los iterados. La convergencia es lenta si el CG se reinicia cada vez que se utiliza esta proyección (*plain BCCG,* Figura 18) pues el algoritmo pasa entonces a comportarse como el método *steepest descent* la mayor parte del tiempo.
- El segundo elemento es la utilización de estrategias diseñadas para la minimización no lineal, como la fórmula de Polak-Ribiere (Polak & Ribière, 1969; Press, Teukolsky, Vetterling, & Flannery, 1992). Este enfoque heurístico sacrifica la convergencia superlineal del Gradiente Conjugado. Por ende, el CG se reinicia cuando el conjunto activo ha sido estable durante un determinado periodo.
- El ingrediente final es la utilización de iteraciones internas en las que no se verifican las restricciones. Esto reduce la carga de trabajo ante un área potencial de contacto grande, pues en dichas iteraciones internas se puede ignorar el área exterior. Por lo tanto, se pueden utilizar productos matriz-vector con una matriz reducida $\boldsymbol{A}_{CC}$ en lugar de la matriz completa $\boldsymbol{A}$[6].

Combinando estos ingredientes, Vollebregt (2013) presenta el algoritmo denominado Enhanced BCCG(K), BCCG(K), o BCCG de forma simplificada. Este algoritmo se fundamenta en los tres principios clave previamente discutidos: la implementación de una proyección simple, la utilización de las fórmulas de Polak-Ribière (Polak & Ribière, 1969) y un bucle interno que itera sin verificar continuamente las restricciones. Esta estructura forma la base del método del gradiente conjugado con restricciones acotadas.

Este enfoque es una extensión directa del método de los gradientes conjugados, que incorpora modificaciones centradas en la gestión de los conjuntos de variables libres y restringidas (correspondientes a las variables del área de contacto y a las de fuera de ella, respectivamente, en el caso del problema normal), así como en la utilización de un paso propio del algoritmo *steepest descent* cuando es necesario.

[6] Nótese que el subíndice CC indica que la matriz está formada únicamente por las entradas de $\boldsymbol{A}$ asociadas a los elementos pertenecientes al área de contacto.

En términos generales, este algoritmo puede interpretarse como un proceso iterativo compuesto por una iteración interna y una iteración externa. La iteración externa implementa la estrategia de conjunto activo (*active set*), encargándose de la actualización de las variables que necesitan ajustarse y asegurando que las soluciones se mantengan dentro de los límites impuestos por las restricciones. La iteración interna aplica el método del gradiente conjugado a un problema reducido, formado por las variables libres, es decir, aquellas que pertenecen al área de contacto en ese momento. De este modo, se preservan propiedades esenciales como la optimalidad y la ortogonalidad. Si durante la iteración interna se detecta un cambio en el conjunto activo (es decir, en el área de contacto), la iteración interna se detiene. Esto permite que la iteración externa actualice las variables necesarias antes de reiniciar el proceso interno, comenzando nuevamente con un paso propio del método *steepest descent*.

De forma más específica, el algoritmo BCCG(K) sigue los siguientes pasos:

1. Dada la matriz simétrica positiva definida $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, el vector de términos independientes $\boldsymbol{b} \in \mathbb{R}^{n}$, y una estimación inicial $\tilde{\boldsymbol{x}}^0 \in \mathbb{R}^{n}$. Se construye entonces un punto factible proyectando, elemento a elemento, mediante:

$$\boldsymbol{x}^0 = \max(0, \tilde{\boldsymbol{x}}^0). \tag{70}$$

2. Se inicia el proceso iterativo con $k = 1,2,3\ldots$ utilizando $\boldsymbol{x}^{k-1}$ y $\boldsymbol{g}^{k-1} = \boldsymbol{A}\boldsymbol{x}^{k-1} - \boldsymbol{b}$ dados por la iteración anterior.
3. Se determinan el conjunto libre y restringido iniciales mediante la condición siguiente:

$$\mathcal{B}^k = \left\{ i \in \mathcal{N} \mid x_i^{k-1} = 0 \wedge g_i^{k-1} > 0 \right\}, \qquad \mathcal{F}^k = \mathcal{N} \backslash \mathcal{B}^k. \tag{71}$$

4. Se define una *dirección de búsqueda* $\boldsymbol{p}^k$, nulo para las componentes de las variables restringidas:
    a. Se define el residuo para el subproblema correspondiente a las incógnitas libres:

$$r_i^{k-1} = \begin{cases} -g_i^{k-1}, & i \in \mathcal{F}^k, \\ 0, & i \in \mathcal{B}^k. \end{cases} \tag{72}$$

    b. Si el conjunto libre ha cambiado, $\mathcal{F}^k \neq \mathcal{F}^{k-1}$, o si $\tilde{\boldsymbol{x}}^{k-1} \neq \boldsymbol{x}^{k-1}$ (necesita proyección), o si es la primera iteración ($k = 1$), se reinicia el gradiente conjugado tomando la dirección correspondiente a la del algoritmo *steepest descent*:

$$\boldsymbol{p}^k = \boldsymbol{r}^{k-1}, \tag{73}$$

    c. Por el contrario, si $k > 1$, $\mathcal{F}^k = \mathcal{F}^{k-1}$ y $\tilde{\boldsymbol{x}}^{k-1} \equiv \boldsymbol{x}^{k-1}$ se usa el gradiente conjugado modificado mediante las fórmulas de Polak-Ribière:

$$\beta_{PR}^k = \left(\boldsymbol{r}^{k-1}, \boldsymbol{r}^{k-1} - \boldsymbol{r}^{k-2}\right) / \left(\boldsymbol{r}^{k-2}, \boldsymbol{r}^{k-2}\right), \tag{74}$$

$$\boldsymbol{p}^k = \boldsymbol{r}^{k-1} + \max\left(0, \beta_{PR}^k\right) \boldsymbol{p}^{k-1}. \tag{75}$$

d. Se establece $\boldsymbol{p}_i^k = 0$, para las variables acotadas $i \in \mathcal{B}^{k-1}$.

5. Se computa el tamaño de paso óptimo α^k , de nuevo según Polak-Ribièr, sin atender a las restricciones:

$$\boldsymbol{q}^k = \boldsymbol{A}^k \boldsymbol{p}^k, \qquad y \;\; \boldsymbol{q}_{\mathcal{B}}^k = 0, \tag{76}$$

$$\alpha_{PR}^k = \left(\boldsymbol{r}^{k-1}, \boldsymbol{p}^k\right) / \left(\boldsymbol{p}^k, \boldsymbol{q}^k\right), \tag{77}$$

$$\widetilde{\boldsymbol{x}}^k = \boldsymbol{x}^{k-1} + \alpha_{PR}^k \boldsymbol{p}^k. \tag{78}$$

El vector $\widetilde{\boldsymbol{x}}^k$ se le llama iterado intermedio.

6. Se genera el nuevo iterado $\boldsymbol{x}^k$ proyectando $\widetilde{\boldsymbol{x}}^k$ mediante:

$$\boldsymbol{x}^k = \max\left(0, \widetilde{\boldsymbol{x}}^k\right), \tag{79}$$

7. Calcular el gradiente:
   - Si $\widetilde{\boldsymbol{x}}^k \equiv \boldsymbol{x}^k$, se calcula:

$$\boldsymbol{g}^k = \boldsymbol{g}^{k-1} + \alpha_{PR}^k \boldsymbol{q}^k. \tag{80}$$

     En este caso q_i^k puede ser no nulo para $i \in \mathcal{B}^k$.
   - En caso contrario, es necesario un segundo producto matriz-vector:

$$\boldsymbol{g}^k = \boldsymbol{A}\boldsymbol{x}^k - \boldsymbol{b}. \tag{81}$$

8. Comprobar la convergencia: si $\boldsymbol{g}_i^k > 0, \;\; \forall i \in \mathcal{B}^k$ y el residuo es lo suficientemente pequeño en el conjunto libre $\mathcal{F}^k$ entonces el algoritmo debe terminar. En caso contrario, se vuelve al paso 1.

2.4. Conclusión

El contexto teórico presentado en este capítulo proporciona una visión integral y detallada del fenómeno de contacto rueda-carril. A través del análisis exhaustivo del contacto de solidos en rodadura, los principios de la mecánica de contacto, el modelo cinemático y la cuasiidentidad elástica, se han sentado las bases necesarias para comprender adecuadamente este fenómeno.

Asimismo, se han revisado las principales teorías empleadas en la actualidad para abordar el problema del contacto, como el método CONTACT, el modelo normal de Hertz y el algoritmo FASTSIM, que son esenciales para la resolución precisa del problema.

Finalmente, se ha descrito la teoría subyacente a los algoritmos de resolución del problema de contacto normal que se implementarán en el próximo capítulo. Entre estos algoritmos se incluyen el método NORM, una aproximación basada en distribuciones de tensiones polinómicas o elípticas utilizando el método NORM como base, y el método del gradiente conjugado con restricciones acotadas.

Este marco teórico es fundamental para guiar la implementación práctica que se abordará en el siguiente capítulo, utilizando la codificación en Fortran para que los métodos programados sean precisos y eficientes. Esto permitirá aplicar, de manera efectiva, las teorías revisadas al análisis del contacto rueda-carril en diversas condiciones. En definitiva, el contexto teórico proporciona el cimiento esencial sobre el cual se basa todo el trabajo de implementación y análisis posterior.

3. METODOLOGÍA

3.1. Introducción

En este TFM se han desarrollado e implementado tres algoritmos en Fortran, con el objetivo de obtener un código que sea lo más simple, estable y eficiente posible. Los códigos correspondientes se encuentran en el ANEXO 1, disponibles para su revisión y análisis detallado.

El propósito principal de este trabajo es evaluar el rendimiento de tres enfoques distintos para resolver el problema de contacto normal entre rueda y carril en ausencia de rozamiento. Los métodos considerados son: la variante NORM del método CONTACT, la aproximación de la distribución de tensiones en la dirección de rodadura mediante una base elíptica o polinómica, y el método del gradiente conjugado con restricciones acotadas. Cada uno de estos enfoques ha sido implementado con estrategias específicas para abordar las particularidades del problema de contacto, garantizando soluciones precisas y fiables. Además de analizar las particularidades de cada enfoque, también se destacan las generalidades que comparten, pues todos ellos se basan en principios comunes que guían la implementación.

A continuación, se presentan en detalle los tres algoritmos, junto con las consideraciones e hipótesis que influyeron en su implementación. Se describirá cómo cada método fue adaptado para optimizar la resolución, destacando tanto sus características únicas como los elementos que los vinculan.

3.2. Implementación de los algoritmos

3.2.1. Objetivos de la implementación

Para lograr una implementación eficaz del código, es esencial establecer objetivos bien definidos y aplicar buenas prácticas de programación. Cada etapa debe estar estructurada de manera que garantice la obtención de un código eficiente y estable. Dado que los algoritmos serán evaluados en función de su precisión, el uso de recursos y el tiempo de CPU, es clave optimizar el rendimiento sin sacrificar la exactitud. Por tanto, los objetivos para asegurar una implementación óptima deben incluir:

1. **Precisión en los cálculos:** El primer objetivo es garantizar que el código, en todas sus etapas, realice cálculos con alta precisión. La exactitud es fundamental, ya que cualquier error podría comprometer la validez de los resultados y afectar el estudio del contacto rueda-carril.
2. **Optimización del tiempo de CPU:** Minimizar el tiempo de CPU es fundamental para mejorar la eficiencia del código, especialmente en el benchmarking. Un código optimizado reduce los costos computacionales y permite realizar múltiples simulaciones en menor tiempo, facilitando estudios más amplios y detallados.
3. **Uso eficiente de recursos:** El código debe hacer un uso eficiente de los recursos del sistema, especialmente la memoria, asegurando un buen rendimiento computacional.

4. **Escalabilidad:** El código debe ser escalable y parametrizable para adaptarse a diferentes escenarios de contacto o problemas con un mayor número de elementos, manteniendo un buen rendimiento incluso en situaciones más complejas.
5. **Robustez:** El código debe ser robusto, capaz de manejar diferentes configuraciones de contacto, tamaños de datos y variaciones en las condiciones de entrada sin fallos. La robustez garantiza que el código funcione correctamente bajo diversas condiciones, proporcionando resultados consistentes y fiables.
6. **Facilidad de mantenimiento y extensibilidad:** El diseño del código debe facilitar su mantenimiento y futuras ampliaciones. Una estructura clara y bien documentada permitirá la incorporación de nuevas funcionalidades o ajustes sin necesidad de grandes modificaciones.

En resumen, estos objetivos no solo aseguran una implementación efectiva, sino que también son fundamentales para la evaluación comparativa del código mediante benchmarking. Garantizan que el código mantenga una alta precisión en todos sus cálculos, optimice el uso de CPU y recursos, sea robusto frente a diversas configuraciones de entrada, y ofrezca escalabilidad para abordar problemas de mayor complejidad. La consecución de estos objetivos permitirá que el código se adapte eficazmente a distintos escenarios de contacto sin comprometer el rendimiento, maximizando su eficacia y facilitando futuras mejoras y extensiones.

3.2.2. Estrategias de optimización y criterios en la implementación

Como se mencionó previamente, una de las principales prioridades era reducir el tiempo de ejecución y el uso de recursos, manteniendo una alta precisión en los cálculos. Para lograrlo, se partió de un código base común, que fue optimizado exhaustivamente antes de implementar los algoritmos específicos. A lo largo del proceso se generaron hasta 16 versiones, cada una incorporando mejoras en el tiempo de ejecución y el uso de CPU, ajustando cálculos, eliminando variables innecesarias y optimizando los algoritmos de resolución.

Para asegurar una implementación eficaz y cumplir con los objetivos propuestos, se consideraron los siguientes puntos clave:

1. **Selección del método numérico más eficiente**: Durante la implementación, se probaron distintos métodos numéricos para la resolución de los sistemas matriciales. A lo largo de las distintas versiones de los códigos, se probaron métodos como la eliminación de Gauss, la descomposición LU y métodos iterativos como el Gradiente Conjugado.
2. **Reducción de la complejidad computacional:** Se implementaron optimizaciones dirigidas a disminuir tanto el tiempo de ejecución como el consumo de memoria, empleando estructuras de datos más eficientes, eliminando cálculos redundantes, omitiendo productos de variables nulas y reduciendo accesos innecesarios a la memoria.
3. **Parametrización del código:** El código se parametrizó para adaptarse automáticamente a las variaciones del problema, ajustando de manera eficiente tanto el número de elementos como las variables clave. Así, puede responder a cambios en el tamaño de las matrices y en el área de contacto sin requerir intervenciones manuales.
4. **Gestión eficiente de la memoria**: Se centralizaron todas las matrices y vectores en un solo vector de gran capacidad, optimizando el uso de memoria y facilitando el acceso mediante

punteros. Esta estrategia redujo la sobrecarga en la transferencia de datos entre subrutinas, maximizando el rendimiento gracias a las capacidades de Fortran para gestionar la memoria eficientemente.

5. ***Profiling* y optimización del código:** Se utilizaron herramientas de *profiling*[7] para identificar cuellos de botella en el código. Las secciones que consumían más tiempo de CPU fueron optimizadas. En cada versión de los códigos, se recogió el tiempo de cálculo de cada subrutina para determinar si eran necesarias nuevas optimizaciones y cómo funcionaban las aplicadas.
6. **Validación y verificación:** una vez optimizado el código, se comparaban los resultados con los obtenidos por CONTACT en MATLAB. Para asegurar la precisión, se calculaba el error relativo, incluso cuando los resultados eran similares, con el fin de verificar que dicho error fuera mínimo.
7. **Documentación y buenas prácticas de programación:** Se mantuvo una documentación clara y detallada, que incluía la descripción de los algoritmos y la estructura del código. Se siguieron buenas prácticas de programación, como la modularidad, la reutilización de código y el control de versiones, lo que garantiza que el código sea fácilmente mantenible y escalable.
8. **Evaluación continua:** Se realizaron evaluaciones periódicas del código midiendo el tiempo de CPU, el tiempo de reloj del código, el uso de memoria y la precisión de los resultados. Esta evaluación continua permitió identificar áreas de mejora y validar las optimizaciones implementadas.

3.2.3. Metodología de implementación

Las estrategias explicadas en el apartado anterior fueron adoptadas mayoritariamente a lo largo de la generación de los códigos. Una vez el código estuvo terminado, surgieron optimizaciones adicionales debido a la adaptación particular de cada algoritmo.

Es relevante señalar que la programación inicial se realizó en lenguaje C, con el cual se generaron las primeras 12 versiones del código base. Tras una revisión por parte de los responsables del proyecto, se concluyó que Fortran era una mejor opción para adaptar el código al propósito buscado y facilitar su optimización posterior. A partir de ese momento, las versiones 13 a 16, así como los algoritmos, fueron completamente implementados, evaluados y optimizados en Fortran, culminando en las versiones finales que sustentan este trabajo. El código base común se implementó conjuntamente con el algoritmo NORM, lo que permitió que, una vez programado este, los otros dos algoritmos se implementaran aprovechando el código ya desarrollado.

En términos generales, todos los programas han sido implementados siguiendo un enfoque similar, lo que permite resumir la metodología empleada de la siguiente manera:

1. **Estudio y revisión previa del modelo a implementar:** Antes de proceder con la implementación, es esencial tener un entendimiento detallado del método a programar. Por

[7] El *profiling* es el proceso de analizar el rendimiento de un programa para identificar cuellos de botella y áreas que necesitan optimización. A través del *profiling*, se recopilan datos sobre el tiempo de ejecución y el uso de recursos de diferentes partes del programa.

esta razón, se realizó una lectura exhaustiva de los recursos y fundamentos teóricos pertinentes. En el caso del método NORM (CONTACT), se analizó la teoría del contacto rueda-carril, descrita en el apartado 2.2 de este trabajo, y posteriormente se implementó el método según lo detallado en los apartados 2.2.6.3.3 y 2.3.2.

Para la implementación de los otros dos algoritmos, se llevó a cabo una revisión bibliográfica exhaustiva, que incluyó el estudio de los artículos científicos de los cuales se derivan las teorías expuestas en las secciones 2.3.3 y 2.3.4.

2. **Implementación previa del modelo en MATLAB**: Esta fase consistía en programar el código de manera preliminar utilizando una sintaxis similar o una simplificada, como la que ofrece MATLAB para gestionar matrices. MATLAB, al ser un entorno interactivo, permite escribir y ejecutar el código en tiempo real, facilitando la visualización inmediata de los resultados y la realización de ajustes sobre la marcha. Esta flexibilidad favoreció la experimentación y aceleró la implementación de los algoritmos.
3. **Traducción a Fortran**: Una vez completada la implementación en MATLAB, se procedía a la traducción del código a Fortran ajustando la sintaxis y optimizando las estructuras de datos para adecuarlas al lenguaje.
4. **Testeo del código:** Tras la traducción a Fortran, se realizaban pruebas exhaustivas para verificar el correcto funcionamiento del código y garantizar la consistencia en los resultados.
5. **Optimización del código:** Se aplicaban diversas técnicas de optimización, apoyadas por el uso de *profiling manual* para identificar áreas de mejora. Posteriormente, se realizaron modificaciones específicas en el código con el objetivo de incrementar su rendimiento y eficiencia.
6. **Verificación de los resultados**: Finalmente, se comparaban los resultados obtenidos en Fortran con los obtenidos en MATLAB para asegurar que la traducción y optimización eran correctas y que los resultados eran consistentes y precisos.

3.2.4. Implementación de los algoritmos

3.2.4.1. Introducción

En este apartado se aborda, tanto de forma teórica como práctica, la implementación de los algoritmos y las optimizaciones realizadas durante este proyecto para llegar a sus versiones definitivas. Los códigos de los algoritmos han sido programados en Fortran y pueden encontrarse en el ANEXO 1 para su revisión y estudio individual.

Recuérdese que la propuesta es evaluar las prestaciones de estos algoritmos de resolución del problema normal de contacto rueda-carril sin rozamiento. Los tres algoritmos postulados en este trabajo para la resolución del problema normal de contacto rueda-carril se denominarán en adelante:

1. **NORM:** El método NORM del software CONTACT (Kalker, 1990).
2. **NORM+E/P:** El modelo NORM combinado con una aproximación basada en el uso de distribuciones de tensiones elípticas o polinómicas en la dirección $\boldsymbol{X_1}$ (Giner Navarro et al., 2023).
3. **BCCG (K):** El método del gradiente conjugado con restricciones acotadas (Vollebregt, 2013).

3.2.4.2. Preámbulo

En el marco de la resolución de sistemas lineales del tipo $\boldsymbol{Ax} = \boldsymbol{b}$, que emergen del análisis del problema normal de contacto, la selección del método de resolución es un aspecto crítico que impacta directamente en la precisión y eficiencia computacional del proceso. Entre la amplia gama de técnicas disponibles, se han escogido dos métodos fundamentales por su idoneidad en relación con las características del problema y su implementación: la Eliminación de Gauss (con sustitución regresiva y pivotaje parcial) y el método de Gradiente Conjugado.

Además de la selección de los métodos, la gestión eficiente de la memoria ha sido un aspecto fundamental en la implementación de todos los códigos desarrollados para este proyecto. Dado que los métodos de resolución, como la eliminación de Gauss y el gradiente conjugado, junto con otros procedimientos auxiliares, operan con grandes volúmenes de datos, una estrategia de memoria bien diseñada es crucial para optimizar el uso de los recursos computacionales. Esto no solo permite manejar sistemas a gran escala de manera eficiente, sino que también mejora la rapidez y confiabilidad de las soluciones obtenidas.

La Eliminación de Gauss con sustitución regresiva y pivotaje parcial (GE) se destaca por su capacidad para proporcionar soluciones exactas en aritmética de punto flotante[8], siendo especialmente adecuada para sistemas de tamaño moderado donde la estabilidad numérica es imperativa. Este método es clave para la confección del método clásico de NORM (CONTACT), que constituye la base del tratamiento matemático del problema normal de contacto en su formulación más tradicional. El uso del pivotaje parcial no es simplemente una mejora, sino una necesidad en escenarios donde la matriz $\boldsymbol{A}$ podría presentar elementos diagonales de magnitud reducida que, de no ser adecuadamente gestionados, podrían comprometer la precisión del resultado. Este enfoque garantiza la robustez del proceso de resolución, haciéndolo ideal para situaciones en las que la exactitud es un requisito ineludible.

En contraste, el método de Gradiente Conjugado (CG) ha sido seleccionado por su eficacia en la resolución de sistemas de gran escala y matrices dispersas, características comunes en la modelización de problemas de contacto. Este método iterativo es especialmente adecuado para sistemas donde la matriz $\boldsymbol{A}$ es simétrica y definida positiva, propiedades que frecuentemente también se presentan en el contexto del problema normal de contacto. Además de su capacidad para manejar grandes volúmenes de datos, el CG es más eficiente en el uso de recursos comparado con los métodos directos, como la eliminación de Gauss, ya que evita el almacenamiento y manipulación de matrices densas. En lugar de trabajar con la matriz completa, el CG realiza operaciones iterativas sobre vectores, lo que reduce significativamente el consumo de memoria y tiempo de CPU, especialmente en sistemas de gran escala.

[8] El punto flotante es un formato numérico que representa números reales usando una mantisa y un exponente, permitiendo manejar valores grandes y pequeños con una precisión limitada a aproximadamente 7-8 dígitos decimales en simple precisión y 15-16 dígitos en doble precisión. (Matemáticas Discretas, 2017)

Diversos autores han propuesto la implementación del CG para abordar estos problemas, combinándolo con estrategias de *active set* y/o optimizando su convergencia mediante técnicas de precondicionamiento, como se mencionó en el apartado 2.3.4. Estas optimizaciones no solo aceleran la convergencia, sino que también aumentan la estabilidad del método, lo que lo convierte en una opción preferente para sistemas de gran tamaño.

Ambos métodos fueron cuidadosamente seleccionados e implementados. Además, la gestión eficiente de la memoria resultó clave para maximizar el rendimiento computacional, garantizando que todos los códigos desarrollados operen con la mayor eficacia posible en cuanto al uso de recursos y velocidad de cálculo.

A continuación, se explica la gestión de memoria en términos generales junto con la formulación del método de eliminación de Gauss y el método del Gradiente Conjugado.

3.2.4.2.1. Gestión de la memoria

Desde el punto de vista de la arquitectura de la implementación, se ha adoptado una estrategia avanzada para la gestión de memoria que resulta crucial para manejar eficientemente las operaciones a gran escala requeridas por los métodos. En este contexto, se emplea un vector de gran capacidad, denominado $\boldsymbol{R}$, diseñado específicamente para contener todas las matrices y vectores involucrados en el problema, es decir, las estructuras de datos fundamentales que sustentan el cálculo. Este vector $\boldsymbol{R}$ se define al inicio del problema con un tamaño aproximado que se estima suficiente para alojar todas las estructuras de datos requeridas, evitando así la necesidad de asignar memoria dinámica durante la ejecución. Al evitar la memoria dinámica, se eliminan los posibles riesgos asociados con la fragmentación de la memoria y se garantiza un acceso más predecible y rápido a los datos, lo que es fundamental para la eficiencia en cálculos de gran escala.

A partir de esta reserva de memoria, se asignan y se definen posteriormente los índices correspondientes a los punteros que delimitan las secciones del vector que serán utilizadas para cada matriz o vector específico. Por ejemplo, en la implementación del método NORM clásico, se calculan los índices que marcan el inicio y el final de cada matriz o vector dentro del vector $\boldsymbol{R}$ tal y como indica la Figura 19.

```
NE = N1 * N2                IPD = IA + NESQR
NESQR = NE ** 2             IH2 = IPD + NE
                            IA2 = IH2 + NE
IH = 1                      IX = IA2 + NESQR
IX1 = IH + NE               IPP = IX + NE
IX2 = IX1 + N1              ITEMP = IPP + NE
IRC = IX2 + N2              IVEC = ITEMP +
IAES = IRC + NE * 2         NESQR
IT = IAES + NE              IEC = IVEC + NE
IA = IT + NESQR             IEND = IEC + NE
```

Figura 19. Definición de los índices de posición en Fortran para el método NORM clásico.

En este caso, I_{END} indica el último índice utilizado en el vector $\boldsymbol{R}$, lo que representa la dimensión total efectiva del vector, es decir, el tamaño que ocupan las estructuras de datos en la memoria. Esta asignación permite que cada subrutina trabaje directamente con las secciones del vector $\boldsymbol{R}$ que corresponden a las matrices y vectores necesarios, utilizando los punteros definidos para acceder a los datos. De este modo, se optimiza tanto la asignación como el acceso a los datos, y se asegura que el uso de la memoria sea eficiente. Además, el código incorpora un mecanismo de control que verifica que la cantidad de memoria utilizada no supere la capacidad asignada realmente al vector $\boldsymbol{R}$, garantizando así una gestión de memoria segura y eficiente.

Este enfoque en la gestión de memoria se ve complementado y potenciado por las capacidades intrínsecas de Fortran. El lenguaje permite pasar a las subrutinas el primer elemento de un bloque de memoria, como el vector $\boldsymbol{R}$ mencionado, y, a partir de esta única referencia, definir matrices y vectores completos dentro de la subrutina, especificando sus dimensiones y forma. Esto se debe a que Fortran maneja los vectores mediante direcciones de memoria, lo que significa que, al pasar el primer elemento de un vector, se transfiere la dirección base de la estructura de datos. En la subrutina, esta dirección se utiliza para reconstruir la matriz o vector con la forma y tamaño requeridos, permitiendo un acceso eficiente y directo a los datos almacenados en la memoria principal. Así, se minimiza la sobrecarga asociada con la transferencia de grandes volúmenes de datos entre distintas partes del programa, algo crucial en la resolución de sistemas a gran escala, donde la eficiencia y velocidad de cálculo son esenciales. De este modo, la combinación de una estrategia de memoria centralizada con las ventajas de gestión de punteros de Fortran optimiza el rendimiento y el uso de los recursos computacionales.

3.2.4.2.2. Eliminación de Gauss con sustitución regresiva y pivotaje parcial

La eliminación de Gauss es un método fundamental para resolver sistemas de ecuaciones lineales de la forma $\boldsymbol{Ax} = \boldsymbol{b}$, donde $\boldsymbol{A}$ es una matriz cuadrada, $\boldsymbol{x}$ es un vector de incógnitas, y $\boldsymbol{b}$ es un vector de términos independientes. Este método transforma el sistema inicial en uno equivalente (82), donde la matriz $\boldsymbol{A}'$ es triangular superior, lo que permite resolver el sistema fácilmente mediante sustitución hacia atrás.

$$\begin{bmatrix} a'_{11} & a'_{12} & a'_{13} & \cdots & a'_{1n} \\ 0 & a'_{22} & a'_{23} & \cdots & a'_{2n} \\ 0 & 0 & a'_{33} & \cdots & a'_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & a'_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b'_1 \\ b'_2 \\ b'_3 \\ \vdots \\ b'_n \end{bmatrix} \tag{82}$$

La notación a'_{ij} y b'_j del sistema (82) es utilizada para referirse a que los elementos ya no tienen su valor numérico original, sino que son los obtenidos tras las operaciones por filas realizadas durante el proceso de eliminación.

Formulación teórica del método

A continuación, se describe la formulación del método junto con la inclusión del pivotaje parcial y el proceso de sustitución regresiva.

1. Selección del pivote:

El proceso de selección del pivote es crucial para evitar inestabilidades numéricas, especialmente cuando se trabaja con números en punto flotante. En cada columna k de la matriz $\boldsymbol{A}$, se selecciona el pivote como el mayor valor, en valor absoluto, desde la fila k hasta la última fila de esa columna. Esto es:

$$\pi_k = \max_{i \geq k} |a_{ik}| \,, \tag{83}$$

donde π_k es el pivote correspondiente y a_{ik} el elemento de la fila i y columna k de la matriz $\boldsymbol{A}$. En caso de que el pivote no esté en la fila k, se realiza un intercambio de filas mediante pivotaje parcial.

2. Pivotaje parcial:

El pivotaje parcial es una técnica crucial para mejorar la estabilidad numérica del algoritmo. Consiste en permutar las filas de la matriz $\boldsymbol{A}$ para asegurar que el pivote tenga el valor absoluto más grande posible entre los elementos de su columna en las filas que aún no han sido eliminadas. Esto minimiza los errores de redondeo y evita divisiones por cero.

El procedimiento de pivotaje sigue, en términos generales, estos pasos:

I. **Seleccionar el pivote**: Encontrar el valor absoluto máximo en la columna k desde la fila k hasta la última.
II. **Intercambiar filas:** Se intercambia la fila actual k con la fila que contiene el valor máximo encontrado.
III. Se continua con la eliminación de Gauss utilizando la nueva fila pivote.

El uso del pivotaje parcial asegura que el sistema de ecuaciones se resuelva de manera más estable, especialmente en casos donde la matriz $\boldsymbol{A}$ tiene elementos muy pequeños o muy grandes en relación con otros elementos.

3. Eliminación:

Una vez seleccionado el pivote π_k, se realizan operaciones con las filas para eliminar los elementos debajo del pivote, convirtiendo esos elementos en ceros. Las operaciones consisten, para cada fila i por debajo de la fila k, en sustraer de la fila i una fracción de la fila k:

$$m_{ik} = \frac{a_{ik}}{a_{kk}} \,, \tag{84}$$

$$a'_{ij} = a_{ij} - m_{ik} \cdot a_{kj} \,, \tag{85}$$

donde m_{ik} es el multiplicador que anula el elemento en la posición (i, k) y a'_{ij} representa el nuevo valor del elemento en la posición (i, j) después de la eliminación.

Al finalizar la eliminación, la matriz $\boldsymbol{A}$ se ha transformado en una matriz triangular superior $\boldsymbol{A}'$, y el sistema original $\boldsymbol{Ax} = \boldsymbol{b}$ se ha reducido a un sistema equivalente $\boldsymbol{A}'\boldsymbol{x} = \boldsymbol{b}'$, donde $\boldsymbol{b}'$ es el vector de términos independientes modificado.

4. Sustitución regresiva:

Después de obtener la matriz triangular superior $\boldsymbol{A}'$, el sistema de ecuaciones puede resolverse fácilmente mediante sustitución regresiva siguiendo los pasos siguientes:

a. **Resolución del último elemento:** Comenzando desde la última fila, el último elemento de $\boldsymbol{x}$ se resuelve directamente, pues sólo está asociado a un único elemento de $\boldsymbol{U}$.

$$x_n = \frac{b'_n}{a'_{nn}}. \tag{86}$$

b. **Sustitución regresiva:** Con el ultimo valor de $\boldsymbol{x}$ conocido, se puede calcular el penúltimo x_{n-1} y así consecutivamente. Así, empelando los valores anteriormente calculados para resolver los siguientes, la formulación resulta:

$$x_i = \frac{1}{a'_{ii}}\left[b'_i - \sum_{j=i+1}^{n} a'_{ij} \cdot x_j\right]. \tag{87}$$

Este proceso se repite hasta que se hayan resuelto todas las incógnitas.

Ventajas y Limitaciones

Una de las ventajas de la eliminación de Gauss es su generalidad y aplicabilidad a una amplia gama de problemas. Es robusto para sistemas con matrices de cualquier tamaño y es fundamental en muchos algoritmos numéricos más complejos, como la factorización LU.

Sin embargo, este método tiene sus limitaciones. En primer lugar, la estabilidad numérica puede ser un problema cuando se trabaja con matrices mal condicionadas, donde pequeños errores de redondeo pueden amplificarse. Además, la complejidad temporal del método, que es $\mathcal{O}(n^3)$ para una matriz de tamaño $n \times n$, puede ser prohibitiva para sistemas extremadamente grandes, lo que ha llevado al desarrollo de variantes más eficientes y especializadas como el método de los residuos mínimos (GMRES) y las técnicas basadas en descomposición QR.

3.2.4.2.3. Método del Gradiente Conjugado

El método del gradiente conjugado es un algoritmo utilizado para resolver sistemas de ecuaciones lineales del tipo $\boldsymbol{Ax} = \boldsymbol{b}$ donde $\boldsymbol{A}$ es una matriz simétrica y definida positiva. Además, este método es especialmente útil para sistemas de gran tamaño debido a su eficiencia y a que no requiere almacenar la matriz $\boldsymbol{A}$ completa, sino únicamente operar con ella.

Para que el método sea aplicable, como se comentaba anteriormente, la matriz $\boldsymbol{A}$ debe ser:

1. Simétrica: $\boldsymbol{A} = \boldsymbol{A}^T$.
2. Definida positiva: para cualquier vector $\boldsymbol{x} \neq \boldsymbol{0}_{n\times 1}$, debe cumplirse que $\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} > 0$.

Para el caso que concierne al problema, $\boldsymbol{A}$ y las matrices que se generan durante el proceso son matrices simétricas y definidas positivas, lo que hace favorable su uso.

Fundamentos teóricos del método

El método del gradiente conjugado se basa en un conjunto de iteraciones que buscan minimizar la función cuadrática asociada al sistema de ecuaciones del problema. Para el caso que concierne a este trabajo, el sistema a resolver es $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ o similar. Así, para un vector $\boldsymbol{x} \in \mathbb{R}^n$, la función cuadrática que se desea minimizar es:

$$\boldsymbol{\Phi}(\boldsymbol{p}) = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{x}\,. \tag{88}$$

donde el punto $\boldsymbol{x}$ que minimiza $\boldsymbol{\Phi}(\boldsymbol{x})$ es la solución del sistema $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$.

El método comienza con un vector inicial $\boldsymbol{x}^0$ y procede generando una sucesión de vectores $\boldsymbol{x}^k$ que convergen a la solución, tal y como enseña la Figura 20.

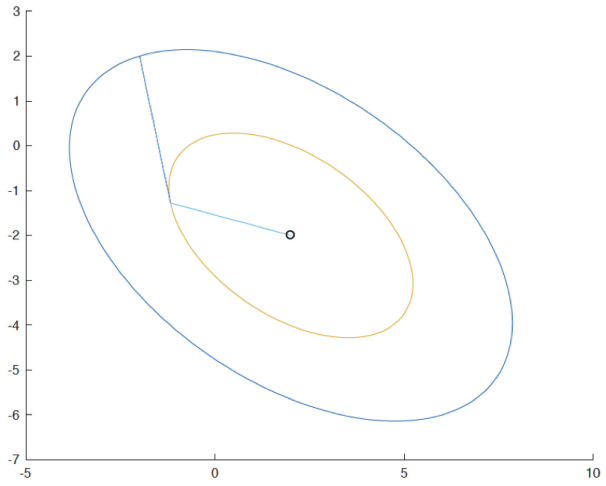

Figura 20. Iteraciones del Gradiente Conjugado (Apuntes de métodos matemáticos, 2019).

Así, los pasos que sigue el gradiente conjugado para converger hacia la solución del sistema de ecuaciones son:

1. Inicialización:
   - Seleccionar una aproximación inicial de la solución $\boldsymbol{x}^0$. En este trabajo se ha partido del vector nulo $\boldsymbol{0}_{n\times 1}$.
   - Calcular el residuo inicial $\boldsymbol{r}^0 = \boldsymbol{h} - \boldsymbol{A}\boldsymbol{x}^0$.
   - Establecer el vector de búsqueda inicial $\boldsymbol{v}^0 = \boldsymbol{r}^0$
   - Definir el umbral de tolerancia $\epsilon = \tau^2 \cdot l$, donde $l = \boldsymbol{b}^T \cdot \boldsymbol{b}$ y τ es un parámetro pequeño (del orden de $1.0 \cdot 10^{-13}$).

2. Iteración k:
   a. Se calcula el escalar α_k que minimiza $\boldsymbol{\Phi}\left(\boldsymbol{x}^{k+1}\right)$ mediante:

$$\alpha_k = \frac{\boldsymbol{r}^{k^T}\boldsymbol{r}^k}{\boldsymbol{v}^{k^T}\boldsymbol{A}\boldsymbol{v}^k}. \tag{89}$$

   b. Se actualiza el vector $\boldsymbol{x}^{k+1}$ con:

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \alpha_k \boldsymbol{v}^k. \tag{90}$$

   c. Se calcula el nuevo residuo $\boldsymbol{r}^{k+1}$:

$$\boldsymbol{r}^{k+1} = \boldsymbol{r}^k - \alpha_k \boldsymbol{A}\boldsymbol{v}^k. \tag{91}$$

   d. Se evalúa si la norma del residuo $\boldsymbol{r}^{k+1}$ es suficientemente pequeña, verificando la *condición de convergencia*:

$$\left\|\boldsymbol{r}^k\right\|^2 \leq \epsilon\,. \tag{92}$$

   Si se cumple la condición, detener el algoritmo, ya que se ha alcanzado la solución.

   e. Si no se ha alcanzado la convergencia, actualizar el *vector de búsqueda* $\boldsymbol{v}^{k+1}$:

$$\boldsymbol{v}^{k+1} = \boldsymbol{r}^{k+1} + \beta_k \boldsymbol{v}^k, \tag{93}$$

   donde β_k se calcula como:

$$\beta_k = \frac{\boldsymbol{r}^{k+1^T}\boldsymbol{r}^{k+1}}{\boldsymbol{r}^{k^T}\boldsymbol{r}^k}. \tag{94}$$

3. Bucle: los pasos del 2.a al 2.e se repiten para cada iteración k, incrementando k en cada paso, hasta que se cumpla la condición de convergencia.

Una variante del método permite realizar un número controlado de iteraciones. En esta variante, el código se detiene al alcanzar un número máximo de iteraciones predefinido o al lograr la convergencia antes de llegar a ese límite (punto 2.d). Sin embargo, optar por este enfoque puede requerir un ajuste del número de iteraciones según las necesidades de cada código específico. Además, implica un compromiso en la precisión del resultado, salvo que se alcance la convergencia antes de llegar al número máximo de iteraciones. Debido a ello, se optó por permitir que el método determinara el número de iteraciones necesarias en cada caso, priorizando la precisión. El único inconveniente de esta decisión es un posible aumento en el tiempo de ejecución.

Ventajas y Desventajas del Método de Gradiente Conjugado

El método de gradiente conjugado es una técnica eficaz para resolver sistemas de ecuaciones lineales, especialmente aquellos con matrices grandes y dispersas. Este método es particularmente útil cuando se trata de matrices simétricas y definidas positivas, tal y como ya se ha comentado.

Una de las principales ventajas del método de gradiente conjugado es su eficiencia computacional. En cada iteración, el costo computacional principal se debe a la multiplicación de la matriz por un vector, lo que tiene una complejidad de $\mathcal{O}(n^2)$ para matrices densas, como la del problema de contacto, donde n es la dimensión de la matriz. Esta característica lo convierte en una opción atractiva para grandes sistemas dispersos, ya que no es necesario almacenar la matriz completa, sino solo los vectores relevantes. Esto también reduce significativamente los requerimientos de memoria. Shewchuk (1994) discute ampliamente este aspecto del método de gradiente conjugado en su informe técnico.

Sin embargo, el método de gradiente conjugado también tiene limitaciones. Su rendimiento depende en gran medida del número de condición de la matriz: si la matriz está mal condicionada, el número de iteraciones necesarias para alcanzar la convergencia puede aumentar considerablemente, lo que afecta la eficiencia global del método. Además, el método estándar está diseñado para matrices simétricas y definidas positivas, por lo que para matrices que no cumplen con estas condiciones, se requieren adaptaciones o métodos alternativos como el gradiente conjugado biconjugado.

Otro punto a considerar es la sensibilidad del método de gradiente conjugado a los errores de redondeo. En cálculos numéricos, especialmente con sistemas mal condicionados, estos errores pueden acumularse a lo largo de muchas iteraciones, afectando la precisión de la solución obtenida. Por ello, el método no siempre es la mejor opción para todos los problemas; sin embargo, su efectividad puede mejorarse mediante el uso de precondicionadores.

3.2.4.3. NORM

3.2.4.3.1. Implementación

Con el método NORM, una vez formulado el sistema matricial (60), la resolución del problema de contacto se lleva a cabo evaluando el signo de las tensiones para determinar el área de contacto, así como la distribución de la tracción normal y la fuerza total, siguiendo los pasos descritos en la sección 2.3.2. Sin embargo, la implementación del código requiere ciertas adaptaciones específicas, las cuales se detallan a continuación.

Como se comentó anteriormente, la matriz $\boldsymbol{A}$ del sistema $\boldsymbol{h} = \boldsymbol{A}\boldsymbol{p_d}$ es una matriz Toeplitz, simétrica, definida positiva y de dimensión $N_e \times N_e$[9]. Esta matriz se construye a partir de los coeficientes de influencia D_{I3}^{J3}, y sus entradas a_{ij} dependen de la posición relativa $x_1 - x_2$ de los elementos en el área potencial de contacto. Basándose en esta idea, se genera la matriz $\boldsymbol{A_{es}}$ de la siguiente manera:

$$\boldsymbol{A_{es}} = 2 \cdot \boldsymbol{D}^{(1)}, \tag{95}$$

[9] $N_e = N_1 \cdot N_2$, siendo N_1 y N_2 el número de elementos de contacto en la dirección de rodadura y en la dirección lateral, respectivamente.

donde $\boldsymbol{D^{(1)}}$ es la matriz de coeficientes de influencia D_{I3}^{13}, es decir, una matriz de tamaño $N_1 \times N_2$ cuyas entradas $d_{ij}^{(1)}$ se definen a partir de los coeficientes de influencia del elemento 1 respecto a cualquier elemento I del área potencial de contacto. El cálculo de estas entradas se realiza en el código según la ecuación (4), sustituyendo x_1 por $x_1(i) - x_1(1)$ y x_2 por $x_2(j) - x_2(1)$, y tomando $p_3 = 1$. De esta manera, $\boldsymbol{A_{es}}$ es la matriz de coeficientes de influencia del primer elemento del área potencial de contacto con respecto al resto de elementos de esta.

Para crear la matriz $\boldsymbol{A}$ a partir de la matriz $\boldsymbol{A_{es}}$, se utilizan las entradas de esta última de la siguiente forma:

$$\boldsymbol{A}(i,j) = \boldsymbol{A_{es}}(r_o, c_o), \qquad \forall i,j \in N_e, \tag{96}$$

donde:

$$r_o = |\boldsymbol{Rc}(j,1) - \boldsymbol{Rc}(i,1)| + 1, \tag{97}$$

$$c_o = |\boldsymbol{Rc}(j,2) - \boldsymbol{Rc}(i,2)| + 1, \tag{98}$$

donde $\boldsymbol{Rc}$ es una matriz de tamaño $N_e \times 2$ que almacena la posición de cada elemento en el área de contacto. Por ejemplo, para el elemento 1, la primera columna de $\boldsymbol{Rc}$ contiene la fila que ocupa el elemento en la malla y la segunda columna contiene la columna correspondiente. Así, r_o y c_o representan las posiciones relativas en términos de fila y columna entre dos elementos, respectivamente. El símbolo $|\cdot|$ denota el valor absoluto.

Después de esto, queda por determinar el vector $\boldsymbol{h}$, que puede construirse de forma relativamente inmediata. Aunque usualmente la distancia entre superficies indeformadas para cada elemento se proporciona mediante una matriz $\boldsymbol{H} \in \mathbb{R}^{N_1 \times N_2}$, es importante considerar la forma en que Fortran maneja la memoria.

Fortran es un lenguaje de almacenamiento en memoria por columnas (*column-major*), lo que significa que los elementos de una matriz se almacenan secuencialmente por columnas. Por ejemplo, una matriz $\boldsymbol{S} \in \mathbb{R}^{2\times 3}$ definida como:

$$\boldsymbol{S} = \begin{bmatrix} s_{11} & s_{12} & s_{13} \\ s_{21} & s_{22} & s_{23} \end{bmatrix}, \tag{99}$$

se guardaría en memoria en el siguiente orden:

$$[s_{11}, s_{21}, s_{12}, s_{22}, s_{13}, s_{23}]. \tag{100}$$

Siguiendo esta lógica, y aprovechando la gestión de memoria en Fortran, si la matriz $\boldsymbol{H}$ se lee y guarda en la correspondiente subrutina de lectura de datos en una sección del bloque de memoria $\boldsymbol{R}$, esa sección del bloque de memoria puede configurarse directamente como el vector $\boldsymbol{h}$ en la subrutina siguiente. Esta propiedad de Fortran permite utilizar la matriz $\boldsymbol{H}$ como el vector $\boldsymbol{h}$ sin necesidad de transformaciones adicionales.

Visto cómo se procede para el cálculo de la matriz $\boldsymbol{A}$ y el vector $\boldsymbol{h}$, el sistema de ecuaciones a resolver queda totalmente definido. A partir de aquí, el algoritmo funciona de la siguiente forma:

1. **Suposición Inicial:** Se asume que todos los elementos del área potencial de contacto forman parte del área de contacto.
2. **Se resuelve el sistema matricial $\boldsymbol{h} = \boldsymbol{A}\boldsymbol{p_d}$ (60).**
3. **Análisis de resultados:** Dado que se ha considerado el sistema matricial (60), si se identifican elementos con tensiones nulas o negativas (elementos pegados) se concluye que estos elementos no forman parte del área de contacto real y, por lo tanto, se eliminan de esta.
4. **Iteración:** Si se encuentran elementos con tensiones nulas o negativas en el paso anterior, se repite el paso 2 con una nueva área de contacto. Para determinar esta nueva área de contacto y realizar la iteración, se siguen los siguientes pasos:
    1. **Generación del subproblema de contacto:** Se genera una matriz $\boldsymbol{T}$, de dimensiones $N_e \times j_{ec}$ (siendo j_{ec} el número de elementos en contacto), calculando sus entradas mediante

$$t_{ij} = \begin{cases} = 1, & p_{d_i} \geq 1.0 \cdot 10^{-9} \\ = 0, & p_{d_i} < 1.0 \cdot 10^{-9} \end{cases}, \tag{101}$$

    donde j es un índice que varía desde 1 hasta el número total de elementos que satisfacen la condición (j_{ec}). Mediante la tolerancia impuesta, se determina qué elementos forman la nueva área de contacto ($t_{ij} = 1$).

    Una vez generada la matriz $\boldsymbol{T}$, se produce un nuevo sistema matricial,

$$\boldsymbol{h_2} = \boldsymbol{A_2}\boldsymbol{x}, \tag{102}$$

    donde $\boldsymbol{A_2} = \boldsymbol{T}^T\boldsymbol{A}\boldsymbol{T}$ es la matriz de coeficientes de influencia del área de contacto actualizada, $\boldsymbol{h_2} = \boldsymbol{T}^T\boldsymbol{r}\boldsymbol{h}\boldsymbol{t}$ el vector que contiene la distancia entre las superficies indeformadas correspondientes a la nueva área de contacto y $\boldsymbol{x}$ el vector de tensiones normales asociadas a sus elementos[10].

    Nótese que el uso de la matriz $\boldsymbol{T}$ genera un subproblema normal de menores dimensiones, lo que, por sí mismo, acelera los cálculos en cierta medida. Sin embargo, el cálculo de $\boldsymbol{A_2}$ mediante el producto $\boldsymbol{T}^T\boldsymbol{A}\boldsymbol{T}$ da lugar a un conjunto de cuatro bucles

[10] Para el sistema matricial propuesto, $\boldsymbol{A_2}$ será de dimensión $j_{ec} \times j_{ec}$ y $\boldsymbol{h_2}$ de dimensión $j_{ec} \times 1$, dada la dimensión de $\boldsymbol{T}$. Por ello, $\boldsymbol{x}$ también será un vector de dimensión $j_{ec} \times 1$, que deberá extenderse a la dimensión del problema global, premultiplicando por $\boldsymbol{T}$. Nótese que $\boldsymbol{T}^T$ representa la matriz traspuesta de $\boldsymbol{T}$.

anidados en el código, lo que ralentiza el algoritmo y lo convierte en un cuello de botella.

Para solucionar este problema, se puede aprovechar la naturaleza de la matriz $\boldsymbol{T}$. Esta matriz es dispersa, es decir, una matriz en la que la mayoría de sus elementos son nulos. Un pequeño ejemplo podría ser el siguiente:

$$\boldsymbol{T} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Esta propiedad de la matriz $\boldsymbol{T}$ se obtiene al identificar cuáles entradas del vector $\boldsymbol{p_d}$ son mayores o iguales que la tolerancia, tal y como se describe en la expresión (101). Dado que la operación $\boldsymbol{T^T A T}$ es asimilable a formar $\boldsymbol{A_2}$ mediante las entradas de la matriz $\boldsymbol{A}$ asociadas a los casos en los que $\boldsymbol{T^T A T}$ no resulta en un producto nulo, se puede generar el vector $\boldsymbol{E_C}$ (de dimensión $j_{ec} \times 1$), cuyos elementos e_{c_j} se definan como:

$$\boldsymbol{E_C} = \{i \in \{1,2,\dots,N_e\} \mid p_i \geq 1.0 \cdot 10^{-9}\}, \tag{103}$$

donde j es un índice que varía desde 1 hasta el número total de elementos que satisfacen la condición, es decir, aquellos que están en el área de contacto (j_{ec}).

A partir de lo anterior, es posible construir los elementos de $\boldsymbol{A_2}$ mediante los de $\boldsymbol{A}$ como:

$$\boldsymbol{A_2}(i,j) = \boldsymbol{A}\big(\boldsymbol{E_C}(i), \boldsymbol{E_C}(j)\big), \tag{104}$$

de tal forma que $a_{2_{ij}} = a_{e_{c_i}, e_{c_j}}$.

Esta forma de operar solo requiere accesos a memoria, mientras que realizar el producto $\boldsymbol{T^T A T}$ implicaría tanto accesos a memoria como la ejecución de productos de tres variables distintas en bucles internos. Esto explica por qué esta metodología reduce el tiempo de ejecución total del problema, evitando que la formación de la matriz $\boldsymbol{A_2}$ se convierta en un cuello de botella.

2. **Se realiza el paso 2 resolviendo (102).**
3. **Extensión de $\boldsymbol{x}$:** Se calcula el vector $\boldsymbol{p_d}$ de tensiones normales en el contacto extendiendo el vector $\boldsymbol{x}$ al tamaño del problema:

$$\boldsymbol{p_d} = \boldsymbol{T x}. \tag{105}$$

4. **Se procede al paso 3 evaluando $\boldsymbol{p_d}$.**

5. **Convergencia:** El proceso iterativo continúa hasta que no se detecten elementos con tensiones negativas, momento en el cual se considera que se ha encontrado la solución al problema normal de contacto.

3.2.4.3.2. Variantes implementadas

Tradicionalmente, el método NORM se ha implementado junto con la eliminación gaussiana (GE) para resolver sistemas matriciales. Este enfoque permite obtener una solución directa y explícita del sistema de ecuaciones generado por el modelo de contacto, siendo especialmente adecuado para problemas en los que el tamaño del sistema no es excesivamente grande y la demanda computacional es razonable.

Sin embargo, a medida que el tamaño del sistema crece, la eliminación gaussiana puede volverse computacionalmente costosa, tanto en términos de tiempo como de memoria. En estos casos, se hace necesario explorar métodos alternativos que puedan reducir el tiempo computacional manteniendo la precisión de los resultados. Es en este contexto que se ha implementado el método de Gradiente Conjugado (CG).

El CG es un método iterativo que es particularmente eficiente para resolver sistemas de ecuaciones lineales con matrices simétricas y definidas positivas, que son comunes en los problemas de contacto en la mecánica. Su principal ventaja radica en que, al ser un método iterativo, no requiere almacenar matrices de gran tamaño en su forma completa, lo que reduce significativamente la carga de memoria. Además, el gradiente conjugado tiende a converger más rápidamente que la eliminación gaussiana en muchos casos, especialmente cuando se aplica a sistemas de gran escala.

Para este trabajo, se han implementado y comparado dos métodos distintos para la resolución de los sistemas de ecuaciones asociados con el método NORM:

- **NORM**: La resolución del sistema de ecuaciones $\boldsymbol{Ap_d = h}$ se lleva a cabo utilizando la eliminación gaussiana.
- **NORM+CG:** la resolución del sistema de ecuaciones se realiza mediante el método de gradiente conjugado.

3.2.4.4. NORM + E/P

Según Giner Navarro et al. (2023), tal y como se ha expuesto en 2.3.3, se puede aproximar la distribución de tensiones normales mediante una base elíptica o polinómica en la dirección $\boldsymbol{X_1}$, dependiendo de ciertas condiciones. La distribución elíptica es adecuada para los casos donde no existe ángulo de guiñada (*yaw o* ψ), mientras que la resolución polinómica es más favorable en el caso contrario.

Basándose en las ideas de estos autores y utilizando el método NORM explicado anteriormente, es posible generar el algoritmo de resolución propuesto en este apartado. Debe entenderse que esta implementación aplica a la generación de un nuevo sistema matricial a resolver. Una vez especificado este sistema, el vector $\boldsymbol{p_d}$ se obtiene según la aproximación definida y a partir de la solución del sistema matricial generado. El método NORM se aplica en todas las demás operaciones del algoritmo.

3.2.4.4.1. Implementación de la distribución elíptica de tensiones

Siguiendo la definición de las tensiones según (61) y extendiéndola a todos los elementos del área de contacto considerada, el vector $\boldsymbol{p_d}$ puede obtenerse como:

$$\boldsymbol{p_d} \approx \boldsymbol{Q\lambda}, \tag{106}$$

donde $\boldsymbol{\lambda}$ es el vector de coeficientes λ_{gs} y $\boldsymbol{Q}$ una matriz que depende de las coordenadas de la malla x_{es} y las longitudes de las tiras L_s.

A modo de ejemplo, un área de contacto como la de la Figura 21 generaría una matriz $\boldsymbol{Q}$ y un vector $\boldsymbol{\lambda}$ con la siguiente estructura:

$$\boldsymbol{Q} = \begin{bmatrix} 1 & \sqrt{1-\left(\frac{-2a}{3a}\right)^2} & 0 & 0 \\ 1 & \sqrt{1-\left(\frac{0}{3a}\right)^2} & 0 & 0 \\ 1 & \sqrt{1-\left(\frac{2a}{3a}\right)^2} & 0 & 0 \\ 0 & 0 & 1 & \sqrt{1-\left(\frac{-4a}{5a}\right)^2} \\ 0 & 0 & 1 & \sqrt{1-\left(\frac{-2a}{5a}\right)^2} \\ 0 & 0 & 1 & \sqrt{1-\left(\frac{0}{5a}\right)^2} \\ 0 & 0 & 1 & \sqrt{1-\left(\frac{2a}{5a}\right)^2} \\ 0 & 0 & 1 & \sqrt{1-\left(\frac{4a}{5a}\right)^2} \end{bmatrix}, \qquad \boldsymbol{\lambda} = \begin{Bmatrix} \lambda_{01} \\ \lambda_{11} \\ \lambda_{02} \\ \lambda_{12} \end{Bmatrix}. \tag{107}$$

La obtención de las entradas de $\boldsymbol{Q}$ se realiza aprovechando la naturaleza simétrica del vector $\boldsymbol{p_d}$. Sin embargo, antes de proceder con estos cálculos, es necesario determinar ciertos parámetros clave.

Durante el proceso de identificación de los elementos de la nueva área de contacto mediante la ecuación (101), es posible calcular simultáneamente L_s (el semieje asociado a la tira s) y el número de tiras $strp$. Para calcular las variables L_s y $strp$, se emplea una variable lógica llamada chk, cuyo valor será 0 si el elemento no pertenece al área de contacto y 1 si pertenece a ella. El algoritmo opera, elemento a elemento, de la siguiente forma:

1. Si el elemento pertenece al área de contacto ($p_{d_i} \geq 1.0 \cdot 10^{-9}$) y el anterior no, se actualiza el número de tiras. Esto se expresa como:

$$strp = strp + 1, \qquad si\ chk = 0\ y\ strp \leq N_2, \tag{108}$$

y se actualiza $chk = 1$ al encontrarse un elemento perteneciente a la nueva área de contacto.

2. $\boldsymbol{L_s}$, el vector que contiene el tamaño de los semiejes de las tiras existentes (que es igual al número total de elementos en cada tira), se inicializa para la tira actual:

$$\boldsymbol{L_s}(strp) = 1. \tag{109}$$

3. Para el siguiente elemento, se evaluará si chk es 0 o 1. En caso de ser 1, se entiende que el elemento anterior estaba en el área de contacto y pertenece a la tira, con lo que se actualiza el número de elementos de esta:

$$\boldsymbol{L_s}(strp) = \boldsymbol{L_s}(strp) + 1, \qquad si\ chk = 1\ y\ strp \leq N_2\ . \tag{110}$$

Llegado a este punto, el algoritmo sigue iterando (pasos 1 a 3) elemento a elemento, actualizando el número de elementos en la tira correspondiente, hasta que se produce una de las siguientes tres situaciones, momento en el que se establece $chk = 0$ de nuevo:

- El elemento no pertenece al área de contacto.
- El número de tiras supera el número de elementos en la dirección lateral: $strp \geq N_2$
- El número de elementos de la tira es mayor al número de elementos en la dirección de rodadura: $\boldsymbol{L_s}(strp, 1) \geq N_1$.

Con lo anterior es posible determinar el valor de L_s para cada tira y el número total de tiras $strp$, construyendo así el vector $\boldsymbol{L_s}$. Es importante destacar que, en la primera resolución del sistema de ecuaciones del problema, $L_s = N_1$ para un total de N_2 tiras, según el paso 1 del método NORM.

Una vez determinados estos parámetros, se procede a iterar sobre el número de tiras y el tamaño del semieje L_s de cada una, calculando las entradas no nulas de la matriz $\boldsymbol{Q}$ de la siguiente forma:

$$\begin{aligned} \boldsymbol{Q}(i_1 + i_3, 2j - 1) &= 1\ , \\ \boldsymbol{Q}(i_1 + i_3, 2j) &= \sqrt{1 - \left(\frac{x_{es}}{i_2}\right)^2}, \end{aligned} \tag{111}$$

donde:

$$x_{es} = -\left(2\left(\frac{i_2}{2} - i_1 + 1\right) - 1\right). \tag{112}$$

En esta formulación:

- i_1 es el índice del elemento de la tira para el que se están calculando las entradas de $\boldsymbol{Q}$.
- $i_2 = \boldsymbol{L_s}(j)$ define el limite al que puede llegar i_1 y corresponde al tamaño de la tira j que se está procesando.
- i_3 almacena el número de elementos que ya se han procesado, ayudando a determinar la fila de $\boldsymbol{Q}$ a partir de la que se almacenan los valores para la tira actual. i_3 se actualiza como $i_3 = i_3 + \boldsymbol{L_s}(j)$.
- j es el índice que indica la tira cuya distribución va a calcularse. Toma valores desde 1 hasta $strp$.

A partir de la utilización de la matriz $\boldsymbol{Q}$ en el método propuesto, se logra una reducción significativa en el número de incógnitas necesarias para resolver el problema de contacto normal rueda-carril. En lugar de trabajar con todas las incógnitas asociadas a cada uno de los elementos discretizados en la malla de contacto, que en el algoritmo NORM equivaldría a N_e incógnitas, el empleo de la matriz $\boldsymbol{Q}$ permite reducir este número a aproximadamente $2\sqrt{N_e}$. Esta disminución se basa en la capacidad de la matriz $\boldsymbol{Q}$ para representar la distribución de las tracciones normales a través de una base adecuada, lo que resulta en una representación más compacta y manejable del problema. Esta reducción en la cantidad de incógnitas no solo mejora la eficiencia computacional, sino que también facilita la obtención de soluciones precisas con un menor esfuerzo computacional, lo cual es crucial para la implementación del método en simulaciones dinámicas de vehículos ferroviarios.

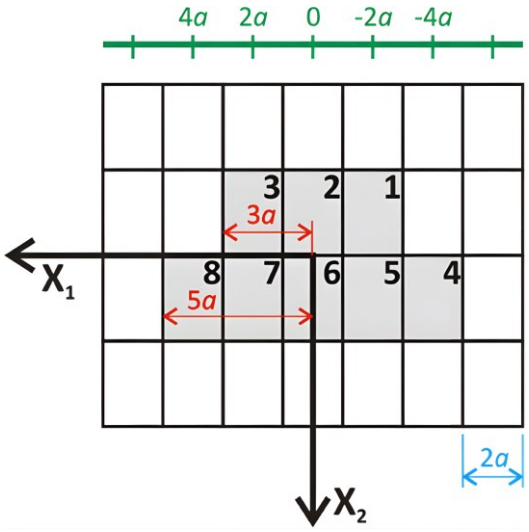

Figura 21. Ejemplo de área de contacto con la que determinar $\boldsymbol{Q}$ y $\boldsymbol{B}$ (Giner Navarro, Gómez Bosch, Alonso, & Baeza, 2023).

Prestando atención a la ecuación (60) y aplicando la expresión (106), al premultiplicar por $\boldsymbol{Q}^T$ se obtiene:

$$\boldsymbol{Q}^T\boldsymbol{h} = \boldsymbol{Q}^T\boldsymbol{A}\boldsymbol{Q}\boldsymbol{\lambda}. \tag{113}$$

Ahora, utilizando las ecuaciones (106) y (113) se obtiene:

$$\boldsymbol{p_d} = \boldsymbol{Q}(\boldsymbol{Q}^T\boldsymbol{A}\boldsymbol{Q})^{-1}\,(\boldsymbol{Q}^T\boldsymbol{h}). \tag{114}$$

Nótese que, dado que λ_{0s} y λ_{1s} son las variables incógnita por tira, la ecuación (113) corresponde a un sistema lineal de dimensión $2N_s$, donde N_s es el número de tiras. Esto da lugar a un problema con dimensiones, en general, mucho menores que las del problema original (60), que cuenta con N_e elementos.

3.2.4.4.2. Implementación de la distribución polinómica de tensiones

La distribución polinómica en el método propuesto se introduce para mejorar la capacidad de modelar la distribución de las tracciones normales en la zona de contacto entre la rueda y el carril, particularmente en situaciones donde existe ángulo de guiñada (yaw) del eje montado, tal y como se ha explicado en el contexto teórico.

Para abordar este problema se propone utilizar una distribución polinómica en lugar de una elíptica, aproximando las tensiones mediante un polinomio de grado g definido según la expresión (65), que permite capturar la variación de la tracción a lo largo de cada tira de la malla en la dirección de rodadura. Con esta definición de las tensiones, el vector $\boldsymbol{p_d}$ se puede configurar como:

$$\boldsymbol{p_d} \approx \boldsymbol{B}\boldsymbol{\rho}, \tag{115}$$

donde $\boldsymbol{\rho}$ es el vector de coeficientes ρ_{js} y $\boldsymbol{B}$ una matriz que depende de las coordenadas de la malla x_{es} y del grado del polinomio g. Como en el caso elíptico, dado que el área de contacto es conocida previamente al cálculo de las presiones, x_{es} y $\boldsymbol{B}$ son conocidas. A modo de ejemplo, como con anterioridad, la Figura 21 ilustra un área de contacto que generaría, para $g = 2$, una matriz $\boldsymbol{B}$ y un vector $\boldsymbol{\rho}$ con la siguiente forma:

$$\boldsymbol{B} = \begin{bmatrix} 1 & -2a & (-2a)^2 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2a & (2a)^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -4a & (-4a)^2 \\ 0 & 0 & 0 & 1 & -2a & (-2a)^2 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2a & (2a)^2 \\ 0 & 0 & 0 & 1 & 4a & (4a)^2 \end{bmatrix}, \qquad \boldsymbol{\rho} = \begin{Bmatrix} \rho_{01} \\ \rho_{11} \\ \rho_{21} \\ \rho_{02} \\ \rho_{12} \\ \rho_{22} \end{Bmatrix}. \tag{116}$$

La obtención de las entradas de $\boldsymbol{B}$ se realiza de forma similar a las de $\boldsymbol{Q}$. Sin embargo, en este caso el proceso resulta más simple.

En primer lugar, L_s (el semieje asociado a la tira s) y el número de tiras $strp$ se calculan de la misma forma:

1. Si el elemento pertenece al área de contacto y el anterior no, se actualiza el número de tiras. Esto se expresa como:

$$strp = strp + 1, \qquad si\ chk = 0\ y\ strp \leq N_2, \tag{117}$$

y se actualiza $chk = 1$ al encontrarse un elemento perteneciente a la nueva área de contacto.

2. $\boldsymbol{L_s}$, el vector que contiene el tamaño de los semiejes de las tiras existentes (que es igual al número total de elementos en cada tira), se inicializa para la tira actual:

$$\boldsymbol{L_s}(strp) = 1. \tag{118}$$

3. Para el siguiente elemento, se evaluará si chk es 0 o 1. En caso de ser 1, se entiende que el elemento anterior estaba en el área de contacto y pertenece a la tira, con lo que se actualiza el número de elementos de esta:

$$\boldsymbol{L_s}(strp) = \boldsymbol{L_s}(strp) + 1, \qquad si\ chk = 1\ y\ strp \leq N_2\ . \tag{119}$$

Llegado a este punto, el algoritmo sigue iterando (pasos 1 a 3) elemento a elemento, actualizando el número de elementos en la tira correspondiente, hasta que se produce una de las siguientes tres situaciones, momento en el que se establece $chk = 0$ de nuevo:

- El elemento no pertenece al área de contacto.
- El número de tiras supera el número de elementos en la dirección lateral: $strp \geq N_2$
- El número de elementos de la tira es mayor al número de elementos en la dirección de rodadura: $\boldsymbol{L_s}(strp, 1) \geq N_1$.

Con lo anterior es posible determinar el valor de L_s para cada tira y el número total de tiras $strp$, construyendo así el vector $\boldsymbol{L_s}$. Es importante recordar que, en la primera resolución del sistema de ecuaciones del problema, $L_s = N_1$ para un total de N_2 tiras, según el paso 1 del método NORM.

Una vez determinados estos parámetros, se procede a iterar sobre el número de tiras y el tamaño del semieje de cada una, calculando las entradas no nulas de la matriz $\boldsymbol{B}$ de la siguiente forma:

$$\boldsymbol{B}\big(i_1 + i_3, (g+1)\cdot j - (g-m)\big) = (x_{es}\cdot a)^m\ , \tag{120}$$

donde:

$$x_{es} = -\left(2\left(\frac{i_2}{2} - i_1 + 1\right) - 1\right). \tag{121}$$

En esta formulación:

- i_1 es el índice del elemento de la tira para el que se están calculando las entradas de $\boldsymbol{Q}$.
- $i_2 = \boldsymbol{L_s}(j)$ define el limite al que puede llegar i_1 y corresponde al tamaño de la tira que se está procesando.
- i_3 almacena el número de elementos que ya se han procesado, ayudando a determinar la fila de $\boldsymbol{Q}$ a partir de la que se almacenan los valores para la tira actual. i_3 se actualiza como $i_3 = i_3 + \boldsymbol{L_s}(j)$.
- g es el grado del polinomio.
- m es un índice que va desde 0 hasta el grado g.
- j indica la tira que se está evaluando, hasta el valor máximo $strp$.

De nuevo, teniendo en cuenta ahora las ecuaciones (60) y (115), y premultiplicando por $\boldsymbol{B}^T$ se puede calcular

$$\boldsymbol{B}^T\boldsymbol{h} = \boldsymbol{B}^T\boldsymbol{A}\boldsymbol{B}\boldsymbol{\rho}, \tag{122}$$

y utilizando las ecuaciones (106) y (122) se obtiene:

$$\boldsymbol{p_d} = \boldsymbol{B}(\boldsymbol{B}^T\boldsymbol{A}\boldsymbol{B})^{-1}\,(\boldsymbol{B}^T\boldsymbol{h}). \tag{123}$$

La reducción del número de incógnitas es un factor crucial en este enfoque. Dependiendo del grado polinómico que se utilice, la dimensión del problema puede ser mayor o menor en comparación con la aproximación elíptica. No obstante, el uso de la matriz $\boldsymbol{B}$ permite nuevamente reducir las dimensiones del problema por debajo de las del problema original. En concreto, el problema final es de dimensión $(g+1)N_s$, siendo N_s el número de tiras en las que se divide el área de contacto en la dirección $\boldsymbol{X_2}$.

3.2.4.4.3. Particularidades

En la implementación del método, se opta por utilizar las mismas variables tanto para $\boldsymbol{Q}$ y $\boldsymbol{B}$ como para los vectores $\boldsymbol{\lambda}$ y $\boldsymbol{\rho}$. Esto implica considerar el vector $\boldsymbol{p_d}$ de forma genérica como:

$$\boldsymbol{p_d} \approx \boldsymbol{Q}\boldsymbol{\lambda}. \tag{124}$$

Para generar la matriz $\boldsymbol{Q}$, se invoca la subrutina 'CALC_Q_E', donde la 'E' indica que se calculará la matriz correspondiente a la distribución elíptica de tensiones. En el caso de querer generar la matriz $\boldsymbol{B}$, se utiliza la subrutina 'CALC_Q_P', donde la 'P' denota que se trata del caso de distribución polinómica; en este último caso, es necesario proporcionar adicionalmente el grado polinómico deseado como argumento de entrada. Una vez calculadas las matrices, el algoritmo está parametriza para trabajar con el número de incógnitas inherente a la matriz generada.

Por otro lado, al igual que en el algoritmo NORM, se hace uso de la matriz $\boldsymbol{T}$ y el vector $\boldsymbol{E_C}$. Esto implica que, en la iteración inicial, donde se asume que el área de contacto es igual al área potencial de contacto, se resolverán las ecuaciones (113) y (122) para obtener $\boldsymbol{\lambda}$ y $\boldsymbol{\rho}$, respectivamente. Posteriormente, para calcular el vector $\boldsymbol{p_d}$, es suficiente con operar de acuerdo con las ecuaciones (106) o (115).

Sin embargo, en iteraciones posteriores, una vez actualizada el área de contacto y calculados $\boldsymbol{T}$ y $\boldsymbol{E_C}$, el problema se basa en resolver el sistema (102). Si a ese sistema matricial, por ejemplo, se le aplica la definición de $\boldsymbol{p_d}$ según (106), la expresión (113) se actualiza a:

$$\boldsymbol{Q}^T\boldsymbol{h_2} = \boldsymbol{Q}^T\boldsymbol{A_2}\boldsymbol{Q}\boldsymbol{\lambda}, \tag{125}$$

donde $\boldsymbol{h_2}$ y $\boldsymbol{A_2}$ se calculan según lo indicado en el paso 4 del algoritmo NORM y de acuerdo con la expresión (104), respectivamente.

Una vez resuelta (125) para obtener $\boldsymbol{\lambda}$, el vector $\boldsymbol{p_d}$ se puede calcular como:

$$\boldsymbol{p_d} = \boldsymbol{TQ\lambda}\,. \tag{126}$$

El procedimiento anterior se aplica de forma análoga para la aproximación polinómica de las tensiones (115).

Es importante destacar que, para calcular los términos $\boldsymbol{Q^T A Q}$ y $\boldsymbol{Q^T A_2 Q}$, se enfrenta nuevamente el problema de los cuatro bucles anidados que obligaban a construir el vector $\boldsymbol{Ec}$. Sin embargo, en este caso, dado que el problema tiene dimensiones mucho menores y que $\boldsymbol{Q}$ es una matriz con muchas entradas nulas, se acelera el proceso utilizando una matriz temporal $\boldsymbol{Temp = AQ}$, de modo que $\boldsymbol{Q^T A Q}$ se puede calcular como $\boldsymbol{Q^T Temp}$. Además, se optimiza el conjunto de bucles omitiendo los productos que se sabe de antemano serán cero.

3.2.4.4.4. Variantes implementadas

Tras la implementación del método NORM+E/P, se desarrollaron dos variantes adicionales para resolver los sistemas de ecuaciones resultantes de manera más eficiente. Estas variantes aprovechan tanto el método NORM como la aproximación de distribuciones para optimizar el proceso de resolución:

- **NORM+E/P:** En esta variante, la resolución del sistema matricial $\boldsymbol{Ap_d = h}$ se realiza utilizando la eliminación gaussiana (GE). Debido a la reducción en las dimensiones del problema obtenida con la aproximación de las distribuciones, la eliminación gaussiana puede resultar una opción favorable, permitiendo una solución eficiente y manejable en términos de recursos computacionales.
- **NORM+E/P+CG:** Esta variante emplea el método de gradiente conjugado para resolver el sistema de ecuaciones.

Considerar ambas variantes es esencial, ya que permiten investigar si existen mejoras significativas en el tiempo de resolución o en el uso de recursos, lo que puede ofrecer valiosas conclusiones sobre cuál de los métodos es más efectivo emplear bajo esta metodología de resolución.

3.2.4.5. BCCG(K)

Vollebregt (2013) proporciona un código del algoritmo *Enhanced BCCG(K)* desarrollado en MATLAB. Este algoritmo implementa lo descrito en el apartado 2.3.4, adaptado al contexto de programación. Para su implementación, se realizó una traducción minuciosa de MATLAB a Fortran, respetando cuidadosamente la estructura del código original. Durante este proceso de transformación y adaptación, surgieron varios aspectos importantes:

- La parte inicial programada para NORM es necesaria aquí también, ya que se debe resolver el mismo sistema de ecuaciones (60). Esto implica que el cálculo de la matriz $\boldsymbol{A}$ y la obtención del vector $\boldsymbol{h}$, tal como se definen en la sección 3.2.4.3.1 y se calculan específicamente con las ecuaciones (4) y (95)-(98), se mantienen sin cambios.

- El código en MATLAB permite utilizar el método *NORM+CG*, el *Plain BCCG* o el *Enhanced BCCG(K)* poniendo a uno las variables lógicas correspondientes. Como en este trabajo se emplea el último, las variables lógicas se mantendrán en cero.
- El Método BCCG(K) utiliza la iteración externa para manejar las restricciones y modificar las variables del problema. Esto permite abordar la resolución del problema normal partiendo del sistema global $\boldsymbol{Ap_d = h}$, sin necesidad de usar la matriz $\boldsymbol{T}$, ni el vector $\boldsymbol{E_c}$. Esto es posible porque la iteración externa distingue entre los elementos que pertenecerán o no al área de contacto, lo que hace innecesario definir y utilizar la matriz $\boldsymbol{A_2}$. Como resultado, se reduce el uso de memoria en comparación con el método NORM de 3 a 4 veces.
- De lo anterior se deduce también que, en comparación con el método NORM, no son necesarias las iteraciones en las que se discernían los puntos fuera del área de contacto, las cuales requerían la eliminación de Gauss o el Método del Gradiente Conjugado para resolver los subproblemas resultantes. Estas operaciones son realizadas directamente por el BCCG(K), lo que alivia considerablemente la carga de trabajo, permitiendo una única llamada al método de resolución del problema.
- La ejecución del código contempla la lectura de unas pocas variables. Sin embargo, el algoritmo trabaja con varios vectores adicionales, cuya memoria conviene sea previamente reservada en el bloque $\boldsymbol{R}$. Así, los argumentos de entrada resultan ser los siguientes:
    - K: Número de iteraciones internas tras las cuales se ejecuta la iteración externa para comprobar las restricciones y modificar el conjunto de variables. Vollebregt (2013) sugiere que $K = 1 - 5$ son valores óptimos para el cálculo. En este trabajo se utiliza $K = 2$, por lo que este algoritmo se denominará BCCG (2) en adelante. En el código se cambia la nomenclatura para esta variable, siendo $K = K_{inn}$.
    - $\boldsymbol{A}$, $\boldsymbol{p_d}$ y $\boldsymbol{h}$: Matriz de coeficientes de influencia, vector de incógnitas y vector de distancias indeformadas de los elementos de las superficies en contacto, respectivamente. En la subrutina correspondiente, $\boldsymbol{p_d}$ se utiliza bajo la nomenclatura $\boldsymbol{x}$ y el vector $\boldsymbol{h}$ bajo $\boldsymbol{b}$.
    - $\boldsymbol{g}, \boldsymbol{r}, \boldsymbol{v}$ y $\boldsymbol{q}$: Vectores destinados a albergar el gradiente, el residuo, la dirección de búsqueda y el producto matriz-vector $\boldsymbol{Av}$, respectivamente.
    - $\boldsymbol{x_{prev}}$ y $\boldsymbol{r_{prev}}$: Vectores que almacenan los valores de $\boldsymbol{p_d}$ y el residuo en la iteración previa a la actual, respectivamente.
    - tol: Tolerancia con la que se verifica la convergencia.
    - $maxit$: Número máximo de iteraciones permitido.
    - $\boldsymbol{upd}$: Vector $\mathbb{R}^{maxit}$ que almacena el parámetro de convergencia del método en cada iteración.
    - $\boldsymbol{Ix_{bnd}}$: Vector que almacena los índices de los elementos restringidos (o no pertenecientes al área de contacto).
    - $\boldsymbol{msk_{bnd}}$: Vector lógico o máscara que almacena unos en las posiciones correspondientes a los elementos de $\boldsymbol{p_d}$ que no pertenecen al área de contacto, y ceros en las demás. Se utiliza para definir posteriormente $\boldsymbol{Ix_{bnd}}$.
    - $\boldsymbol{Ix_{prj}}$: Vector que almacena los índices correspondientes a aquellos elementos de $\boldsymbol{p_d}$ cuyo valor es menor a un determinado umbral (llamado $small$ en el código, del orden de 10^{-14}) y que deben ser proyectados mediante $\boldsymbol{p_d(Ix_{prj}) = 0}$.

- $\boldsymbol{Ix_{rel}}$: Vector que almacena los índices de las variables que deben ser cambiadas al conjunto libre (es decir, variables que deben considerarse pertenecientes al área de contacto). Es utilizado durante la iteración externa para ajustar la máscara $\boldsymbol{msk_{bnd}}$ y actualizar las restricciones.
- N_e: la dimensión del problema.

Una vez adaptado el código, es posible discernir cómo se han implementado cada uno de los pasos del algoritmo descritos en el contexto teórico. Para facilitar una mejor comprensión de la implementación, se detallan a continuación algunos aspectos clave, describiendo los pasos y su adaptación particular al problema normal de contacto. Cabe mencionar que se mantiene la misma nomenclatura utilizada en los otros métodos, y que el conjunto de todas las variables p_{d_i} se denota, siguiendo la notación de Vollebregt (2013), por $\mathcal{N} = \{1, \dots, N_e\}$ [11]:

1. Dada la matriz $\boldsymbol{A} \in \mathbb{R}^{N_e \times N_e}$, el vector que contiene las distancias entre las superficies indeformadas en contacto $\boldsymbol{h} \in \mathbb{R}^{N_e}$ y una estimación inicial $\widetilde{\boldsymbol{p_d}}^0 \in \mathbb{R}^{N_e}$, el algoritmo empieza con la construcción de un punto factible proyectando la estimación inicial elemento a elemento:

$$\boldsymbol{p_d} = \max\left(0, \widetilde{\boldsymbol{p_d}}^0\right). \tag{127}$$

2. Posteriormente, se calculan los vectores $\boldsymbol{g}^0$, $\boldsymbol{msk_{bnd}}$, $\boldsymbol{Ix_{bnd}}$ y el número de elementos libres (es decir, pertenecientes al área de contacto) n_{free} como:

$$\boldsymbol{g}^0 = \boldsymbol{A}\boldsymbol{p_d} - \boldsymbol{h}, \tag{128}$$

$$msk_{bnd_i} = \begin{cases} 1, & si\ p_{d_i} \leq 0\ y\ g_i \geq 0 \\ 0, & en\ caso\ contrario \end{cases}, \qquad \forall i \in \mathcal{N} \tag{129}$$

$$\boldsymbol{Ix_{bnd}} = \left\{ i \in \mathcal{N} \,|\, msk_{bnd_i} = 1 \right\}, \tag{130}$$

$$n_{free} = N_e - n_{bnd}, \tag{131}$$

donde n_{bnd} es el número de elementos o variables con restricciones (el número de elementos en $\boldsymbol{Ix_{bnd}}$). Nótese que las ecuaciones (129) y (130) son análogas a determinar el conjunto de variables con restricciones $\mathcal{B}$ de $\mathcal{N}$ o determinar los elementos externos al área de contacto.

3. Se inicia el proceso iterativo con $k = 1,2,3 \dots$ (el número de iteraciones global), $k_{inn} = 1,2,3 \dots$ (el número de iteraciones internas) y utilizando $\boldsymbol{p_d}^{k-1}$ y $\boldsymbol{r}^{k-2}$ dados por la iteración anterior[12]:
   a. Se almacena el valor anterior de $\boldsymbol{p_d}$:

[11] Recuérdese que N_e representa al número de elementos total de la malla del área potencial de contacto.

[12] Por notación, aquí se utiliza k y k_{inn} para denotar lo que en el código corresponde a it e it_{inn}, respectivamente.

$$\boldsymbol{x_{prev}} = \boldsymbol{p_d^{k-1}}. \tag{132}$$

b. Si $k > 1$:

$$\boldsymbol{r_{prev}} = \boldsymbol{r}^{k-2}, \tag{133}$$

$$\delta_{prev} = \delta. \tag{134}$$

c. En caso contrario, se pasa directamente al paso 3.

4. Se define el residuo $\boldsymbol{r}^{k-1}$ y el producto escalar δ:
   a. Se calcula el residuo para el subproblema correspondiente mediante:

$$\boldsymbol{r}^{k-1} = -\boldsymbol{g}^{k-1}. \tag{135}$$

   b. Se anulan los elementos con restricciones de este:

$$\boldsymbol{r}_j^{k-1} = 0, \qquad \forall j \in \boldsymbol{Ix_{bnd}}. \tag{136}$$

   c. Se calcula δ como:

$$\delta = \boldsymbol{r}^{k-1^T} \cdot \boldsymbol{r}^{k-1}. \tag{137}$$

5. Se define una *dirección de búsqueda* $\boldsymbol{v}^k$, nula para las componentes de las variables restringidas:
   a. Si $k \leq 1$, se reinicia el Gradiente Conjugado tomando la dirección correspondiente a la del algoritmo *steepest descent*:

$$\boldsymbol{v}^k = \boldsymbol{r}^{k-1}. \tag{138}$$

   b. Por el contrario, si $k > 1$, se usan las fórmulas de Polak-Ribière para definir la nueva dirección:

$$\beta_{PR}^k = \left(\delta - \boldsymbol{r}^{k-1^T} \cdot \boldsymbol{r}^{k-2}\right) / \delta_{prev}, \tag{139}$$

$$\boldsymbol{v}^k = \boldsymbol{r}^{k-1} + \max\left(0, \beta_{PR}^k\right) \boldsymbol{v}^{k-1}. \tag{140}$$

   c. Se establece posteriormente:

$$\boldsymbol{v}_j^k = 0, \qquad \forall j \in \boldsymbol{Ix_{bnd}}. \tag{141}$$

6. Se computa el tamaño de paso óptimo α^k, según Polak-Ribière, sin atender a las restricciones:

$$\boldsymbol{q}^k = \boldsymbol{A}^k \boldsymbol{v}^k, \tag{142}$$

Si $k_{inn} < K$, se establece $\boldsymbol{q}_i^k = 12345.67 \ \forall i \in \boldsymbol{Ix_{bnd}}$. Independientemente de ello, se calcula posteriormente:

$$\alpha_{PR}^k = \left(\boldsymbol{r}^{k-1}, \boldsymbol{v}^k\right) / \left(\boldsymbol{v}^k, \boldsymbol{q}^k\right), \tag{143}$$

$$\widetilde{\boldsymbol{p_d}}^k = \boldsymbol{p_d}^{k-1} + \alpha_{PR}^k \boldsymbol{v}^k. \tag{144}$$

Al vector $\widetilde{\boldsymbol{p_d}}^k$ se le llama iterado intermedio.

7. Se calculan los parámetros asociados al criterio de convergencia:

$$RMS_{xk} = \frac{\left\|\widetilde{\boldsymbol{p_d}}^k\right\|_{\boldsymbol{2}}}{\sqrt{n_{free}}}, \tag{145}$$

$$RMS_{upd} = \frac{\left\|\widetilde{\boldsymbol{p_d}}^k - \boldsymbol{x_{prev}}\right\|_{\boldsymbol{2}}}{\sqrt{n_{free}}}, \tag{146}$$

$$\boldsymbol{upd}(k) = \frac{RMS_{upd}}{RMS_{xk}}, \tag{147}$$

En este contexto, $\|\cdot\|_{\boldsymbol{2}}$ se refiere a la norma euclídea, calculada como $\|\boldsymbol{x}\|_2 = \sqrt{\sum_{i \in \mathcal{N}}^{N_e} \boldsymbol{x}_i^2}$

8. Se comprueba si ha habido cambios en las variables y debe realizarse la iteración externa:
   a. Se establece como falsa la variable lógica llamada $changed$, cuyo objetivo es indicar si ha habido cambios en las variables con restricciones.
   b. Se determina el valor de la variable lógica $outer_{it}$, responsable de indicar si se procede con la iteración externa, mediante la siguiente condición booleana:

$$outer_{it} = (k_{inn} \geq K) \vee (\boldsymbol{upd}(k) < tol) \tag{148}$$

   c. Si $outer_{it}$ es verdadero (o 1), se calcula:

$$\boldsymbol{Ix_{prj}} = \left\{ l \in \mathcal{N} \,|\, p_{d_l} < -small \right\} \tag{149}$$

      i. Si $\boldsymbol{Ix_{prj}}$ contiene algún elemento, entonces las variables de $\boldsymbol{p_d}$ correspondientes a esos índices se proyectan y se actualiza el vector lógico $\boldsymbol{msk_{bnd}}$. Esto es:

$$p_{d_l} = 0, \qquad msk_{bnd_l} = 1, \qquad \forall l \in \boldsymbol{Ix_{prj}} \tag{150}$$

y $changed$ pasa a ser verdadero (o 1).

ii. Si $\boldsymbol{Ix_{prj}}$ representa un conjunto nulo, $changed$ sigue siendo falsa y se continua con el siguiente paso

9. Se actualiza el gradiente:

$$\boldsymbol{g}^k = \begin{cases} \boldsymbol{Ap_d} - \boldsymbol{h}, & si\ (changed) \vee (outer_{it} \wedge K > 1), \\ \boldsymbol{g}^{k-1} + \alpha_{PR}^k \boldsymbol{q}^k, & de\ otro\ modo. \end{cases} \quad (151)$$

10. Se revisan las restricciones:
    a. Se define una condición booleana $check_{grad}$ para revisar si hay variables que deban ser cambiadas al conjunto libre (es decir, que deban considerarse pertenecientes al área de contacto):

$$check_{grad} = outer_{it} \wedge (\neg\, use_{normcg} \vee \neg changed) \quad (152)$$

    donde ¬ se usa para indicar la negación lógica.
    b. Si $check_{grad}$ es verdadero (o 1), se identifica el conjunto de índices $\boldsymbol{Ix_{rel}}$ asociados a las variables que deben ser movidos al conjunto libre (o área de contacto) mediante:

$$\boldsymbol{Ix_{rel}} = \{m \in \mathcal{N} | msk_{bnd_m} = 1 \wedge g_m < -small\} \quad (153)$$

    Si hay elementos en $\boldsymbol{Ix_{rel}}$, se establece $changed$ como verdadero y se actualiza la máscara $\boldsymbol{msk_{bnd}}$:

$$msk_{bnd_m} = 0, \qquad \forall m \in \boldsymbol{Ix_{rel}} \quad (154)$$

    c. En caso contrario se sigue con el algoritmo.
11. Se actualiza el vector $\boldsymbol{Ix_{bnd}}$ y se recalcula el número de variables libres n_{free} como sigue:

$$\boldsymbol{Ix_{bnd}} = \{i \in \mathcal{N} |\, msk_{bnd_i} = 1\}, \quad (155)$$

$$n_{free} = N_e - n_{bnd}, \quad (156)$$

    donde n_{bnd} es nuevamente el número de elementos en $\boldsymbol{Ix_{bnd}}$.
12. Comprobación de la convergencia:
    a. Si $outer_{it}$ es verdadero, se reinicia el contador de iteraciones internas: $k_{in} = 0$,
    b. El algoritmo se detiene si se han alcanzado el número máximo de iteraciones o si el criterio de convergencia se satisface:

$$stop = (k \geq maxit) \vee (outer_{it} \wedge \neg changed \wedge \boldsymbol{upd}(k) < tol) \quad (157)$$

    c. En caso contrario, se vuelve al paso 3.

Siguiendo todos estos pasos, el método realiza, en un número máximo de $maxit$ iteraciones, lo que el método NORM o el NORM + E/P realiza resolviendo el problema global y los subproblemas asociados a las nuevas áreas de contacto hasta llegar a la solución final.

3.2.4.6. Generalidades

Debe tenerse en cuenta que la implementación específica de cada código es la que se ha formulado y explicado anteriormente. Cabe señalar que, como se ha mencionado en varias ocasiones, la creación de la matriz $\boldsymbol{A}$ y el vector $\boldsymbol{h}$ se realiza de la misma forma en todos los códigos. Asimismo, estos comparten la etapa final de la resolución. Una vez determinado el vector $\boldsymbol{p_d}$ y alcanzada la convergencia del método, los tres códigos proporcionan varios resultados, además de los errores relativos y absolutos correspondientes: la distribución de tensiones, el centro de presiones y la fuerza normal total. Para ello, se procede de la siguiente forma:

1. Se construye la matriz $\boldsymbol{P} \in \mathbb{R}^{N_1 \times N_2}$, que contiene la distribución de tensiones normales en el área de contacto, asociando cada bloque de N_1 elementos de $\boldsymbol{p_d}$ a una columna de $\boldsymbol{P}$ hasta completar la dimensión lateral N_2:

$$\boldsymbol{P}(m,n) = \boldsymbol{p_d}\big((n-1)N_1 + m\big), \qquad \forall m \in N_1, \qquad \forall n \in N_2. \tag{158}$$

2. Se calcula la tracción resultante en el área de contacto σ sumando todas las entradas de la matriz $\boldsymbol{P}$:

$$\sigma = \sum_{m=1}^{N_1} \sum_{n=1}^{N_2} p_{mn}\,. \tag{159}$$

3. Se calcula el centro de presiones (x_1^p, x_2^p) a partir de la tracción resultante como sigue:

$$x_1^p = \frac{\sum_{m=1}^{N_1} \sum_{n=1}^{N_2} p_{mn} x_{1m}}{\sigma}, \tag{160}$$

$$x_2^p = \frac{\sum_{m=1}^{N_1} \sum_{n=1}^{N_2} p_{mn} x_{2n}}{\sigma}\,. \tag{161}$$

4. Por último, se calcula la fuerza normal total N_f a partir de las dimensiones de los elementos y la tracción resultante σ:

$$N_f = 4 \cdot \sigma \cdot a \cdot b. \tag{162}$$

donde a y b son el semiancho de los elementos en la dirección de rodadura y en la dirección lateral, respectivamente.

Estos valores son relevantes para la evaluación del comportamiento del sistema en condiciones de contacto, ya que permiten determinar la estabilidad del contacto y la distribución de

cargas sobre la superficie. El centro de presiones proporciona información crucial sobre el punto de aplicación de la fuerza resultante, lo que es esencial para garantizar que las cargas estén equilibradas y no provoquen deslizamientos o inestabilidades. Por otro lado, la fuerza normal total es un parámetro fundamental para evaluar la magnitud de la carga que se está transfiriendo entre las superficies en contacto, lo que influye directamente en la durabilidad y eficiencia del sistema ferroviario. Así, la correcta determinación de estos parámetros es clave para optimizar el diseño y operación de los sistemas en cuestión, asegurando tanto su rendimiento como su longevidad.

3.3. Validación de la implementación

La validación de los algoritmos implementados es un paso crucial para garantizar que el código desarrollado proporciona resultados precisos y confiables en el análisis del contacto entre rueda y carril. Aquí, el proceso de validación se basa fundamentalmente en el cálculo del error relativo entre la solución numérica obtenida con el algoritmo propuesto y una solución de referencia.

En este caso, la solución de referencia es la fuerza normal total obtenida mediante CONTACT en MATLAB, que ha servido como base para la construcción los algoritmos implementados basados en NORM. Por lo tanto, se considera la referencia más confiable. El cálculo del error relativo porcentual se define matemáticamente como:

$$E_r(\%) = \frac{|N_a - N_r|}{N_r} \cdot 100, \tag{163}$$

donde N_a representa la solución (la tracción total) del algoritmo evaluado y N_r la solución de CONTACT o solución de referencia.

Este indicador permite cuantificar la desviación de la solución numérica respecto a la referencia, y un error relativo inferior al 0,1% se considera aceptable para confirmar que la implementación del algoritmo es correcta. Este umbral ha sido seleccionado para asegurar que cualquier discrepancia sea lo suficientemente pequeña como para no comprometer la validez de las simulaciones. Es importante destacar que este 0,1% también contempla las posibles aproximaciones inherentes a los algoritmos, una consideración relevante dado que las aproximaciones son parte integral del proceso de modelado y resolución numérica.

Por otro lado, dado que los algoritmos se implementan inicialmente en MATLAB y posteriormente se traducen a Fortran, es crucial asegurar la coherencia de los resultados entre ambos lenguajes, aunque no se mida directamente el error relativo entre ellos.

Por último, además del cálculo del error relativo de la fuerza normal total, se ha dado especial prioridad a optimizar la eficiencia computacional durante las implementaciones, con el objetivo de minimizar los tiempos de ejecución. La eficiencia es un factor crucial, y tiempos de cálculo prolongados indican posibles áreas donde es necesario optimizar el código. Uno de los objetivos principales de este trabajo es implementar algoritmos que no solo sean precisos, sino también altamente eficientes. Por ello, cuando se detectaban tiempos de ejecución elevados o imprecisiones, la implementación se

consideraba incompleta. Esto llevaba a realizar ajustes y optimizaciones para garantizar tiempos de cálculo más reducidos y resultados suficientemente precisos.

En este TFM se han implementado cinco algoritmos, dos de los cuales, las variantes de NORM+E/P, ofrecen la opción de trabajar con una distribución elíptica o polinómica de las tensiones, según lo requiera el análisis. Los errores relativos obtenidos para todas las configuraciones posibles se presentan en la Tabla 4, proporcionando una evaluación detallada y comparativa del error relativo de los algoritmos en sus diferentes modos de funcionamiento.

Tabla 4. Errores relativos de los algoritmos respecto a la Fuerza Normal Total obtenida. NORM+E y NORM+P(g=2) indican que la variante NORM+E/P opera con la aproximación elíptica o la polinómica de grado 2, respectivamente.

	$N_f\ (N)$	$E_r^{N_f}(\%)$
NORM	50037,2858	0,0000039%
NORM + CG	50037,2858	0,0000039%
NORM + E	50050,2664	0,0259456%
NORM + E + CG	50050,2664	0,0259456%
NORM + P(g=2)	50085,4030	0,0961667%
NORM + P(g=2) + CG	50085,4031	0,0961668%
NORM + P(g=4)	50045,8597	0,0171389%
NORM + P(g=4) + CG	50039,7576	0,0049437%
BCCG (2)	50037,7870	0,0010055%
CONTACT (NORM)	50037,2839	

Los resultados muestran que todas las versiones presentan errores relativos inferiores al 0,1%, lo que confirma la consistencia en la implementación de los algoritmos de resolución. Es importante destacar que el método NORM+E/P emplea una distribución de tensiones aproximada matemáticamente, lo que podría justificar el error relativo ligeramente mayor en comparación con otros métodos.

3.4. Conclusiones

En este capítulo se ha presentado la metodología utilizada para la implementación y optimización de los tres algoritmos empleados en este estudio: NORM, NORM con la aproximación de la distribución de tensiones mediante una base elíptica o polinómica y el método BCCG(K). Estos algoritmos fueron implementados en Fortran con un enfoque en maximizar la precisión, la eficiencia computacional, la robustez y la escalabilidad, aspectos que son esenciales para su evaluación posterior mediante benchmarking.

La implementación se llevó a cabo siguiendo un enfoque iterativo, donde cada versión del código fue meticulosamente ajustada para mejorar tanto el rendimiento como la exactitud de los resultados. El uso inicial de MATLAB facilitó la experimentación y el ajuste detallado del código en un entorno interactivo antes de trasladarlo a Fortran, lo que permitió una transición más eficiente y la optimización de los códigos finales. Durante la implementación en Fortran, se optimizaron los códigos

mediante diversas estrategias, como la selección adecuada de algoritmos numéricos, la gestión eficiente de la memoria, la eliminación de operaciones superfluas y la optimización de aquellas que podían ejecutarse de manera más eficiente. Estas acciones contribuyeron a reducir tanto el consumo de recursos como los tiempos de ejecución. Además, se utilizaron técnicas de *profiling manual* para identificar y corregir cuellos de botella, lo que permitió mejorar significativamente el rendimiento general de los algoritmos.

Estos avances en la implementación han dado como resultado algoritmos altamente optimizados, listos para su evaluación mediante benchmarking. En el próximo capítulo, se detallará el hardware empleado, el escenario del problema y los resultados obtenidos, con el objetivo de realizar un análisis comparativo del rendimiento y la eficacia de los algoritmos en un entorno controlado y rigurosamente definido.

4. EJECUCIÓN Y BENCHMARKING DE LOS ALGORITMOS

4.1. Introducción

Tras haber detallado la metodología de implementación y optimización de los algoritmos NORM, NORM+E/P y BCCG(K) en el capítulo anterior, este capítulo se centrará en la evaluación comparativa de estos algoritmos mediante benchmarking. El objetivo principal es analizar su eficacia y eficiencia bajo un entorno controlado, utilizando para ello un caso de estudio específico.

Para llevar a cabo esta evaluación, es fundamental considerar varios aspectos clave. En primer lugar, se describirá el hardware utilizado en las pruebas, que juega un papel crucial en la medición del rendimiento computacional de los algoritmos. A continuación, se definirá el escenario especifico a resolver, detallando las condiciones y parámetros bajo los cuales se han ejecutado los algoritmos, asegurando así la consistencia y la comparabilidad de los resultados obtenidos.

Finalmente, se presentarán los resultados del benchmarking, donde se analizará el desempeño de cada algoritmo en términos de tiempo de ejecución, uso de recursos y precisión de los resultados. Esta evaluación permitirá no solo identificar las fortalezas y debilidades de cada enfoque, sino también ofrecer una visión integral sobre cuál de los algoritmos es más adecuado según las características del problema y los objetivos específicos del estudio.

4.2. Hardware utilizado

El hardware desempeña un papel fundamental en el desarrollo, implementación y evaluación de métodos numéricos complejos, siendo crucial para asegurar un análisis preciso de las prestaciones de los códigos implementados. Un sistema óptimo no solo garantiza la capacidad de ejecutar las cargas de trabajo con eficiencia, sino que también resulta esencial para obtener resultados fiables en el posterior análisis comparativo de los algoritmos.

El equipo utilizado para este proyecto es un MSI Thin GF63 12VF, que incorpora un procesador Intel Core i7-12650H de 12ª generación, con 10 núcleos físicos y 20 hilos, operando a una frecuencia base de 2.3 GHz. Estas especificaciones permiten gestionar de manera eficiente las tareas de cálculo intensivo, siendo fundamentales para asegurar un rendimiento adecuado en la ejecución de los algoritmos numéricos implementados. El procesador ofrece la capacidad necesaria para manejar el volumen de operaciones matemáticas involucradas, optimizando los tiempos de respuesta en los procesos computacionales más exigentes del proyecto.

Asimismo, el sistema dispone de 16 GB de memoria RAM DDR4 a 3200 MHz, suficiente para manejar grandes volúmenes de datos y soportar el elevado número de iteraciones inherente a los métodos numéricos implementados. El equipo está configurado en modo de máximo rendimiento energético, lo que garantiza un uso óptimo de los recursos del hardware durante la ejecución de los códigos.

El sistema operativo Windows 11 Pro, versión 10.0.22621.2506, ofrece un entorno robusto para aplicaciones que requieren un uso intensivo de recursos. La placa base MSI MS-16R8, junto con

la BIOS actualizada a la versión E16R8IMS.10D, garantiza la compatibilidad y estabilidad del conjunto de componentes, asegurando un rendimiento óptimo en la evaluación de los métodos numéricos implementados.

En conjunto, este hardware, configurado para operar a su máxima capacidad, ofrece el soporte necesario para la correcta implementación y evaluación de los algoritmos desarrollados. Este entorno garantiza que las pruebas y análisis de rendimiento se realicen bajo condiciones que optimizan tanto la eficiencia como la fiabilidad de los resultados obtenidos.

4.3. Escenario de contacto

El escenario de contacto influye directamente en el comportamiento del algoritmo. Un mayor número de elementos aumenta las iteraciones, lo que incrementa el coste computacional. Los códigos están parametrizados para ajustarse a diversas configuraciones, aunque el tiempo de ejecución dependerá de la complejidad del escenario.

En cuanto al estudio realizado, el escenario de contacto es relativamente simple y permite al autor validar las implementaciones y las traducciones entre MATLAB y Fortran, además de obtener las mediciones necesarias para el análisis de rendimiento. Específicamente, el caso de estudio se basa en un eje montado centrado en la vía, sin ángulo de lazo. El perfil de rueda corresponde al S1002, y el del carril al UIC60 con una inclinación de 1/40.

4.3.1. Parámetros del material

Tanto las ruedas como los carriles están fabricados predominantemente en acero debido a su alta resistencia y capacidad para soportar cargas elevadas. Un perfil de rueda S1002 y un carril UIC60 pueden modelarse perfectamente con las siguientes características del material:

- Módulo de Elasticidad (E): $210\ GPa$.
- Coeficiente de Poisson (ν): 0.3.

4.3.2. Discretización y geometría del área potencial de contacto

Como se mencionó en el contexto teórico sobre el método NORM, CONTACT es una variante del Método de los Elementos de Contorno. Esto implica que el problema se resuelve mediante la discretización del dominio continuo. En el caso de la mecánica de contacto rueda-carril, dicha discretización consiste en dividir el dominio –las superficies de la rueda y el carril en contacto— en un área potencial de contacto lo suficientemente grande para incluir, al menos, todos los elementos del área real de contacto.

En el caso de estudio propuesto, la discretización se realiza de la siguiente forma:

- $N_1 = 40$: Número de elementos en la dirección de rodadura X_1.
- $N_2 = 40$: Número de elementos en la dirección lateral X_2.
- $N_e = N_1 \cdot N_2 = 1600$: Número total de elementos en el área potencial de contacto.

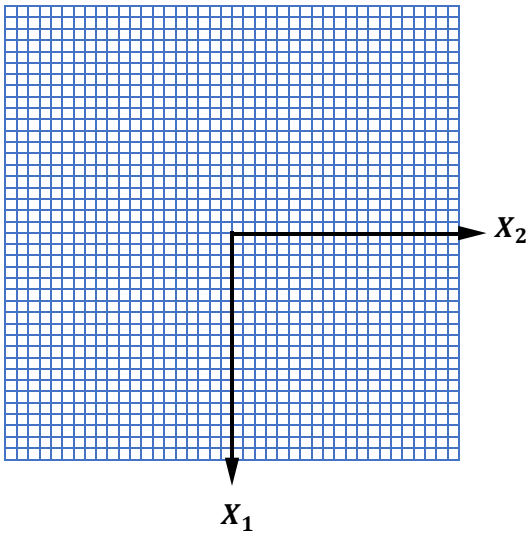

Figura 22. Área potencial de contacto del caso de estudio empleado (elaboración propia).

Mediante esta discretización, el área potencial de contacto resulta ser similar a la ilustrada en la Figura 22. Los parámetros definidos a partir de ella son los siguientes:

- $\boldsymbol{H}$: Matriz $N_1 \times N_2$ que contiene las distancias entre las superficies indeformadas en contacto de la rueda y el carril para todos los elementos. Esta matriz se convierte posteriormente en el vector $\boldsymbol{h} \in \mathbb{R}^{N_e}$.
- $\boldsymbol{x_1}$: Vector de coordenadas de los elementos en el área potencial de contacto según la dirección de rodadura $\boldsymbol{X_1}$. Los valores se encuentran en el rango $[-0.0063397\ ,\ 0.0063397]$.
- $\boldsymbol{x_2}$: Vector de coordenadas de los elementos en el área potencial de contacto según la dirección lateral $\boldsymbol{X_2}$. Los valores se encuentran en el rango $[0.00027625\ ,\ 0.02182375]$.
- $\boldsymbol{a} = 1.6256 \cdot 10^{-4}$: Semiancho de los elementos en la dirección de rodadura $\boldsymbol{X_1}$.
- $\boldsymbol{b} = 2.7625 \cdot 10^{-4}$: Semiancho de los elementos en la dirección lateral $\boldsymbol{X_2}$.

Estas dimensiones y coordenadas permiten definir con precisión la geometría del área potencial de contacto, lo cual es esencial para evaluar la interacción de las superficies bajo carga utilizando los métodos numéricos implementados. A partir de estos parámetros, y mediante los algoritmos implementados, es posible determinar la distribución de tensiones normales, la fuerza normal total y el centro de presiones.

4.4. Ejecución de los algoritmos

4.4.1. Argumentos de entrada

Los parámetros asociados al escenario de contacto específico, para el cual se busca determinar tanto el área de contacto como los esfuerzos correspondientes, constituyen las entradas para los diferentes algoritmos. Tal como se mencionó anteriormente, la primera parte del código es común en todos los algoritmos. Esta etapa incluye la construcción de la matriz $\boldsymbol{A}$, la creación del vector $\boldsymbol{h}$ de distancias indeformadas, y concluye con la invocación del método BCCG(K), la eliminación de Gauss o el método del gradiente conjugado.

A partir de aquí, cada algoritmo sigue su propio flujo, según lo descrito en el apartado 3.2.4. Esto implica que, posteriormente, se procede ya sea a la resolución de los sistemas de ecuaciones asociados o al ajuste del conjunto de variables del problema durante iteraciones externas, una vez se ha verificado la convergencia en las iteraciones internas.

A modo de recordatorio, las variables básicas de entrada a los algoritmos NORM y NORM+E/P y BCCG(K) son:

- E: Módulo de Elasticidad.
- v: Coeficiente de Poisson.
- N_1, N_2: Número de elementos en las direcciones de rodadura y lateral, respectivamente.
- $\boldsymbol{H}$: Matriz que contiene las distancias entre las superficies indeformadas en contacto de la rueda y el carril para todos los elementos.
- $\boldsymbol{x_1}, \boldsymbol{x_2}$: Vectores de coordenadas de los elementos en el área potencial de contacto según la dirección de rodadura y lateral, respectivamente.
- a, b: Semianchos de los elementos en las direcciones de rodadura y lateral, respectivamente.

Los algoritmos NORM y NORM+E/P parten de estas variables para generar las que emplearán en las etapas posteriores. Una vez definido el sistema de ecuaciones, este se resuelve inicialmente mediante el método matemático correspondiente (GE o CG), y luego se aplican las técnicas descritas para generar subproblemas que serán resueltos utilizando los mismos métodos matemáticos.

En el caso del BCCG(K), el algoritmo parte de la definición previa del sistema de ecuaciones $\boldsymbol{A p_d = h}$. Su invocación requiere, ya sea por reserva previa de memoria o por cálculos preliminares, las siguientes variables como argumentos de entrada: K, $\boldsymbol{A}$, $\boldsymbol{p_d}$, $\boldsymbol{h}$, $\boldsymbol{g}$, $\boldsymbol{r}$, $\boldsymbol{v}$, $\boldsymbol{q}$, $\boldsymbol{x_{prev}}$, $\boldsymbol{r_{prev}}$, tol, $maxit$, $\boldsymbol{upd}$, $\boldsymbol{Ix_{bnd}}$, $\boldsymbol{msk_{bnd}}$, $\boldsymbol{Ix_{prj}}$, $\boldsymbol{Ix_{rel}}$ y N_e.

4.4.2. Salidas de los algoritmos

La resolución del problema normal de contacto mediante los algoritmos presentados permite determinar varios parámetros clave, entre ellos la distribución de presiones normales, la fuerza normal total, el centro de presiones y el área de contacto:

- **Distribución de tensiones normales ($\boldsymbol{P}$):** Los códigos generan una matriz $\boldsymbol{P} \in \mathbb{R}^{N_1 \times N_2}$, que recoge la tracción normal en cada elemento del área de contacto, expresada en N/m^2. Este resultado es esencial para comprender cómo se distribuye la carga sobre la superficie de contacto.
- **Fuerza Normal Total ($\boldsymbol{N_f}$):** La suma de todas las tensiones normales en los elementos discretizados, expresada en Newtons (N). Esta fuerza es crítica para evaluar la magnitud total de la carga transmitida entre las superficies en contacto. Es la salida principal de los códigos programados.
- **Coordenadas del centro de presiones ($\boldsymbol{x_1^p}, \boldsymbol{x_2^p}$):** Estas coordenadas identifican el punto donde se concentra la fuerza de presión normal resultante, un dato clave para la estabilidad del contacto y la distribución de tensiones.

- **Área de contacto:** La configuración final de la matriz $\boldsymbol{P}$ define la forma del área de contacto (véase la Figura 23).

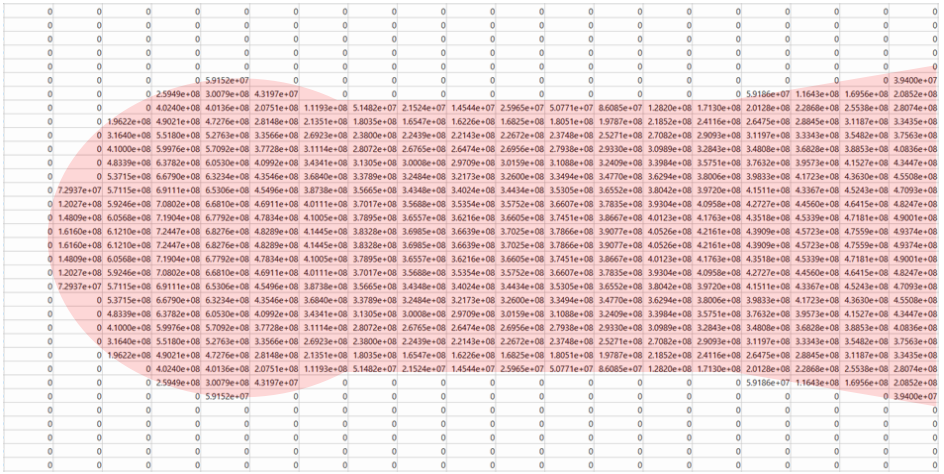

Figura 23. Forma del área de contacto (roja) ilustrada por la matriz $\boldsymbol{P}$ tras la resolución del problema normal (elaboración propia).

Además de estos parámetros, cuyo cálculo se ha explicado en la sección 3.2.4.6, el código monitorea el rendimiento de los algoritmos a través de los siguientes métricas y mediciones:

- **Medición del tiempo de cálculo:** Se mide el tiempo necesario para ejecutar el algoritmo desde la lectura de los datos hasta la obtención de la fuerza normal total. Esta medición se realiza antes de calcular el error relativo y absoluto de las presiones. El tiempo se desglosa en:
  - *Wall-clock time*: Tiempo real transcurrido durante la ejecución.
  - *CPU time*: Tiempo de procesamiento efectivo utilizado durante la ejecución.
- **Errores relativos y absolutos:** Tras medir los tiempos de cálculo, se cuantifican los errores relativos para la fuerza normal total y la distribución de presiones, tomando en este último caso el valor máximo. Adicionalmente, se calcula el error absoluto máximo para la distribución de tensiones. Estos errores son esenciales para validar la precisión y robustez del código implementado, comparado con la referencia en MATLAB (CONTACT).
- **Impresiones temporales:** A lo largo de la implementación, se emplean comandos como print* en puntos específicos del código para verificar el estado de las variables clave. Históricamente, esto ha permitido comprobar el valor de entradas de matrices y otros parámetros relevantes. En las versiones finales de los algoritmos, se ha empleado para reportar el número total de iteraciones y el uso de memoria.

4.5. Extracción de datos para el análisis de los algoritmos

En el proceso de extracción de datos, se adoptó un enfoque riguroso para garantizar tanto la precisión como la efectividad en el análisis de los algoritmos. Inicialmente, se realizó una ejecución preliminar de los códigos con el objetivo de validar su implementación y establecer una línea base de referencia para las evaluaciones subsiguientes.

Posteriormente, se efectuaron 20 ejecuciones consecutivas para cada código, durante las cuales se registraron de manera precisa los tiempos de CPU. Es importante destacar que estas mediciones se realizaron exclusivamente con los archivos ejecutables (.exe) generados, garantizando así que los tiempos reflejan fielmente el rendimiento del código compilado. El ordenador utilizado operaba en una configuración de máximo rendimiento energético, asegurando que los tiempos registrados representaran el desempeño óptimo del hardware. A partir de esta serie de ejecuciones, se calculó una media del tiempo de cálculo, proporcionando una métrica sólida y consistente para evaluar el rendimiento temporal de los algoritmos bajo condiciones controladas y repetitivas.

Simultáneamente, los códigos fueron diseñados para reportar tanto el error relativo como el absoluto. Los errores se calculaban comparando los resultados obtenidos mediante el método con los de referencia (CONTACT en MATLAB), específicamente en relación con la fuerza normal y la distribución de tensiones. Esta comparación resulta crucial para validar la correcta implementación de los algoritmos y evaluar las posibles discrepancias al utilizar un método alternativo.

Además, se realizó un control exhaustivo del uso de memoria, mostrando en pantalla el valor del índice I_{END}, que indica la memoria efectivamente utilizada en el bloque de memoria $\boldsymbol{R}$. Este control es crucial para identificar posibles ineficiencias en la gestión de recursos, las cuales podrían afectar directamente el rendimiento global del sistema.

En cuanto al comportamiento iterativo y la convergencia de los algoritmos, se registró tanto el número de iteraciones como una estimación del coste computacional asociado a cada uno de ellos. La medición de las iteraciones abarcó la resolución del sistema de ecuaciones inicial, el número de actualizaciones del área de contacto y, en caso de aplicarse, las iteraciones necesarias del gradiente conjugado. Además, se incluyó la dimensión inicial del problema (y su parametrización), junto con los FLOPs (*Floating Point Operations*), que estiman el número de operaciones aritméticas requeridas según las dimensiones de las matrices y vectores involucrados (Golub & Van Loan, 2013). La Tabla 5 muestra los FLOPs para distintas operaciones básicas, lo que permite evaluar el coste computacional de los algoritmos de manera más precisa.

Tabla 5. FLOPs de las operaciones básicas (Golub & Van Loan, 2013).

Operación	Dimensión	FLOPs
$\alpha = \boldsymbol{x}^T\boldsymbol{y}$	$\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$	$2n$
$\boldsymbol{y} = \boldsymbol{y} + a\boldsymbol{x}$	$\boldsymbol{a} \in \mathbb{R}, \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$	$2n$
$\boldsymbol{y} = \boldsymbol{y} + \boldsymbol{A}\boldsymbol{x}$	$\boldsymbol{A} \in \mathbb{R}^{m\times n}, \boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{y} \in \mathbb{R}^m$	$2mn$
$\boldsymbol{A} = \boldsymbol{A} + \boldsymbol{y}\boldsymbol{x}^T$	$\boldsymbol{A} \in \mathbb{R}^{m\times n}, \boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{y} \in \mathbb{R}^m$	$2mn$
$\boldsymbol{C} = \boldsymbol{C} + \boldsymbol{A}\boldsymbol{B}$	$\boldsymbol{A} \in \mathbb{R}^{m\times r}, \boldsymbol{B} \in \mathbb{R}^{r\times n}, \boldsymbol{C} \in \mathbb{R}^{m\times n}$	$2mnr$

El cálculo de los FLOPs se realiza considerando que, en el proceso de resolución del problema de contacto, se ejecuta la eliminación gaussiana o el gradiente conjugado un total de k veces. De este modo, los FLOPs correspondientes a la eliminación de Gauss y al gradiente conjugado pueden definirse como:

$$FLOPs_{GE} = \sum_{i=1}^{k} \left(\frac{2n_i^3}{3} + 2n_i^2 \right), \tag{164}$$

$$FLOPs_{CG} = \sum_{i=1}^{k} m_i \left(6n_i^2 + 14n_i \right), \tag{165}$$

donde k representa el número de iteraciones sobre el área de contacto necesarias para la resolución del problema, m_i el número de iteraciones del gradiente conjugado en la iteración i del algoritmo y n_i la dimensión de las matrices (y vectores) en la iteración i.

Todos estos parámetros proporcionan una base sólida para la evaluación rigurosa de los algoritmos, asegurando que el análisis sea preciso y que se puedan identificar y corregir posibles áreas de mejora Todos estos parámetros proporcionan una base sólida para la evaluación rigurosa de los algoritmos, asegurando que el análisis sea preciso y que se puedan identificar y corregir posibles áreas de mejora. La meticulosidad en la recolección y evaluación de estos datos es fundamental para interpretar correctamente el rendimiento y la eficacia de los algoritmos en el contexto de su aplicación.

4.6. Análisis comparativo de los algoritmos (*benchmarking*)

4.6.1. Introducción

Este apartado presenta los resultados del benchmarking, centrado en aspectos clave como el tiempo de ejecución, la precisión de los resultados y la gestión de la memoria de los algoritmos evaluados. Mediante múltiples ejecuciones bajo condiciones controladas, se ha recopilado un conjunto de datos que permite comparar de manera precisa la eficiencia y eficacia de cada uno de los algoritmos implementados.

El tiempo de cálculo registrado en cada ejecución, junto con los parámetros asociados, proporciona una medida clara de la eficiencia temporal de los algoritmos. Además, se comparan los errores relativo y absoluto respecto a la solución de referencia (CONTACT en MATLAB), lo que asegura la exactitud de los cálculos y permite identificar las diferencias al utilizar distintos métodos de resolución. También se analiza la gestión de la memoria, con un monitoreo detallado del uso de recursos para identificar posibles ineficiencias.

Cabe destacar que, aunque se han propuesto un total de cinco algoritmos, los correspondientes a NORM+E/P pueden operar con dos representaciones diferentes: la elíptica (E) y la polinómica (P). En el análisis se presentan varias configuraciones para dos de estos cinco algoritmos. Para la base polinómica, se evaluarán dos grados específicos: $g = 2$ y $g = 4$, para explorar el rendimiento y la precisión empelando distintos niveles de complejidad. Por otro lado, el método del gradiente conjugado con restricciones acotadas se configurará con $K = 2$, y sus resultados se referenciarán mediante el acrónimo BCCG (2).

Este análisis proporciona una evaluación completa del rendimiento computacional, estableciendo una base sólida para interpretar los resultados y guiar la optimización de futuras implementaciones.

4.6.2. Tiempo de ejecución

El análisis de los tiempos de ejecución de los algoritmos se centra en el tiempo de cálculo necesario para obtener el valor de la tensión normal total N_f. Los datos recopilados se resumen en la Tabla 7, que incluye los tiempos promedio de CPU, la eficiencia computacional respecto a NORM, el número total de iteraciones por ejecución, el parámetro FLOPs según la dimensión n de las matrices y el número inicial de incógnitas. Para mayor claridad, la Tabla 6 presenta los tiempos específicos obtenidos en cada una de las 20 ejecuciones de los algoritmos.

Tabla 6. Tiempos de ejecución: tiempo de CPU de los algoritmos (en segundos) para todas las ejecuciones.

N.º	NORM	NORM +CG	NORM +E	NORM +E+CG	NORM +P(g=2)	NORM +P(g=2)+CG	NORM +P(g=4)	NORM +P(g=4)+CG	BCCG (2)
1	4,3438	0,2969	0,0625	0,0000	0,0313	0,1094	0,1250	13,5156	0,0938
2	5,0313	0,2500	0,0469	0,0625	0,1563	0,0938	0,2656	9,3906	0,0938
3	4,8750	0,3125	0,0469	0,0313	0,0469	0,2031	0,2031	12,8281	0,0469
4	4,1406	0,4531	0,0625	0,0625	0,1094	0,1563	0,2031	12,2031	0,1250
5	4,6563	0,3750	0,0313	0,0000	0,0625	0,1875	0,1250	13,6719	0,1250
6	5,1094	0,5938	0,0938	0,0625	0,1094	0,2031	0,1250	15,5625	0,0625
7	4,2031	0,1406	0,0156	0,0000	0,0938	0,2813	0,1406	15,3438	0,1875
8	5,1719	0,1094	0,0313	0,0469	0,1406	0,1717	0,1563	16,0313	0,1875
9	4,5938	0,3594	0,0938	0,1563	0,0625	0,2813	0,0781	16,6719	0,0781
10	5,8594	0,1406	0,0156	0,0938	0,0938	0,2031	0,0625	14,0938	0,1250
11	4,3438	0,2188	0,0000	0,0781	0,1406	0,1250	0,2188	15,3125	0,1094
12	4,3594	0,2500	0,1406	0,0469	0,0625	0,2500	0,1250	12,8125	0,1094
13	3,6250	0,3281	0,0156	0,0000	0,0313	0,0938	0,2031	14,9219	0,1094
14	4,8125	0,4219	0,0781	0,0469	0,1563	0,1563	0,0938	14,7031	0,1250
15	4,4219	0,1094	0,0781	0,0469	0,0625	0,2344	0,1563	14,6563	0,1094
16	5,0000	0,5469	0,1250	0,1094	0,0938	0,3125	0,0938	16,4063	0,1094
17	4,3906	0,3125	0,1563	0,0469	0,1094	0,1719	0,1406	15,0156	0,1719
18	5,3906	0,2344	0,0781	0,0938	0,0938	0,1563	0,0781	16,0938	0,2031
19	4,6406	0,3438	0,0000	0,1563	0,0313	0,2188	0,0781	13,5469	0,1406
20	5,2344	0,2344	0,0313	0,0938	0,0625	0,1875	0,1094	15,8438	0,0938

A la vista de los resultados, el algoritmo NORM (CONTACT), que sirve como referencia para la evaluación temporal mediante el valor de eficiencia computacional, presenta un tiempo medio de cálculo de 4.71016 segundos, alcanzando la solución en 7 iteraciones con 1600 elementos en la malla.

Comparativamente, otros algoritmos muestran mejoras significativas en el tiempo de ejecución. Por ejemplo, al aplicar el método del gradiente conjugado al método NORM, obteniendo así el método NORM + CG, el tiempo de ejecución se reduce a 0.30156 segundos, logrando una

eficiencia computacional de 15.6 respecto a NORM. Esta mejoría se consigue a costa de un total de 338 iteraciones, lo que muestra la eficiencia aportada por el método CG.

Tabla 7. Tiempos de ejecución: Tiempo medio de CPU, eficiencia computacional, número total de iteraciones y número de elementos en el área potencial.

	Tiempo medio de CPU (s)	**Eficiencia computacional** (respecto a NORM)	**Iteraciones** (Resolución inicial + Iteraciones + CG)	**FLOPs**	**Número inicial de incógnitas**
NORM	4,7102	1,0	7	5542277845	1600 (40×40)
NORM + CG	0,3016	15,6	338	2147021572	1600 (40×40)
NORM + E	0,0602	78,3	7	1812611	80 $(2N_s)$
NORM + E + CG	0,0617	76,3	718	23211028	80 $(2N_s)$
NORM + P(g=2)	0,0875	53,8	6	5307984	120 $[(2+1)N_s]$
NORM + P(g=2) + CG	0,1898	24,8	18512	1329983952	120 $[(2+1)N_s]$
NORM + P(g=4)	0,1391	33,9	7	28355733	200 $[(4+1)N_s]$
NORM + P(g=4) + CG	14,4313	0,3	290585	63903412620	200 $[(4+1)N_s]$
BCCG (2)	0,1203	39,1	20	5542277845	1600 (40×40)

El algoritmo NORM + E destaca por registrar el menor tiempo de CPU, 0.06016 segundos con 7 iteraciones. Esto implica que el método es 78.3 veces más rápido que NORM. La propuesta de Giner Navarro et al. (2023) proporciona un algoritmo que optimiza de manera significativa el tiempo requerido para obtener la solución.

Con el objetivo de mejorar aún más este resultado, se propuso utilizar el método del gradiente conjugado en combinación con NORM + E, formando el método NORM + E + CG. Sin embargo, no se observó una mejora significativa. Esto parece deberse a que el número de iteraciones asciende a 718 al considerar las iteraciones del CG, la resolución inicial del sistema completo y la determinación de las nuevas áreas de contacto. Además, la eliminación gaussiana característica del método NORM (y, por ende, de NORM + E) resulta más eficiente debido a la reducción de la dimensión del vector incógnita (de 1600 a 80 en la primera resolución). Aunque el tiempo de CPU registrado para NORM+E+CG es de 0.06172 segundos, esta ligera pérdida de eficiencia se atribuye al aumento de iteraciones necesarias para alcanzar la convergencia.

Por su parte, el algoritmo NORM + P(g=2) ofrece un tiempo de 0.0875 segundos, representando una eficiencia computacional de 53.8 en comparación con NORM, con un total de 6 iteraciones. Este es el menor número de iteraciones conseguido hasta el momento, aunque el tiempo total es algo superior al de la aproximación elíptica en sus dos variantes.

La implementación del gradiente conjugado en NORM + P(g=2) + CG no produjo los resultados esperados, pues aumentó el tiempo de CPU en aproximadamente 0.1 segundos en comparación con su variante programada utilizando eliminación gaussiana. Además, el número de iteraciones se

incrementa significativamente, alcanzando las 18512. Un resultado similar se observa en NORM + P(g=4) + CG, donde el tiempo de ejecución se eleva considerablemente a 14.4313 segundos, un valor muy superior al de las otras implementaciones. Este aumento drástico en el tiempo de CPU, junto con un número de iteraciones extremadamente alto (290585), podría indicar que la combinación con el gradiente conjugado no es eficiente para este algoritmo o que existe un fallo en la implementación.

En cuanto a NORM + P(g=4), el tiempo registrado fue de 0.1391 segundos. Esto representa una eficiencia computacional 33.9 veces mayor en comparación con NORM. Aunque este tiempo es mayor en comparación con la aproximación elíptica y la polinómica con $g = 2$, este incremento puede atribuirse al aumento en las dimensiones de la matriz y, por ende, a una mayor complejidad computacional. No obstante, es relevante señalar que esta configuración mejora la precisión de los resultados, lo cual es crucial en contextos donde la exactitud es prioritaria, a pesar de que el rendimiento en términos de tiempo sea relativamente inferior.

Finalmente, el algoritmo BCCG (2) registra un tiempo de CPU de 0.12031 segundos, lo que se traduce en una eficiencia computacional 39.1 veces superior a la del método NORM.

Si se analiza el número de FLOPs (véase la Tabla 8), los tiempos de cálculo están fuertemente relacionados con la cantidad de operaciones requeridas por cada algoritmo, lo que se observa en los resultados obtenidos. El algoritmo NORM emplea 5542277845 FLOPs, lo que refleja una carga computacional considerable. Sin embargo, NORM + CG solo emplea un total de 2147021572 FLOPs, con un tiempo mucho menor, demostrando que el uso del gradiente conjugado optimiza el tiempo de cálculo a pesar de requerir un número mayor de operaciones.

El método más eficiente en cuanto al número de FLOPs es NORM + E, con solo 1812611 FLOPs, lo que justifica su tiempo de cálculo extremadamente bajo. En contraste, NORM + P(g=4) + CG alcanza hasta 63903412620 FLOPs, mostrando la enorme cantidad de operaciones requeridas. Esto sugiere que la combinación con el Gradiente Conjugado podría no ser eficiente en este caso o, alternativamente, que podrían existir errores en la implementación que estén afectando negativamente al rendimiento.

Un caso intermedio es el método BCCG (2), con 307648000 FLOPs, combinando eficiencia y costo computacional. Aunque no es tan eficiente como NORM + E, ofrece una solución más rápida y precisa que otras alternativas.

Tabla 8. Estimación del número de FLOPs de los algoritmos implementados.

	Expresión para el cálculo de los FLOPs	**FLOPS**
NORM	(164)	5542277845
NORM + CG	(165)	2147021572
NORM + E	(164)	1812611
NORM + E + CG	(165)	23211028
NORM + P(g=2)	(164)	5307984
NORM + P(g=2) + CG	(165)	1329983952

	Expresión para el cálculo de los FLOPs	FLOPS
NORM + P(g=4)	(164)	28355733
NORM + P(g=4) + CG	(165)	63903412620
BCCG (2)	(165)	307648000

En resumen, los métodos BCCG (2) y NORM + E/P (en sus dos variantes) son altamente competitivos para la resolución del problema normal de contacto rueda-carril sin rozamiento. Entre ellos, NORM + E resulta ser el más eficiente. Para mayor claridad, la Tabla 9 muestra las eficiencias computacionales de todos los algoritmos propuestos, tomando cada uno de ellos como referencia. La eficiencia computacional se calcula como:

$$\eta_r = \frac{t_c}{t_r}, \tag{166}$$

donde η_r es la eficiencia computacional del método de la fila, t_c el tiempo de cálculo medio del método de la columna y t_r el tiempo de cálculo medio del método de la fila.

Tabla 9. Tiempos de ejecución: eficiencia computacional del método de la columna respecto al método de la fila.

	NORM	NORM +CG	NORM +E	NORM +E+CG	NORM +P(g=2)	NORM +P(g=2)+CG	NORM +P(g=4)	NORM +P(g=4)	BCCG (2)
NORM	**1,00**	0,06	0,01	0,01	0,02	0,04	0,03	3,06	0,03
NORM + CG	15,62	**1,00**	0,20	0,20	0,29	0,63	0,46	47,85	0,40
NORM + E	78,30	5,01	**1,00**	1,03	1,45	3,16	2,31	239,90	2,00
NORM + E + CG	76,32	4,89	0,97	**1,00**	1,42	3,08	2,25	233,82	1,95
NORM +P(g=2)	53,83	3,45	0,69	0,71	**1,00**	2,17	1,59	164,93	1,38
NORM +P(g=2)+CG	24,81	1,59	0,32	0,33	0,46	**1,00**	0,73	76,02	0,63
NORM +P(g=4)	33,87	2,17	0,43	0,44	0,63	1,37	**1,00**	103,78	0,87
NORM +P(g=4)+CG	0,33	0,02	0,00	0,00	0,01	0,01	0,01	**1,00**	0,01
BCCG (2)	39,15	2,51	0,50	0,51	0,73	1,58	1,16	119,95	**1,00**

4.6.3. Precisión de los resultados

La Tabla 10 presenta una evaluación exhaustiva de la precisión de los resultados obtenidos mediante los distintos algoritmos, analizando el error relativo asociado a la fuerza normal (N_f) y el máximo error relativo y absoluto en la distribución de tensiones ($\boldsymbol{P}$). En este caso, la referencia es la misma empleada para la validación de los modelos, es decir, CONTACT (NORM en MATLAB).

Tabla 10. Precisión de los resultados: errores relativos y absolutos. El superíndice $*$ denota que se trata del error máximo.

	$N_f\ (N)$	$E_r^{N_f}(\%)$	$E_r^{P*}(\%)$	$E_{abs}^{P*}(MPa)$
NORM	50037,2858	0,0000039%	0,0032%	0,00026
NORM + CG	50037,2858	0,0000039%	0,0032%	0,00026
NORM + E	50050,2664	0,0259456%	1766,28%	65,518
NORM + E + CG	50050,2664	0,0259456%	1766,28%	65,518
NORM + P(g=2)	50085,4030	0,0961667%	4616,12%	171,230
NORM + P(g=2) + CG	50085,4031	0,0961668%	4616,13%	171,231
NORM + P(g=4)	50045,8597	0,0171389%	1300,61%	51,024
NORM + P(g=4) + CG	50039,7576	0,0049437%	1681,21%	62,363
BCCG (2)	50037,7870	0,0010055%	15,44%	1,237
CONTACT	**50037,2839**			

El algoritmo NORM muestra una alta precisión con una fuerza normal total de 50037.2858 N. Los errores relativos asociados a N_f ($E_r^{N_f}$) y a la distribución de tensiones (E_r^{P*}) son extremadamente bajos, registrando valores del 0.0000039% y 0.0032%, respectivamente. Asimismo, el error absoluto máximo en la distribución de tensiones (E_{abs}^{P*}) es de solo 0.00026 MPa. Estos resultados indican una correcta implementación del procedimiento NORM de CONTACT en Fortran.

El método NORM + CG refleja la misma precisión que NORM, mostrando valores idénticos de N_f, $E_r^{N_f}$, E_r^{P*} y E_{abs}^{P*}, lo que sugiere que el uso del gradiente conjugado no afecta a la precisión en el cálculo de la fuerza normal o la distribución de tensiones.

Sin embargo, los algoritmos como NORM + E y NORM + P(g=2) muestran una reducción en la precisión. El algoritmo NORM + E registra un incremento en el error relativo $E_r^{N_f}$ hasta un 0.0259456%, con un error relativo máximo en la distribución de tensiones que se eleva al 1766.28% y un error absoluto máximo E_{abs}^{P*} de 65.518 MPa. Esta disminución de la precisión se concentra principalmente en los elementos ubicados en los bordes del área de contacto, donde la representación de las tensiones resulta menos precisa. El uso del CG en combinación con NORM + E (NORM + E + CG) no mejora la situación, manteniéndose los mismos valores de error que en NORM + E.

El algoritmo NORM + P(g=2) también muestra una disminución significativa en la precisión, con un error relativo en la fuerza normal total resultante del 0.0961667% y un error relativo máximo en la distribución de tensiones (E_r^{P*}) que alcanza el 4616.12%, con un error absoluto máximo de 171.230 MPa. Nuevamente, esta reducción en la precisión es más acusada en los bordes del área de contacto. La combinación de NORM + P(g=2) con el gradiente conjugado (NORM + P(g=2) + CG) apenas varía los resultados, presentando errores similares.

Sin embargo, NORM + P(g=4) presenta una mejora en precisión respecto a NORM + P(g=2), con un error relativo $E_r^{N_f}$ del 0.0171389% y un error relativo máximo en la distribución de tensiones del 1300.61%. El error absoluto máximo en la distribución de tensiones es de 51.024 MPa, lo que supone

una mejora significativa en comparación con la variante de $g = 2$, aunque el error sigue siendo considerablemente más alto que en las aproximaciones elípticas. La combinación de NORM + P(g=4) con el gradiente conjugado muestra una ligera mejora, con $E_r^{N_f}$ de 0.0049437%, pero el error relativo en la distribución de tensiones sigue siendo elevado, alcanzando un 1681.21%, y el error absoluto $E_{abs}^{\boldsymbol{P}*}$ es de 62.363 MPa. Aún con esta mejora en precisión, el tiempo de cálculo empeora drásticamente cuestionando la viabilidad de esta solución.

Por último, el algoritmo BCCG (2) muestra una precisión razonable con una fuerza normal total de 50037,7870 N, un error relativo $E_r^{N_f}$ del 0,0010055%, un $E_r^{\boldsymbol{P}*}$ de 15,44% y un error absoluto máximo $E_{abs}^{\boldsymbol{P}*}$ de 1,237 MPa. Aunque no es tan preciso como los métodos NORM y NORM + CG, BCCG (2) mantiene una buena precisión en los resultados, lo que lo convierte en una opción eficiente y precisa.

En resumen, mientras que NORM, NORM + CG y BCCG (2) demuestran una elevada precisión, otros métodos como NORM + E y NORM + P (y sus variantes con el Gradiente Conjugado) muestran una disminución en la precisión debido al modelado aproximado de las tensiones para reducir la dimensión del problema y, con ello, la complejidad. Esta pérdida de precisión, especialmente notable en los bordes del área de contacto, puede no ser tan relevante como parece, ya que los valores de presión se mantienen similares en el interior del área de contacto.

4.6.4. Gestión de la memoria

La Tabla 11 proporciona un análisis detallado del uso de memoria asociado a cada uno de los algoritmos evaluados, considerando la memoria utilizada, la memoria reservada, el exceso de memoria y la memoria destinada para uso temporal. Es importante destacar que la memoria temporal se refiere a la memoria asignada para las iteraciones del gradiente conjugado o para almacenar matrices y vectores que serán utilizados posteriormente. Además, debe aclararse que el exceso de memoria se obtiene calculando:

$$E_M = \frac{M_r - M_u}{M_r}, \tag{167}$$

donde M_r es la memoria reservada, M_u la memoria usada y E_M el exceso de memoria.

Tabla 11. Gestión de la memoria: Memoria usada, reservada, excesos y memoria temporal.

	Memoria usada (elementos)	**Memoria reservada** (elementos)	**Exceso de memoria** (%)	**Memoria reservada para uso temporal** (elementos)	**Memoria reservada para uso temporal** (%)
NORM	10256081	10260000	0.038%	2561600	24.97%
NORM + CG	7699281	7700000	0.009%	4800	0.06%
NORM + E	15403321	15500000	0.624%	5121600	33.04%
NORM + E + CG	12846521	12900000	0.415%	2564800	19.88%
NORM + P	15403321	15500000	0.624%	5121600	33.04%
NORM + P + CG	12846521	12900000	0.415%	2564800	19.88%
BCCG (2)	2584101	2600000	0.612%	0	0.00%

Según el análisis, el algoritmo NORM emplea 10256081 elementos en memoria, mientras que la cantidad de memoria reservada asciende a 10260000 elementos, lo que resulta en un pequeño exceso del 0.038%. De la memoria total reservada, el 24.97%, equivalente a 2561600 elementos, se destina específicamente al almacenamiento de las matrices $\boldsymbol{A}$ y $\boldsymbol{A_2}$ originales antes de la eliminación gaussiana**.**

El método NORM + CG reduce significativamente el uso de memoria, utilizando 7699281 elementos, con un exceso de memoria mínimo del 0.009%. La memoria reservada para uso temporal, destinada a albergar las direcciones de búsqueda y los residuos obtenidos mediante el CG, es la más baja entre todos los algoritmos que la utilizan. Se emplean temporalmente solo 4800 elementos, lo que representa el 0.06% de la memoria total reservada.

Por otro lado, los algoritmos NORM + E y NORM + P incrementan considerablemente el uso de memoria, empleando 15403321 elementos. Ambos algoritmos reservan 5121600 elementos para uso temporal, lo que representa el 33.04% de la memoria total. Esta memoria se destina a albergar las matrices antes de su modificación por la eliminación de Gauss.

La combinación de NORM + E o NORM + P con el gradiente conjugado reduce el uso de memoria a 12846521 elementos, con un exceso de memoria del 0.415%. La memoria reservada para uso temporal se reduce a 2564800 elementos, equivalente al 19.88% de la memoria total reservada, lo que indica una optimización en la utilización de memoria temporal en comparación con sus variantes sin CG.

Finalmente, el algoritmo BCCG (2) es el que presenta el menor uso de memoria, con 2584101 elementos utilizados y un exceso de memoria del 0.612%. Además, como se apuntaba al inicio de esta sección, no se requiere memoria adicional para uso temporal más allá que la empleada por las iteraciones internas (el gradiente conjugado), lo que lo convierte en el algoritmo más eficiente en términos de manejo de memoria.

En resumen, el análisis de la gestión de memoria revela que, si bien los métodos NORM + E y NORM + P requieren más memoria, especialmente para uso temporal, los métodos que combinan estas aproximaciones con el gradiente conjugado la optimizan. Cabe destacar que esto es debido a que se evita la necesidad de guardar las matrices y vectores originales de los sistemas que se resuelven mediante eliminación gaussiana. El algoritmo BCCG (2) sobresale por su eficiencia en la gestión de memoria, minimizando tanto el uso como el exceso, y eliminando la necesidad de memoria temporal adicional o que no sirva directamente para la resolución.

4.7. Conclusiones

La evaluación comparativa realizada en este capítulo revela importantes conclusiones sobre la eficiencia y precisión de los algoritmos NORM, NORM+E/P y BCCG(K) en el contexto del problema de contacto normal rueda-carril sin rozamiento. Utilizando un entorno controlado y un hardware optimizado, se han ejecutado pruebas exhaustivas para determinar no solo el tiempo de ejecución de cada algoritmo, sino también su manejo de recursos y la precisión de los resultados.

El método con mejor rendimiento computacional es NORM + E, con un tiempo de 0.0602 segundos, seguido de BCCG (2), que registra 0.12031 segundos. Estos métodos son 78.3 y 39,1 veces más rápidos que NORM, respectivamente. NORM + P (g=2) también presenta tiempos competitivos, aunque con menor precisión en comparación con la variante con $g = 4$. Estos métodos permiten una reducción significativa en los tiempos de ejecución, aunque a costa de una ligera disminución en la precisión.

NORM y NORM + CG presentan errores relativos muy bajos, con valores prácticamente insignificantes. En contraste, NORM + E presenta un error relativo respecto a la fuerza normal del 0.0259%, mientras que NORM + P (g=4) muestra un error relativo del 0.0171%. Aunque estos métodos presentan un mayor error en comparación con NORM y NORM + CG, y la distribución de tensiones sea menos precisa en los bordes del área de contacto, la reducción de los tiempos de cálculo los convierte en opciones viables para situaciones donde el tiempo de cálculo es relevante.

Por otro lado, el algoritmo BCCG (2) proporciona un buen balance entre tiempo de ejecución y precisión, siendo competitivo tanto en la gestión del tiempo como en el uso de memoria. Aunque no alcanza la precisión extrema de NORM o NORM + CG, su desempeño es suficientemente óptimo como para considerarlo una opción viable en situaciones donde tanto la eficiencia como la precisión son cruciales.

La gestión de la memoria también desempeña un papel fundamental en la evaluación. Los métodos que integran el gradiente conjugado muestran una notable optimización en el uso de memoria temporal, aunque a costa de una mayor complejidad en la implementación. El algoritmo BCCG (2) nuevamente destaca por su eficiencia, minimizando tanto el uso de memoria como los excesos, lo que lo convierte en una solución eficiente en términos de recursos computacionales.

En conclusión, este estudio comparativo no solo ha permitido identificar las fortalezas y debilidades de cada algoritmo, sino que también proporciona una referencia clara para seleccionar el método más adecuado según las necesidades específicas de precisión y eficiencia en el análisis de problemas de contacto rueda-carril. Los próximos capítulos se centrarán en discutir los resultados obtenidos y, finalmente, en presentar las conclusiones junto con los futuros proyectos que podrían emanar de este trabajo.

5. DISCUSIÓN DE LOS RESULTADOS

Los resultados presentados anteriormente permiten discernir diversos efectos y consecuencias sobre los algoritmos implementados. Como se ha observado, NORM es el método que presenta mayores tiempos de CPU junto con NORM+P(G=4)+CG. Durante las etapas de optimización del código del método NORM, los cuellos de botella identificados, mediante técnicas de *profiling manual*, fueron la generación de la matriz $\boldsymbol{A_2}$ y la resolución mediante la eliminación de Gauss. El primer cuello de botella se solventó mediante la expresión (104) del apartado 3.2.4.3. No obstante, la eliminación de Gauss no se pudo optimizar, conjeturándose que esto se debía al tamaño de la matriz de coeficientes de influencia $\boldsymbol{A}$, de dimensiones 1600×1600. Además, la eliminación gaussiana implica realizar sumas, restas, multiplicaciones, divisiones y operar con la matriz completa. Considerando estos factores y los FLOPs asociados (véase la Tabla 8), parece razonable que el problema radique en el tamaño de la matriz aunque los tiempos de CPU resulten tan elevados.

El comportamiento del método NORM+CG parece respaldar esta hipótesis. El uso del gradiente conjugado reduce significativamente el tiempo de ejecución y el número de FLOPs. De manera similar, el método NORM+E/P reduce el tamaño de la matriz del sistema de ecuaciones, de 1600×1600 a 80×80, o incluso menos, dependiendo de la iteración. Esto acelera la resolución y, como resultado, el gradiente conjugado, que mostró buenos resultados en NORM+CG, parece volverse innecesario.

Según los resultados del *benchmarking*, el método NORM+CG es una buena técnica para agilizar la resolución manteniendo una alta precisión. No obstante, NORM+E/P lo supera en eficiencia, aunque la precisión se vea ligeramente afectada, como se observa en la Tabla 10.

Con respecto al método con mejor rendimiento, NORM+E, Giner Navarro et al. (2023) señala que los coeficientes λ_{0s} de la ecuación (61), asociados al término independiente de la aproximación elíptica, tienden a anularse una vez alcanzada la solución mediante NORM. En la implementación aquí realizada dichos valores disminuyen iteración tras iteración, pero no se anulan. Aun con este pequeño error de implementación, la Figura 25 muestra que el mayor error absoluto es de aproximadamente 70 MPa, concentrado en el borde del área de contacto, y en este trabajo es de 65.518 MPa (véase la Tabla 10). Además, aunque el error relativo de la fuerza normal es de aproximadamente del 0.05% en el artículo original y del 0.09% en este estudio para un grado polinómico $g = 2$, la aproximación elíptica muestra un error relativo del 0.0259%. Si el grado polinómico se incrementa a $g = 4$, el error relativo obtenido, del 0.017%, se ajusta mejor a lo reportado por Giner Navarro et al. (2023) (véanse la Figura 25 y la Figura 26).

A la luz de estos datos, se puede concluir que la implementación realizada en este trabajo no presenta diferencias significativas en comparación con Giner Navarro et al. (2023), y los resultados parecen razonables y comparables con los del artículo (véanse la Tabla 10 y la Figura 26).

Ahora bien, al aumentar el grado polinómico en el análisis, se identificó un aspecto relevante. Cuando el grado polinómico es mayor o igual a 2 en una base como la de la ecuación (65), la matriz resultante de $\boldsymbol{B^T AB}$ presenta entradas extremadamente pequeñas (en algunos casos, del orden de 10^{-46}), algunas de ellas negativas, y parece no ser simétrica a primera vista. Estas características podrían comprometer la aplicabilidad del método del gradiente conjugado y explicar el aumento

significativo de iteraciones y tiempo de cálculo observado en la Tabla 7. De hecho, se detectó que NORM+P(g=3 o 4)+CG no operaba correctamente cuando se trabajaba con una precisión de 64 bits o inferior.

Debido a lo anterior, se realizó un pequeño estudio sobre el proceso de resolución de los métodos con CG. Como se observa en la Tabla 12, el incremento en el número de iteraciones ya es notable al pasar de la base elíptica a la polinómica de grado 2. No obstante, este incremento se vuelve aún más pronunciado cuando el grado polinómico aumenta de $g = 3$ a $g = 4$ (véase la Figura 24). Además, la convergencia para $g \geq 3$ exige un aumento de la precisión del bloque de memoria $\boldsymbol{R}$ para obtener resultados numéricos consistentes. Estos hallazgos podrían indicar una mayor eficiencia de la eliminación gaussiana en la resolución de los algoritmos NORM+E/P, o bien la presencia de un error en la implementación que minora el rendimiento del CG.

Tabla 12. Comparativa de funcionamiento de los métodos que aplican el Gradiente Conjugado (CG). ES y PS señala si la matriz de coeficientes es exactamente simétrica o prácticamente simétrica, respectivamente.

	Matriz	Precisión necesaria	N.º de iteraciones hasta resolver
NORM + CG	$\boldsymbol{A}$, $\boldsymbol{A_2}$	64 bits	338
NORM + E + CG	$\boldsymbol{Q^T AQ}$	64 bits	718
NORM + P(g=2) + CG	$\boldsymbol{B^T AB}$	64 bits	18512
NORM + P(g=3) + CG	$\boldsymbol{B^T AB}$	128 bits	18773
NORM + P(g=4) + CG	$\boldsymbol{B^T AB}$	128 bits	290585

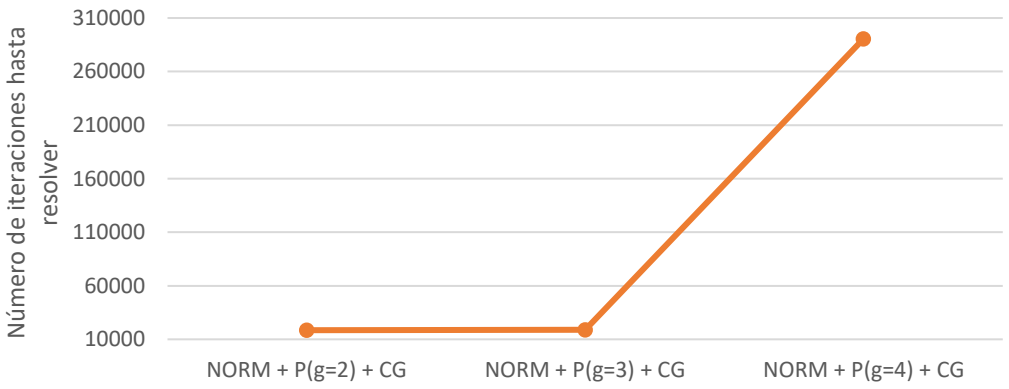

Figura 24. Número de iteraciones hasta resolver el problema normal frente al grado utilizado en la aproximación polinómica del algoritmo NORM+E/P+CG (elaboración propia).

Por su parte, el método BCCG (2) se distingue por su velocidad y precisión, operando en solo 20 iteraciones. Es más rápido que NORM+CG y el error absoluto máximo en la distribución de tensiones asciende únicamente a 1.24 MPa. Vollebregt (2014) presenta una mejora del método del gradiente conjugado con restricciones acotadas a través de la aplicación de precondicionamiento y la optimización de los productos matriz-vector utilizando transformadas de Fourier (FFTs). Según el autor, el método se ve beneficiado debido al mejor condicionamiento de las matrices mediante la llamada Inversa Aproximada (*FFT-based approximate inverse preconditioner*) y a la eficiencia que proporciona el uso de la transformada rápida de Fourier en los productos matriz-vector involucrados

en el algoritmo. Analizar la implementación de estas estrategias sería un interesante proyecto futuro en la misma línea de este trabajo.

A la luz de los resultados, el método más eficiente desde el punto de vista computacional es NORM+E/P, ya que ofrece los menores tiempos de CPU al emplear la base elíptica para representar las tensiones en la dirección $\boldsymbol{X_1}$. Sin embargo, el método BCCG(K) destaca tanto por su eficiencia como por su precisión, mostrando errores absolutos y relativos bajos, además de un tiempo de CPU de solo 0.12 segundos. Ahora bien, si prima la precisión sobre la eficiencia, el método NORM+CG sería la mejor opción. La elección del método depende de las prioridades del problema, ya sea la velocidad o la precisión. CONTACT, por sí solo, introduce un margen de error respecto a la realidad, por lo que es crucial determinar qué nivel de discrepancia es aceptable asumir.

Más allá de la eficiencia y la precisión de los algoritmos, otro factor crucial es la gestión de la memoria. En aplicaciones que manejan grandes volúmenes de datos, una asignación eficiente de la memoria puede marcar la diferencia en el rendimiento del sistema. Las variantes con gradiente conjugado, como NORM+CG, NORM+E/P+CG y BCCG(K), destacan por su bajo consumo de memoria en comparación con otras técnicas. Como se mencionó en el apartado 3.2.4.2.3, el gradiente conjugado no almacena ni modifica directamente la matriz $\boldsymbol{A}$. En su lugar, se realizan productos matriciales entre $\boldsymbol{A}$ y los vectores en cada iteración, lo que reduce significativamente la necesidad de memoria. Este enfoque iterativo evita la creación de matrices temporales o adicionales, ya que el algoritmo solo necesita los vectores de búsqueda, el residuo y otros pocos parámetros para avanzar en cada iteración. En contraste, la eliminación de Gauss modifica directamente la matriz $\boldsymbol{A}$ para triangularla durante el proceso de eliminación. Esto implica que, si es necesario su uso en operaciones posteriores, la matriz $\boldsymbol{A}$ debe ser almacenada, lo que conlleva un mayor consumo de memoria en comparación con el gradiente conjugado, especialmente en problemas de gran tamaño.

Lo anterior se comprueba con el método BCCG(K), que destaca por utilizar menos memoria al resolver el problema de contacto normal entre rueda y carril. Gracias a una gestión eficiente de las matrices, los vectores y las restricciones del problema, se logran soluciones precisas de forma muy eficiente trabajando directamente con la dimensión completa del problema sin necesidad de crear subsistemas de ecuaciones o matrices adicionales.

En el contexto del trabajo, el uso de memoria estática resulta más eficiente que la memoria dinámica, al permitir una planificación previa de los recursos y evitar los costes de gestión durante la ejecución. Sin embargo, una asignación excesiva puede generar ineficiencias, especialmente al operar con un gran volumen de datos, por lo que es esencial ajustar las reservas de memoria de manera precisa.

En este trabajo, la gestión de la memoria estática se ha realizado mediante el bloque de memoria $\boldsymbol{R}$, que agrupa todos los vectores y matrices a emplear. Esto permite un control eficiente de los recursos, evitando la fragmentación y facilitando un acceso más rápido y coherente a los datos. En Fortran, esta técnica es especialmente favorable, ya que el lenguaje optimiza el manejo de arreglos contiguos en memoria, lo que mejora el rendimiento en cálculos intensivos como los de estos algoritmos. Además, el exceso de memoria reservada, entendido como la diferencia entre la memoria asignada y la efectivamente utilizada, se ha mantenido por debajo del 0.624% en todos los casos (véase

la Tabla 11), lo que garantiza una asignación precisa y eficiente, sin comprometer el rendimiento del sistema.

En resumen, los resultados obtenidos no quedan muy allá de los esperados. Todos los aspectos clave del comportamiento de los algoritmos evaluados parecen mostrar coherencia con los objetivos iniciales y las hipótesis planteadas en este proyecto. El método NORM+E/P ha destacado en términos de eficiencia computacional, mientras que el BCCG(K) ha mostrado una excelente combinación de precisión y velocidad, corroborando la adecuación de su implementación para problemas como el contacto rueda-carril. Por otro lado, el método NORM+CG, aunque menos eficiente en comparación, ha demostrado ser la opción más robusta cuando la prioridad es la precisión. A pesar de las pequeñas desviaciones en los errores absolutos y relativos, estos se mantienen dentro de rangos aceptables para las aplicaciones prácticas y se adecuan a los reportados en los artículos de referencia.

En cuanto a la gestión de memoria, la implementación ha demostrado ser eficiente, con un exceso de memoria reservado en niveles mínimos, lo que confirma la viabilidad del uso de memoria estática y su implementación mediante el bloque de memoria $\boldsymbol{R}$. Este enfoque permite determinar previamente la cantidad de memoria necesaria, evitando la sobrecarga de una asignación dinámica durante la ejecución. En conjunto, estos hallazgos sugieren que, aunque ciertos aspectos podrían optimizarse —como la precisión en iteraciones con grados polinómicos más elevados—, el análisis de prestaciones realizado es adecuado y consistente. Esto proporciona una base sólida para futuras investigaciones y mejoras, especialmente en la optimización de los métodos iterativos y en la integración de las ventajas que ofrece cada algoritmo.

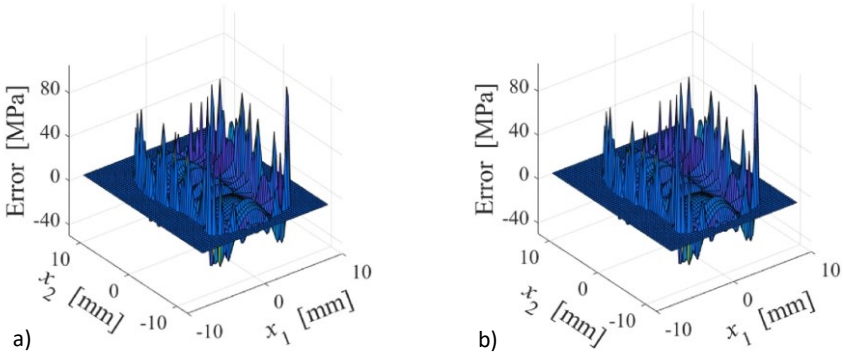

Figura 25. Errores absolutos reportados por Giner Navarro et al. (2023): (a) Aproximación elíptica de las tensiones (para un desplazamiento lateral del eje montado $y = 0$); (b) Aproximación polinómica de las tensiones (para un ángulo nulo de guiñada en el eje montado y $g = 4$).

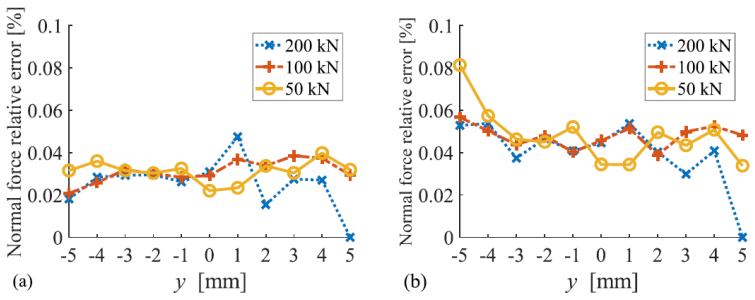

Figura 26. Error relativo de la fuerza normal para diferentes desplazamientos del eje montado según Giner Navarro et al. (2023): (a) Aproximación elíptica; (b) Aproximación polinómica con $g = 4$.

6. CONCLUSIONES Y FUTUROS PROYECTOS

Este trabajo ha brindado al autor la oportunidad de adquirir nuevos conocimientos y adentrarse en una línea de investigación de gran relevancia. El análisis del contacto rueda-carril es esencial, ya que es en este punto donde se generan y transmiten las fuerzas que afectan directamente a la dinámica de los vehículos ferroviarios. La dinámica influye en aspectos fundamentales como la estabilidad, el confort y el desgaste de los componentes, lo que tiene un impacto directo en la seguridad y la eficiencia operativa de todo el sistema ferroviario. Optimizar estos parámetros no solo permite reducir los costos de operación y mantenimiento, sino que también contribuye a mejorar la eficiencia energética. Los avances en esta dirección son clave para desarrollar vehículos e infraestructuras más seguros, duraderos y sostenibles, alineándose con los Objetivos de Desarrollo Sostenible (ODS), en particular el ODS 9 (Industria, Innovación e Infraestructura) y el ODS 11 (Ciudades y Comunidades Sostenibles). Asimismo, la optimización de algoritmos se alinea con el ODS 12 (Producción y Consumo Responsables), ya que promueve una mayor eficiencia en el uso de los recursos.

En este estudio, se ha evaluado de manera exhaustiva la eficacia y eficiencia de diversos algoritmos: una aproximación basada en distribuciones de tensiones polinómicas o elípticas utilizando el método NORM como base, el método del gradiente conjugado con restricciones acotadas y la variante NORM de CONTACT. Este análisis ha permitido identificar las fortalezas y limitaciones de cada enfoque, estableciendo una base sólida para futuras investigaciones.

De los resultados obtenidos se concluye que la combinación de NORM con la aproximación elíptica o polinómica de tensiones, propuesta por Giner Navarro et al. (2023), mejora notablemente la eficiencia computacional de NORM. Este enfoque es 78 veces más rápido que el software CONTACT original y duplica la velocidad del método BCCG(K). En cuanto a precisión, ambos algoritmos ofrecen resultados muy similares a la solución de referencia, aunque el método del gradiente conjugado con restricciones acotadas reduce el error casi a cero. Por otro lado, en el algoritmo NORM+E/P, el error varía según la base utilizada, siendo 0.0259% para distribuciones elípticas y 0.0171% para polinómicas de grado 4.

A la vista de los resultados y la discusión realizada, los algoritmos implementados proporcionan una base sólida para continuar mejorando en esta línea de investigación. La elección del método más adecuado puede ser compleja: los enfoques más rigurosos suelen conllevar mayores tiempos de cálculo, mientras que los más rápidos pueden comprometer la precisión. La selección óptima dependerá del contexto y la aplicación específica. Esto subraya la importancia de realizar análisis exhaustivos, ya que no solo permiten identificar las mejores prácticas, sino también descubrir innovaciones potenciales mediante la optimización y combinación de diferentes técnicas.

La primera idea de proyecto futuro surge del interés de generar un algoritmo híbrido que combine el método BCCG (K) con la aproximación elíptica o polinómica, con el objetivo de mejorar aún más la velocidad de cálculo y presentar un artículo al respecto.

Otro proyecto de gran interés sería la evaluación e implementación de la extensión del método BCCG(K) propuesta por Vollebregt (2014). En su estudio, Vollebregt combina el gradiente conjugado

con restricciones acotadas con un precondicionamiento FAI (*Fourier Approximate Inverse*), logrando mejoras significativas en la convergencia y la eficiencia temporal. Además, propone el uso de la *Fast Fourier Transform* (FFT) para acelerar los productos matriz-vector y optimizar otras operaciones. Un paso crucial sería aplicar estas mejoras en los códigos implementados. Asimismo, sería interesante investigar cómo incorporar el uso de la FFT en los productos matriz-vector o cómo implementar el precondicionamiento para mejorar el rendimiento del gradiente conjugado en los algoritmos existentes. La integración de estos avances podría suponer un progreso significativo en la resolución del problema de contacto rueda-carril.

Futuras investigaciones podrían extender los algoritmos propuestos al contacto normal con rozamiento o extrapolar el benchmarking a algoritmos que aborden tanto el contacto tangencial como ambas variantes, ofreciendo así una evaluación integral de las técnicas de resolución del problema del contacto rueda-carril. Este enfoque es clave para alcanzar una comprensión más completa de la dinámica del vehículo, ya que tanto el rozamiento como las fuerzas tangenciales influyen directamente en la estabilidad del tren, la tracción y el desgaste de los componentes. Incluir estos factores mejoraría la precisión de los estudios y permitiría un análisis más realista del fenómeno.

En resumen, los objetivos propuestos en este trabajo han sido alcanzados de manera satisfactoria. La implementación de los algoritmos en Fortran para el modelado y resolución de la interacción normal entre rueda y carril, así como su evaluación y optimización, han logrado un rendimiento competitivo en términos de eficiencia y precisión. El benchmarking llevado a cabo ha permitido identificar las soluciones más eficientes computacionalmente y señalar áreas con potencial de mejora, lo que contribuye al desarrollo de herramientas computacionales más optimizadas en el ámbito ferroviario. El conocimiento adquirido y las líneas de investigación propuestas abren nuevas oportunidades para avanzar en la precisión y eficacia de las simulaciones. A largo plazo, estos esfuerzos pueden impulsar el desarrollo de técnicas más avanzadas que mejoren la seguridad, la eficiencia y la sostenibilidad del transporte, consolidando el papel clave de la dinámica en la optimización integral del sistema ferroviario.

A título personal, quedo más que satisfecho con el desarrollo de este trabajo y con muchos más aspectos que de él se desprenden. He conseguido aprender mucho más de lo que esperaba, investigar temáticas de gran interés para mí, aplicarlo en el trabajo, en mi vida y hasta afrontar aspectos que en su momento creía imposibles de solucionar. Lo que más me llevo en la mochila, sin embargo, más allá de la gran cantidad de conocimientos que este trabajo ha conseguido asentar en mí, es la gente que ha estado apoyándome hasta el momento. He aprendido a imponerme a mis miedos y a avanzar.

7. BIBLIOGRAFÍA

Baeza, L. (2022). *Problema de contacto rueda-carril.* Universitat Politécnica de Valéncia, Ingeniería Mecánica y de Materiales.

Giner Navarro, J., Gómez Bosch, J., Alonso, A., & Baeza, L. (2023). A fast version of 'CONTACT' for normal problem calculations. *Wear*, 12. doi:https://doi.org/10.1016/j.wear.2023.205074

Golub, G. H., & Van Loan, C. F. (2013). *Matrix Computations.* Baltimore: The Johns Hopkins University Press.

Hager, W. W., & Zhang, H. (1991). A New Active Set Algorithm for Box Constrained Optimization. *SIAM Journal on Optimization*.

Iwnicki, S. (Ed.). (2006). *Handbook of Railway Vehicle Dynamics.* CRC Press.

Kalker, J. J. (1990). *Three-Dimensional Elastic Bodies in Rolling Contact.* Kluwer Academic Publishers, Dordrecht.

Kalker, J. J. (2000). Rolling Contact Phenomena: Linear Elasticity.

Mase, G. T., Mase, G. E., & Smelser, R. E. (2010). *Continuum Mechanics for Engineers.* CRC Press.

Matemáticas Discretas. (22 de Noviembre de 2017). *Medium*. Recuperado el 10 de Agosto de 2024, de Capítulo 3: Punto Flotante: https://medium.com/@matematicasdiscretaslibro/cap%C3%ADtulo-3-punto-flotante-c689043db98b

Moré, J. J., & Toraldo, G. (2006). On the Solution of Large Quadratic Programming Problems with Bound Constraints. *SIAM Journal on Optimization, 17*.

Polak, E., & Ribière, G. (1969). Note sur la convergence de méthodes de directions conjugées. *Revue française d'informatique et de recherche opérationnelle. Série Rouge, 3*, 35-43.

Polonsky, I. A., & Keer, L. M. (1999). A numerical method for solving rough contact problems based on the multi-level multi-summation and conjugate gradient techniques. *Wear, 231*, 206-219.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (1992). *Numerical Reciepes in Fortran77: The Art of Scientific Coumputing* (Segunda ed.). Cambridge University Press.

Profesores del departamento de Matemática Aplicada de la Escuela Técnica Superior de Ingeniería Industrial. (2019). *Apuntes de métodos matemáticos.* Universitat Politècnica de València, Matemática Aplicada.

Sainsot, P., & Lubrecht, A. A. (2011). Efficient solution of the dry contact of rough surfaces: a comparison of fast Fourier transform and multigrid methods. *Journal of Engineering Tribology*.

San-Segundo Cordero, C. (2019). *Estimación del contacto rueda-carril con ANSYS.* Trabajo Fin de Grado, Universidad Carlos III de Madrid, Departamento de Ingeniería Mecánica.

Shewchuk, J. R. (1994). *An Introduction to the Conjugate Gradient Method without the Agonizing Pain.* Carnegie Mellon University, School of Computer Science.

Sichani, M. S. (2016). *On Efficient Modelling of Wheel-Rail Contact in Vehicle Dynamics Simulation.* Tesis Doctoral, KTH Royal Institute of Technology, Stockholm.

Soemantri, S., Puja, W., Budiwantoro, B., Parwata, M., & Schipper, D. J. (2010). Solution to Hertzian Contact Problem between Wheel and Rail for Small Radius of Curvature. *Journal of Solid Mechanics and Materials Engineering*.

Vollebregt, E. A. (13 de Diciembre de 2013). The Bound-Constrained Conjugate Gradient Method for Non-negative Matrices. *Journal of Optimization Theory and Applications, 162*, 931-953.

Vollebregt, E. A. (2014). A new solver for the elastic normal contact problem using conjugate gradients, deflation, and an FFT-based preconditioner. *Journal of Computational Physics*. doi:https://doi.org/10.1016/j.jcp.2013.10.005

PRESUPUESTO

ÍNDICE DEL PRESUPUESTO

1. INTRODUCCIÓN. 111

2. MANO DE OBRA. 111

2.1. Ingeniero Industrial. 111
2.2. Ingenieros Senior. 111
2.3. Coste total de la mano de obra 111

3. SOFTWARE Y EQUIPOS ELECTRÓNICOS. 112

4. RESUMEN DEL PRESUPUESTO. 112

1. INTRODUCCIÓN.

Para la realización de este presupuesto, se concibe el proyecto desarrollado como un trabajo de investigación en el campo de la optimización de algoritmos para la resolución del problema normal de contacto rueda-carril sin rozamiento. Por ello, el objeto del proyecto es no tangible.

Partiendo de lo anterior, el coste inherente a este proyecto está asociado al tiempo invertido por el ingeniero para su realización.

2. MANO DE OBRA.

2.1. Ingeniero Industrial.

Tabla 13. Medición de horas de trabajo realizadas por el Ingeniero industrial.

Tarea	Tiempo (h)
Reuniones presenciales	12
Videollamadas por Teams	10
Programación del modelo NORM y sus variantes	270
Programación del modelo NORM+E/P y sus variantes	50
Programación del modelo BCCG (K)	25
Redacción del proyecto	90
TOTAL	**457h**

2.2. Ingenieros Senior.

Tabla 14. Medición de horas de trabajo realizadas por el Ingeniero Senior.

Tarea	Tiempo (h)
Reuniones presenciales	12
Videollamadas por Teams	10
Revisión de la redacción	7
TOTAL	**17h**

2.3. Coste total de la mano de obra

El coste total del trabajo realizado por la mano de obra se deduce directamente del tiempo trabajado por el Ingeniero Industrial y el Ingeniero Senior. Así, la tabla siguiente presenta el coste total de ese trabajo:

Tabla 15. Coste total de la mano de obra.

Concepto	Precio (€/h)	Rendimiento (h)	Importe (€)
Ingeniero Industrial	19,00 €	457	8.683,00 €
Ingeniero Senior	30,00 €	17	510,00 €
TOTAL			**9.193,00 €**

3. SOFTWARE Y EQUIPOS ELECTRÓNICOS.

Los elementos usados para la elaboración del proyecto son: Un portátil, la licencia de Windows 11 Pro y los entornos de programación de MATLAB y Visual Studio Code. Debe decirse que se tiene en cuenta una vida útil de 4 años (duración del grado) para los entornos de programación y de 5 años para el ordenador y la licencia de Windows (dado que está asociada al ordenador). Además, se concibe que la duración del proyecto ha sido de 6 meses. Estos valores se tienen en cuenta para el cálculo de la amortización de los equipos electrónicos y el software.

Tabla 16. Coste del Software y los equipos electrónicos.

Concepto	Tiempo de uso (meses)	Vida útil esperada (años)	Coste (€)	Coste anual (€/año)	Coste total (€)
PC - Portátil	6	5	1.000,00 €	200,00 €	100,00 €
Windows 11 Pro	6	5	50,00 €	10,00 €	5,00 €
VS Code	6	4	0 €	0 €	0 €
MATLAB Academic	6	4	0 €	262,00 €	131,00 €
TOTAL					**236,00 €**

4. RESUMEN DEL PRESUPUESTO.

Calculados los costes asociados a la mano de obra, los equipos electrónicos y los softwares utilizados, debe elaborarse el presupuesto de ejecución material. Es importante destacar que este es un cálculo aproximado del coste inherente al proyecto, pero no se contemplan gastos fungibles relacionados con su ejecución.

Tabla 17. Presupuesto base de licitación.

Concepto	Coste (€)
Mano de obra	9.193,00 €
Software y equipos electrónicos	236,00 €
Presupuesto de ejecución material	**9.429,00 €**
Gastos generales (13%)	1.225,77 €
Beneficio industrial (6%)	565,74 €
Presupuesto de ejecución por contrata	**11.220,51 €**
21% (I.V.A.)	2.356,31 €
Presupuesto base de licitación	**13.576,82 €**

El coste de este proyecto asciende a un importe total de **TRECE MIL QUINIENTOS SETENTA Y SEIS EUROS CON OCHENTA Y DOS CÉNTIMOS.**

ANEXOS

INDICE DE LOS ANEXOS

ANEXO 1. CÓDIGOS.....115

1.1. Introducción 115
1.2. NORM 115
1.3. NORM+CG..... 132
1.4. NORM+E/P..... 149
1.5. NOMR+E/P+CG 171
1.6. BCCG (K)..... 193

ANEXO 1. CÓDIGOS

1.1. Introducción

Este anexo presenta los códigos fuente desarrollados e implementados en el marco de este Trabajo Fin de Máster (TFM). Los códigos aquí incluidos fueron programados en Fortran y corresponden a los diferentes algoritmos evaluados en términos de rendimiento computacional.

El propósito de este anexo es proporcionar una visión clara y detallada de las implementaciones realizadas durante el estudio. Al incluir estos códigos, se busca garantizar la transparencia del proceso y permitir que otros investigadores puedan replicar los experimentos o adaptar los algoritmos para sus propios estudios. Los códigos están organizados de manera coherente y se acompañan de comentarios que facilitan su comprensión y uso.

1.2. NORM

```
      PROGRAM NORMAL_CONTACT
C     The following code computes the normal traction distribution. Non-Hertzian model.
C     This is the fastest version
C     ========================================================================
C     INPUTS:
C     |    E       : Young modulus [N/m^2]
C     |    NU      : Poisson`s ratio [·]
C     |    N1,N2   : Number of elements in the rolling and lateral directions
C     |    H       : Undeformed distance matrix N1xN2 [m]
C     |    X1      : Longitudinal coordinates of the elements in the PCA [m]
C     |    X2      : Lateral coordinates of the elements in the PCA [m]
C     |    A       : Half size of the element - rolling direction [m]
C     |    B       : Half size of the element - lateral direction [m]
C     OUTPUTS:
C     |    P     : Normal traction N1xN2 matrix [N/m^2]
C     |    NF      : Total normal force [N]
C     |    X1P,X2P : Coordinates of the centre of pressure [m]
C     ========================================================================
C     PCA Diagram:
C                     (Lateral)
C                     A X2
C     ---------------|------------------
C     |  |  |  |  |  |  |  |  |  |  |44|
C     ----------------------------------
C     |  |  |  |  |  |  |  |  |  |  |  |
C     ---------------------------------------> X1 (Rolling)
C     |12|  |  |  |  |  |  |  |  |  |  |
```

```
C     ---------------------------------
C     | 1| 2| 3| 4| 5| 6| 7| 8| 9|10|11|
C     ---------------------------------
C
C     ========================================================================
        REAL TIME_ARRAY(2)
        double precision E, NU, A, B, NF, X1P, X2P, TOL, TOL2
        DOUBLE PRECISION CPU_TIME_USED,CPU_TIME_USED1, ERR
        INTEGER N1, N2, NE, NESQR, RUN, JEC, TT, IX1, IX2
        INTEGER IRC, IH, IAES, IT, IA, IPD, IH2
        INTEGER IA2, IX, IP, ITEMP, IEC, IEND
        INTEGER MAX_ITER_NUM,IVEC
        DOUBLE PRECISION PI, R(10260000)
        PARAMETER (PI = 3.141592653589793D0)
        DOUBLE PRECISION TIME_START2, TIME_END2
        INTEGER start_count, end_count, count_rate, count_max
        DOUBLE PRECISION WALLCLOCK

        TOL = 1.0E-9
        TOL2 = 1.0E-13
        MAX_ITER_NUM=20
        PRINT *, "//========================================//"
        PRINT *, "//=========== NORM_V16_GE.F CODE ==========//"
        PRINT *, "//========================================//"
        PRINT *, "//==============CLASSIC NORM===============//"

       CALL READ_SCALAR_VARIABLES(E, NU, A, B, N1, N2)

        NE = N1 * N2
        NESQR = NE ** 2
        IH = 1
        IX1 = IH + NE
        IX2 = IX1 + N1
        IRC = IX2 + N2
        IAES = IRC + NE * 2
        IT = IAES + NE
        IA = IT + NESQR
        IPD = IA + NESQR
        IH2 = IPD + NE
        IA2 = IH2 + NE
        IX = IA2 + NESQR
```

```
        IP = IX + NE
        ITEMP = IP + NE
        IVEC=ITEMP+NESQR
        IEC=IVEC+NE
        IEND = IEC + NE

        PRINT*, 'Memory used:',IEND, 'elements'
        IF (IEND .LT. 10260000) THEN
          WRITE (*,*) 'Correct Allocation'
        ELSE
          WRITE (*,*) 'IEND is greater than maximum capacity for R.'
        END IF
        CALL ZEROS_VEC(IEND,R)
        CALL READ_VECTOR_AND_MATRIX(N1, N2, R(IH), R(IX1), R(IX2))
c       ¡CODE TIMMER STARTS!
        TIME_START2 = ETIME(TIME_ARRAY)
        CALL SYSTEM_CLOCK(COUNT_RATE=count_rate, COUNT_MAX=count_max)
        CALL SYSTEM_CLOCK(COUNT=start_count)
C       AES CREATION
        CALL CREATION_AES(N1, N2, A, B, NU, E, R(IAES), R(IX1), R(IX2))
C       MATRIX ROW-COLUMN CREATION
        CALL RC_CALC(N1, N2, NE, R(IRC))
C       A MATRIX CREATION
        CALL A_FORMAT(NE, N1, N2, R(IA), R(IAES), R(IRC))
C       CONJUGATE GRADIENT METHOD
        CALL GE_PIVOT(NE, R(IA), R(IH), R(IPD),R(ITEMP),
     & R(IVEC))
        RUN = 1
        TT = 0
C       NEW PD AND PCA CALCULATION LOOP
  10  CONTINUE
        IF (RUN .EQ. 0) GOTO 20
        RUN = 0
        IF (TT .EQ. 1) THEN
          CALL ZEROS_MAT(NE,JEC, R(IT))
          CALL ZEROS_VEC(JEC, R(IX))
          CALL ZEROS_MAT(JEC,JEC, R(IA2))
          CALL ZEROS_VEC(JEC, R(IH2))
          CALL ZEROS_MAT(JEC,JEC, R(ITEMP))
          CALL ZEROS_VEC(JEC, R(IVEC))
        END IF
```

```
        TT = 1
        CALL JEC_VALUE(TOL, RUN, JEC, NE, R(IT), R(IPD), R(IEC))
        CALL ZEROS_VEC(NE, R(IPD))
        CALL MULTIPLY_MAT_VEC2(NE, JEC, R(IH2), R(IH), R(IT))
        CALL MAKEA2(JEC, NE, R(IA), R(IA2),R(IEC))
        CALL GE_PIVOT(JEC, R(IA2), R(IH2), R(IX),R(ITEMP),
     & R(IVEC))
        CALL MULTIPLY_MAT_VEC(NE, JEC, R(IPD), R(IX), R(IT))
        IF (RUN .EQ. 1) GOTO 10
  20  CONTINUE
        NF = 0.0
        X1P = 0.0
        X2P = 0.0
        CALL FORCE_CALC(N1, N2, R(IPD), R(IP), R(IX1), R(IX2), X1P,
     &      X2P, NF, A, B)
        ERR = (ABS(NF - 5.003728385561319D+4)/5.003728385561319D+4)*
     & 100.0D0
        TIME_END2 = ETIME(TIME_ARRAY)
        CALL SYSTEM_CLOCK(COUNT=end_count)
c       CALCULATE WALL-CLOCK TIME AND CPUT TIME
        WALLCLOCK = REAL(end_count - start_count) / REAL(count_rate)
        CPU_TIME_USED = TIME_END2 - TIME_START2
        write ( *, * ) 'Wall-clock time: ', WALLCLOCK, 'seconds'
        write ( *, * ) 'CPU time used: ', CPU_TIME_USED ,
     & 'seconds'
        write ( *, * ) "//=======================================//"
        write ( *, * ) "Total Normal Force (N) value is:", NF, "N"
        write ( *, * ) "//=======================================//"
        write ( *, * ) "Relative error ",
     & "(NORM (Fortran) vs. NORM (Matlab))"
        write ( *, * )  ERR, "%"
        write ( *, * ) "//=======================================//"
        CALL compare_matrices(R(IP), R(IH), N1, N2)
        CALL SYSTEM('PAUSE')
      END

      SUBROUTINE READ_SCALAR_VARIABLES(E, NU, A, B, N1, N2)
C     ========================================================================
C     READING OF SCALAR VARIABLES` VALUES FROM FILES
C     ========================================================================
C     INPUTS (which are outputs itself):
```

```
C     |   E         : Young modulus [N/m^2]
C     |   nu        : Poisson`s ratio [·]
C     |   N1,N2     : Number of elements in the rolling and lateral directions
C     |   a         : Half size of the element - rolling direction [m]
C     |   b         : Half size of the element - lateral direction [m]
C     ======================================================================
        DOUBLE PRECISION E, NU, A, B
        INTEGER N1, N2
        INTEGER IERR

        OPEN(UNIT=1, FILE='E.txt', STATUS='OLD', IOSTAT=IERR)
        IF (IERR .ne. 0) STOP "Error opening E.txt"
        READ(1, *) E
        CLOSE(1)
        OPEN(UNIT=2, FILE='nu.txt', STATUS='OLD', IOSTAT=IERR)
        IF (IERR .ne. 0) STOP "Error opening nu.txt"
        READ(2, *) NU
        CLOSE(2)


        OPEN(UNIT=3, FILE='a.txt', STATUS='OLD', IOSTAT=IERR)
        IF (IERR .ne. 0) STOP "Error opening a.txt"
        READ(3, *) A
        CLOSE(3)
        OPEN(UNIT=4, FILE='b.txt', STATUS='OLD', IOSTAT=IERR)
        IF (IERR .ne. 0) STOP "Error opening b.txt"
        READ(4, *) B
        CLOSE(4)
        OPEN(UNIT=5, FILE='N1.txt', STATUS='OLD', IOSTAT=IERR)
        IF (IERR .ne. 0) STOP "Error opening N1.txt"
        READ(5, *) N1
        CLOSE(5)
        OPEN (UNIT=20, FILE='N2.txt', STATUS='OLD', IOSTAT=IERR)
        IF (IERR .ne. 0) STOP "Error opening N1.txt"
        READ(20, *) N2
        CLOSE(20)
      END

      SUBROUTINE READ_VECTOR_AND_MATRIX(N1, N2, H, X1, X2)
C     ======================================================================
C     READING OF VECTORS AND MATRICES VALUES FROM FILES
```

```
C     ======================================================================
C     INPUTS (which are outputs itself):
C     |   N1,N2     : Number of elements in the rolling and lateral directions
C     |   H         : Undeformed distance matrix N1xN2 [m]
C     |   x1        : Longitudinal coordinates of the elements in the PCA [m]
C     |   x2        : Lateral coordinates of the elements in the PCA [m]
C     ======================================================================
        INTEGER N1, N2, I, J, IERR
        DOUBLE PRECISION H(N1, N2), X1(N1), X2(N2)

        ! Read vector X1 from x1.txt
        OPEN(UNIT=7, FILE='x1.txt', STATUS='OLD', IOSTAT=IERR)
        IF (IERR /= 0) STOP "Error opening x1.txt"
        DO I = 1, N1
            READ(7, *) X1(I)
        END DO
        CLOSE(7)
        OPEN(UNIT=8, FILE='x2.txt', STATUS='OLD', IOSTAT=IERR)
        IF (IERR /= 0) STOP "Error opening x2.txt"
        DO I = 1, N2
            READ(8, *) X2(I)
        END DO
        CLOSE(8)
        OPEN(UNIT=9, FILE='h.txt', STATUS='OLD', IOSTAT=IERR)
        IF (IERR /= 0) THEN
            PRINT *, 'Error opening h.txt'
            STOP
        END IF
        DO I = 1, N1
            READ(9, *, IOSTAT=IERR) (H(I, J), J = 1, N2)
            IF (IERR /= 0) THEN
                PRINT *, 'Error reading h.txt at row', I
                CLOSE(9)
                STOP
            END IF
        END DO
        CLOSE(9)
      END

      SUBROUTINE CREATION_AES(N1, N2, A, B, NU, E, AES, X1, X2)
C     ======================================================================
```

```
C     Aes CREATION
C     ============================================================================
C     INPUTS:
C     |    N1,N2   : Number of elements in the rolling and lateral directions
C     |    a       : Half size of the element - rolling direction [m]
C     |    b       : Half size of the element - lateral direction [m]
C     |    nu      : Poisson`s ratio [·]
C     |    E       : Young modulus [N/m^2]
C     |    F       : Flexibility coefficient
C     |    Aes     : Influence coefficient`s matrix for each element [N1xN2]
C     |    x1      : Longitudinal coordinates of the elements in the PCA [m]
C     |    x2      : Lateral coordinates of the elements in the PCA [m]
C     ============================================================================
C     WHAT IS BEING CALCULATED?:
C     |    Aes     : Influence coefficient`s matrix for each element [N1xN2]
C     ============================================================================
      INTEGER N1, N2, I, J
      DOUBLE PRECISION A, B, NU, E, F, X, Y, AUX
      DOUBLE PRECISION AES(N1, N2), X1(N1), X2(N2)
      DOUBLE PRECISION PI

      PI = ACOS(-1.0)
      AUX = 2.0 * ((1.0 - NU**2) / (PI * E))
      DO I = 1, N1
        X = X1(I) - X1(1)
        DO J = 1, N2
            Y = X2(J) - X2(1)
            CALL CT_BC(X, Y, A, B, F)
            AES(I, J) = F * AUX
        END DO
      END DO
      END

      SUBROUTINE CT_BC(X, Y, A, B, F)
C     ============================================================================
C     FLEXIBILITY COEFFICIENTS CALCULATION
C     ============================================================================
C     Calculate the flexibility coeff. A, which relates the displacement
C     at (x,y) due to an applied unitary load on a 2ax2b origin-centered
C     rectangle. Love (1929), Johnson page 54
C     Note: Multiply by (1-poisson^2)/pi/E
```

```
C     ======================================================================
C     INPUTS:
C     |    x      : x-coord where displacement is measured [m]
C     |    y      : y-coord where displacement is measured [m]
C     |    a      : Half size of the element - rolling direction [m]
C     |    b      : Half size of the element - lateral direction [m]
C     ======================================================================
C     OUTPUT:
C     |    F      : Flexibility coefficient
C     ======================================================================
        double precision X, Y, A, B, F

       F = (X + A) * LOG(((Y + B) + SQRT((Y + B)**2 + (X + A)**2)) /
     & ((Y - B) + SQRT((Y - B)**2 + (X + A)**2))) +
     &   (Y + B) * LOG(((X + A) + SQRT((Y + B)**2 + (X + A)**2)) /
     & ((X - A) + SQRT((Y + B)**2 + (X - A)**2))) +
     &   (X - A) * LOG(((Y - B) + SQRT((Y - B)**2 + (X - A)**2)) /
     & ((Y + B) + SQRT((Y + B)**2 + (X - A)**2))) +
     &   (Y - B) * LOG(((X - A) + SQRT((Y - B)**2 + (X - A)**2)) /
     & ((X + A) + SQRT((Y - B)**2 + (X + A)**2)))
       END

       SUBROUTINE RC_CALC(N1, N2, NE, RC)
C     ======================================================================
C     ROW - COLUMN MATRIX
C     ======================================================================
C     INPUTS:
C     |    N1     : Number of elements in the rolling direction
C     |    N2     : Number of elements in the lateral direction
C     |    NE     : Dimension of the problem
C     |    RC     : Matrix that stores the position of PCA elements [NEx2]
C     ======================================================================
C     WHAT IS BEING CALCULATED?:
C     |    RC     : Matrix that stores the position of PCA elements [NEx2]
C     ======================================================================
        INTEGER N1, N2, NE, I, J, V
        DOUBLE PRECISION RC(NE, 2)

        V = 1
        DO I = 1, N2
          DO J = 1, N1
```

```
            IF (V .LE. NE) THEN
              RC(V, 1) = J
              RC(V, 2) = I
              V = V + 1
            END IF
          END DO
        END DO
      END

      SUBROUTINE A_FORMAT(NE, N1, N2, A, AES, RC)
C     ================================================================================
C     CREATION OF MATRIX A
C     ================================================================================
C     INPUTS:
C     |    NE      : Dimension of the problem
C     |    N2      : Number of elements in the lateral direction
C     |    A       : Matrix containing the influence between all elements of the PCA [NExNE]
C     |    Aes     : Influence coefficient`s matrix for each element [N1xN2]
C     |    RC      : Matrix that stores the position of PCA elements [NEx2]
C     ================================================================================
C     WHAT IS BEING CALCULATED?:
C     |    A       : Matrix containing the influence between all elements of the PCA [NExNE]
C     ================================================================================
        INTEGER NE, N2, I, J, Ro, Co
        DOUBLE PRECISION A(NE, NE), AES(N1, N2), RC(NE, 2)

        DO I = 1, NE
          DO J = 1, NE
            Ro = ABS(RC(J, 1) - RC(I, 1))+1
            Co = ABS(RC(J, 2) - RC(I, 2))+1
            A(I, J) = AES(Ro, Co)
          END DO
        END DO
      END

      SUBROUTINE GE_PIVOT(M, A, H, P, TEMP, VEC)
c     //=============================================================================
c     //  | GAUSS ELIMINATION WITH PARTIAL PIVOTING - OPTIMIZED
c     //=============================================================================
c     //  | This subroutine solves the linear system A * P = H using the Gauss
c     //  | Elimination method with partial pivoting, followed by back substitution.
```

```
c     //  | The code has been optimized to reduce execution time and improve
c     //  | numerical stability.
c     //  | INPUTS:
c     //  |    M        : Dimension of the problem (size of the matrix)
c     //  |    A        : Matrix containing the coefficients of the system
c     //  |    H        : Vector representing the right-hand side of the system
c     //  |    TEMP     : Temporary matrix used to store a copy of A
c     //  |    VEC      : Temporary vector used to store a copy of H
c     //  | OUTPUTS:
c     //  |    P        : Solution vector of the system
c     //=====================================================================
      INTEGER M, I, J, K, MAX_ROW
      DOUBLE PRECISION A(M, M), H(M), P(M)
      DOUBLE PRECISION TEMP(M,M), VEC(M)
      DOUBLE PRECISION FACTOR, MAX_VAL, TT

c     Copy A and H into TEMP and VEC to preserve the original matrices
      VEC = H
      TEMP = A
c     Forward Elimination with Partial Pivoting
      DO K = 1, M-1
c         Find the row with the largest pivot element in column K
          MAX_ROW = K
          MAX_VAL = ABS(A(K, K))
          DO I = K+1, M
              IF (ABS(A(I, K)) .GT. MAX_VAL) THEN
                  MAX_ROW = I
                  MAX_VAL = ABS(A(I, K))
              END IF
          END DO
c         Swap rows in matrix A and vector H if necessary
          IF (MAX_ROW .NE. K) THEN
              DO J = 1, M
                  TT = A(K, J)
                  A(K, J) = A(MAX_ROW, J)
                  A(MAX_ROW, J) = TT
              END DO
              TT = H(K)
              H(K) = H(MAX_ROW)
              H(MAX_ROW) = TT
          END IF
```

```
c         Perform elimination to convert A into an upper triangular matrix
          DO I = K+1, M
              FACTOR = A(I, K) / A(K, K)
              DO J = K+1, M
                  A(I, J) = A(I, J) - FACTOR * A(K, J)
              END DO
              H(I) = H(I) - FACTOR * H(K)
          END DO
      END DO
c     Back Substitution to find the solution vector P
      P(M) = H(M) / A(M, M)
      DO I = M-1, 1, -1
          P(I) = H(I)
          DO J = I+1, M
              P(I) = P(I) - A(I, J) * P(J)
          END DO
          P(I) = P(I) / A(I, I)
      END DO

c     Restore the original matrices A and H
      A = TEMP
      H = VEC
      RETURN
      END

      SUBROUTINE ZEROS_VEC(SIZE, v)
C     ================================================================================
C     INITIALIZATION OF VECTORS
C     ================================================================================
C     INPUTS:
C     |    size    : Dimension of the problem
C     |    v       : Vector to be initialized
C     ================================================================================
C     WHAT IS BEING CALCULATED?:
C     |    v       : Vector initialized
C     ================================================================================
        INTEGER SIZE, I
        DOUBLE PRECISION v(SIZE)

        DO I = 1, SIZE
          v(I) = 0.0
```

```
        END DO
      END

      SUBROUTINE ZEROS_MAT(SIZE1,SIZE2, MAT)
C     ============================================================================
C     INITIALIZATION OF MATRICES
C     ============================================================================
C     INPUTS:
C     |    size1   : Rows of the matrix
C     |    size2   : Columns of the matrix
C     |    MAT     : Matrix to be initialized
C     ============================================================================
C     WHAT IS BEING CALCULATED?:
C     |    MAT     : Matrix initialized
C     ============================================================================
        INTEGER SIZE1, SIZE2, J, I
        DOUBLE PRECISION MAT(SIZE1,SIZE2)

        DO I = 1, SIZE1
          DO J=1, SIZE2
            MAT(I,J) = 0.0
          END DO
        END DO
      END

      SUBROUTINE JEC_VALUE(TOL, RUN, JEC, NE, T, PD, EC)
C     ============================================================================
C     JEC`S VALUE CALCULATION
C     ============================================================================
C     INPUTS:
C      |     TOL      : Tolerance to differentiate elements in contact against those not in
contact
C     |    RUN     : Logical value (0: The real contact area has been found. 1: The contact
area must be re-calulated)
C     |    JEC     : Number of elements in contact in the real contact area
C     |    NE      : Total dimension of the problem
C      |     T        : Matrix that stores (with unitary value) the positions to be used in
matrices and vectors to calculate the new values after updating JEC
C     |    PD      : Normal traction vector NEx1 [N/m^2]
C     |    EC      : Vector that stores the position in Influence coefficients` matrix A to
be considered based on PD`s value.
C     ============================================================================
```

```
C     WHAT IS BEING CALCULATED?:
C     |    RUN     : Logical value (0: The real contact area has been found. 1: The contact area must be re-calulated)
C     |    JEC     : Number of elements in contact in the real contact area
C     |    T       : Matrix that stores (with unitary value) the positions to be used in matrices and vectors to calculate the new values after updating JEC
C     |    EC      : Vector that stores the position in Influence coefficients` matrix A to be considered based on PD`s value.
C     ========================================================================
        INTEGER RUN, JEC, NE, N
        DOUBLE PRECISION TOL, T(NE,NE), PD(NE), EC(NE)

        JEC=0
        DO N = 1, NE
          IF (PD(N) .LT. TOL) THEN
              IF ((PD(N) .LT. (-TOL)) .AND.
     & (RUN .EQ. 0)) THEN
                RUN = 1
              END IF
          ELSE
                  JEC=JEC + 1
                  T(N,JEC)= 1.0
                  EC(JEC)=N
          END IF
        END DO
      END

      SUBROUTINE MULTIPLY_MAT_VEC(R, C, RE, V, A)
C     ========================================================================
C     MULTIPLY A VECTOR AND A MATRIX (RE=A*V)
C     ========================================================================
C     INPUTS:
C     |    c       : Columns
C     |    r       : Rows
C     |    Re      : Result of the multiplication
C     |    v       : vector to multiply
C     |    A       : matrix to multiply
C     ========================================================================
C     WHAT IS BEING CALCULATED?:
C     |    Re      : Result of the multiplication
C     ========================================================================
        INTEGER C, R, I, J
```

```
        DOUBLE PRECISION RE(R), V(C), A(R, C)

        DO I = 1, R
          DO J = 1, C
            RE(I) = RE(I) + A(I, J) * V(J)
          END DO
        END DO
      END

      SUBROUTINE MULTIPLY_MAT_VEC2(R, C, RE, V, A)
C     ================================================================================
C     MULTIPLY A VECTOR AND A MATRIX (RE=A^T*V)
C     ================================================================================
C     INPUTS:
C     |    c      : Columns
C     |    r      : Rows
C     |    Re     : Result of the multiplication
C     |    v      : vector to multiply
C     |    A      : matrix to multiply
C     ================================================================================
C     WHAT IS BEING CALCULATED?:
C     |    Re     : Result of the multiplication
C     ================================================================================
        INTEGER C, R, I, J
        DOUBLE PRECISION RE(C), V(R), A(R, C)

        DO I = 1, R
          DO J = 1, C
            RE(J) = RE(J) + A(I, J) * V(I)
          END DO
        END DO
      END

      SUBROUTINE MAKEA2(JEC, NE, A, A2,EC)
C     ================================================================================
C     CALCULATION OF THE A2 MATRIX
C     ================================================================================
C     INPUTS:
C     |    jec     : New dimension of the problem (after tension value evaluation)
C     |    NE      : Dimension of the problem
C     |    T       : Matrix containing the positions of A to consider (after tension value
evaluation) [NExNE]
```

```
C     |    A       : Matrix containing the influence coefficients between all elements of the
PCA [NExNE]
C     |    A2      : Matrix containing the influence coefficients between all elements of the
updated PCA (after tension value evaluation) [jec x jec]
C     |    EC      : Vector that stores the position in Influence coefficients` matrix A to
be considered in A2 based on PD`s value. (Elements in contact area)
C     ========================================================================
C     WHAT IS BEING CALCULATED?:
C     |    A2      : Matrix containing the influence between all elements of the updated PCA
(after tension value evaluation) [jec x jec]
C     ========================================================================
        INTEGER JEC, NE, I, J
        DOUBLE PRECISION A(NE, NE), A2(JEC, JEC)
        double precision EC(NE)

        DO J =1,JEC
          DO I=1,JEC
            A2(I,J)=A(INT(EC(I)),INT(EC(J)))
          END DO
        END DO
      END

      SUBROUTINE FORCE_CALC(N1, N2, PD, P, X1, X2, X1P,
     & X2P, NF, A, B)
C     ========================================================================
C     CALCULATION OF THE TOTAL FORCE
C     ========================================================================
C     INPUTS:
C     |    N1,N2   : Number of elements in the rolling and lateral directions
C     |    PD      : Normal traction vector NEx1 [N/m^2]
C     |    P       : Normal traction N1xN2 matrix [N/m^2]
C     |    X1      : Longitudinal coordinates of the elements in the PCA [m]
C     |    X2      : Lateral coordinates of the elements in the PCA [m]
C     |    X1P,X2P : Coordinates of the centre of pressure [m]
C     |    NF      : Total normal force [N]
C     |    A       : Half size of the element - rolling direction [m]
C     |    B       : Half size of the element - lateral direction [m]
C     ========================================================================
C     WHAT IS BEING CALCULATED?:
C     |    P       : Normal traction N1xN2 matrix [N/m^2]
C     |    NF      : Total normal force [N]
C     |    X1P,X2P : Coordinates of the centre of pressure [m]
```

```
C     ======================================================================
        INTEGER N1, N2, I, J, M, N
        DOUBLE PRECISION PD(NE), P(N1, N2)
        DOUBLE PRECISION X1(N1), X2(N2),SIGMA
        DOUBLE PRECISION X1P, X2P, SUMA, NF, A, B

        SIGMA=0.0
        DO N = 1, N2
          DO M = 1, N1
            P(M, N) = PD((N - 1) * N1 + M)
          END DO
        END DO
C Addition of all elements in pa to calculate the total traction
        DO N=1,N2
          DO M=1,N1
            SIGMA=SIGMA+P(M,N)
          END DO
        END DO
C Calculate X1P
        SUMA = 0.0
        DO I = 1, N1
          DO J = 1, N2
              SUMA = SUMA + P(I, J)
          END DO
            X1P = X1P + SUMA * X1(I)
        END DO
        X1P = X1P / SIGMA
C Calculate X2P
        SUMA = 0.0
        DO I = 1, N2
          DO J = 1, N1
              SUMA = SUMA + P(J, I)
          END DO
              X2P = X2P + SUMA * X2(I)
        END DO
        X2P = X2P / SIGMA
C Calculate NF
      NF = SIGMA * 4 * A * B
      END

      SUBROUTINE COMPARE_MATRICES(MATRIX, REF_MATRIX, N1, N2)
```

```
C     ======================================================================
C     COMPARE TWO MATRICES AND CALCULATE ERRORS
C     ======================================================================
C     INPUTS:
C     |    MATRIX        : Matrix to be compared [n1 x n2]
C     |    REF_MATRIX    : Reference matrix to compare against [n1 x n2]
C     |    N1, N2        : Dimensions of the matrices
C     OUTPUTS:
C     |    MAX_ABS_ERROR : Maximum absolute error between the two matrices
C     |    MAX_REL_ERROR : Maximum relative error between the two matrices
C     FUNCTIONALITY:
C     |    Reads the reference matrix from a file (REF.TXT), compares it
C     |    element by element with the provided matrix, and computes the
C     |    maximum absolute and relative errors.
C     ======================================================================
      integer n1, n2, i, j
      DOUBLE PRECISION matrix(n1, n2), ref_matrix(n1, n2)
      DOUBLE PRECISION max_abs_error, max_rel_error, abs_error, rel_error

      max_abs_error = 0.0d0
      max_rel_error = 0.0d0
C     Read the reference matrix from the file REF.TXT
      open(unit=10, file='ref.txt', status='old')
      do i = 1, n1
         read(10,*) (ref_matrix(i, j), j = 1, n2)
      enddo
      close(10)
C     Compare the matrices
      do i = 1, n1
         do j = 1, n2
            abs_error = abs(matrix(i,j) - ref_matrix(i,j))
            if (ref_matrix(i,j) .ne. 0.0d0) then
               rel_error = abs_error / abs(ref_matrix(i,j))
            else
               rel_error = 0.0d0
            endif
            if (abs_error > max_abs_error) max_abs_error = abs_error
            if (rel_error > max_rel_error) max_rel_error = rel_error
         enddo
      enddo
```

```
C       Print the maximum absolute and relative errors
        print*, 'Max absolute error:', max_abs_error/1d+6
        print*, 'Max relative error:', max_rel_error
        return
      END
```

1.3. NORM+CG

```
      PROGRAM NORMAL_CONTACT
C     The following code computes the normal traction distribution. Non-Hertzian model.
C     This is the fastest version
C     ============================================================================
C     INPUTS:
C     |    E       : Young modulus [N/m^2]
C     |    NU      : Poisson`s ratio [·]
C     |    N1,N2   : Number of elements in the rolling and lateral directions
C     |    H       : Undeformed distance matrix N1xN2 [m]
C     |    X1      : Longitudinal coordinates of the elements in the PCA [m]
C     |    X2      : Lateral coordinates of the elements in the PCA [m]
C     |    A       : Half size of the element - rolling direction [m]
C     |    B       : Half size of the element - lateral direction [m]
C     OUTPUTS:
C     |    P     : Normal traction N1xN2 matrix [N/m^2]
C     |    NF      : Total normal force [N]
C     |    X1P,X2P : Coordinates of the centre of pressure [m]
C     ============================================================================
C     PCA Diagram:
C                     (Lateral)
C                     A X2
C     ----------------|------------------
C     |  |  |  |  |  |  |  |  |  |  |44|
C     ----------------------------------
C     |  |  |  |  |  |  |  |  |  |  |  |
C     -------------------------------------> X1 (Rolling)
C     |12|  |  |  |  |  |  |  |  |  |  |
C     ----------------------------------
C     | 1| 2| 3| 4| 5| 6| 7| 8| 9|10|11|
C     ----------------------------------
C     ============================================================================
        REAL TIME_ARRAY(2)
        double precision E, NU, A, B, NF, X1P, X2P, TOL, TOL2
        DOUBLE PRECISION CPU_TIME_USED,CPU_TIME_USED1, ERR
```

```
      INTEGER N1, N2, NE, NESQR, RUN, JEC, TT, IX1, IX2
      INTEGER IRC, IH, IAES, IT, IA, IPD, IH2,ITER
      INTEGER IA2, IX, IP, IR, IVK, IAVK, IEC, IEND
      INTEGER MAX_ITER_NUM
      DOUBLE PRECISION PI, R(7700000)
      PARAMETER (PI = 3.141592653589793D0)
      DOUBLE PRECISION TIME_START2, TIME_END2
      INTEGER start_count, end_count, count_rate, count_max
      DOUBLE PRECISION WALLCLOCK

      TOL = 1.0E-9
      TOL2 = 1.0E-13
      MAX_ITER_NUM=20
      PRINT *, "//========================================//"
      PRINT *, "//=========== NORM_V16_CG.F CODE ==========//"
      PRINT *, "//========================================//"
      PRINT *, "//================NORM + CG===============//"

     CALL READ_SCALAR_VARIABLES(E, NU, A, B, N1, N2)

      NE = N1 * N2
      NESQR = NE ** 2
      IH = 1
      IX1 = IH + NE
      IX2 = IX1 + N1
      IRC = IX2 + N2
      IAES = IRC + NE * 2
      IT = IAES + NE
      IA = IT + NESQR
      IPD = IA + NESQR
      IH2 = IPD + NE
      IA2 = IH2 + NE
      IX = IA2 + NESQR
      IP = IX + NE
      IR = IP + NE
      IVK = IR + NE
      IAVK = IVK + NE
      IEC=IAVK+NE
      IEND = IEC + NE

      PRINT*, 'Memory used:',IEND, 'elements'
```

```
        IF (IEND .LT. 7700000) THEN
          WRITE (*,*) 'Correct Allocation'
        ELSE
          WRITE (*,*) 'IEND is greater than maximum capacity for R.'
        END IF
        CALL READ_VECTOR_AND_MATRIX(N1, N2, R(IH), R(IX1), R(IX2))
c       ¡CODE TIMMER STARTS!
        TIME_START2 = ETIME(TIME_ARRAY)
        CALL SYSTEM_CLOCK(COUNT_RATE=count_rate, COUNT_MAX=count_max)
        CALL SYSTEM_CLOCK(COUNT=start_count)
C       AES CREATION
        CALL CREATION_AES(N1, N2, A, B, NU, E, R(IAES), R(IX1), R(IX2))
C       MATRIX ROW-COLUMN CREATION
        CALL RC_CALC(N1, N2, NE, R(IRC))
C       A MATRIX CREATION
        CALL A_FORMAT(NE, N1, N2, R(IA), R(IAES), R(IRC))
C       CONJUGATE GRADIENT METHOD
        CALL CGR_METHOD(NE, R(IA), R(IH), R(IPD), R(IR), R(IVK),
     &                  R(IAVK), TOL2, MAX_ITER_NUM)
        RUN = 1
        TT = 0
C       NEW PD AND PCA CALCULATION LOOP
  10  CONTINUE
        IF (RUN .EQ. 0) GOTO 20
        RUN = 0
        IF (TT .EQ. 1) THEN
          CALL ZEROS_MAT(NE,JEC, R(IT))
          CALL ZEROS_VEC(JEC, R(IX))
          CALL ZEROS_MAT(JEC,JEC, R(IA2))
          CALL ZEROS_VEC(JEC, R(IH2))
        END IF
        TT = 1
        CALL JEC_VALUE(TOL, RUN, JEC, NE, R(IT), R(IPD), R(IEC))
        CALL ZEROS_VEC(NE, R(IPD))
        CALL MULTIPLY_MAT_VEC2(NE, JEC, R(IH2), R(IH), R(IT))
        CALL MAKEA2(JEC, NE, R(IA), R(IA2),R(IEC))
        CALL CGR_METHOD(JEC, R(IA2), R(IH2), R(IX), R(IR), R(IVK),
     &       R(IAVK), TOL2, MAX_ITER_NUM)

        CALL MULTIPLY_MAT_VEC(NE, JEC, R(IPD), R(IX), R(IT))
        IF (RUN .EQ. 1) GOTO 10
```

```
  20  CONTINUE
        NF = 0.0
        X1P = 0.0
        X2P = 0.0
        CALL FORCE_CALC(N1, N2, R(IPD), R(IP), R(IX1), R(IX2), X1P,
     &      X2P, NF, A, B)
        ERR = (ABS(NF - 5.003728385561319D+4)/5.003728385561319D+4)*
     & 100.0D0
        TIME_END2 = ETIME(TIME_ARRAY)
        CALL SYSTEM_CLOCK(COUNT=end_count)
c       CALCULATE WALL-CLOCK TIME AND CPUT TIME
        WALLCLOCK = REAL(end_count - start_count) / REAL(count_rate)
        CPU_TIME_USED = TIME_END2 - TIME_START2
        write ( *, * ) 'Wall-clock time: ', WALLCLOCK, 'seconds'
        write ( *, * ) 'CPU time used: ', CPU_TIME_USED ,
     & 'seconds'
        write ( *, * ) "//=======================================//"
        write ( *, * ) "Total Normal Force (N) value is:", NF, "N"
        write ( *, * ) "//=======================================//"
        write ( *, * ) "Relative error ",
     & "(NORM + CG (Fortran) vs. NORM (Matlab))"
        write ( *, * )  ERR, "%"
        write ( *, * ) "//=======================================//"
        CALL compare_matrices(R(IP), R(IH), N1, N2)
        CALL SYSTEM('PAUSE')
      END

      SUBROUTINE READ_SCALAR_VARIABLES(E, NU, A, B, N1, N2)
C     ========================================================================
C     READING OF SCALAR VARIABLES` VALUES FROM FILES
C     ========================================================================
C     INPUTS (which are outputs itself):
C     |   E         : Young modulus [N/m^2]
C     |   nu        : Poisson`s ratio [·]
C     |   N1,N2     : Number of elements in the rolling and lateral directions
C     |   a         : Half size of the element - rolling direction [m]
C     |   b         : Half size of the element - lateral direction [m]
C     ========================================================================
        DOUBLE PRECISION E, NU, A, B
        INTEGER N1, N2
        INTEGER IERR
```

```
      OPEN(UNIT=1, FILE='E.txt', STATUS='OLD', IOSTAT=IERR)
      IF (IERR .ne. 0) STOP "Error opening E.txt"
      READ(1, *) E
      CLOSE(1)
      OPEN(UNIT=2, FILE='nu.txt', STATUS='OLD', IOSTAT=IERR)
      IF (IERR .ne. 0) STOP "Error opening nu.txt"
      READ(2, *) NU
      CLOSE(2)
      OPEN(UNIT=3, FILE='a.txt', STATUS='OLD', IOSTAT=IERR)
      IF (IERR .ne. 0) STOP "Error opening a.txt"
      READ(3, *) A
      CLOSE(3)
      OPEN(UNIT=4, FILE='b.txt', STATUS='OLD', IOSTAT=IERR)
      IF (IERR .ne. 0) STOP "Error opening b.txt"
      READ(4, *) B
      CLOSE(4)
      OPEN(UNIT=5, FILE='N1.txt', STATUS='OLD', IOSTAT=IERR)
      IF (IERR .ne. 0) STOP "Error opening N1.txt"
      READ(5, *) N1
      CLOSE(5)
      OPEN (UNIT=20, FILE='N2.txt', STATUS='OLD', IOSTAT=IERR)
      IF (IERR .ne. 0) STOP "Error opening N1.txt"
      READ(20, *) N2
      CLOSE(20)
    END

    SUBROUTINE READ_VECTOR_AND_MATRIX(N1, N2, H, X1, X2)
C     ========================================================================
C     READING OF VECTORS AND MATRICES VALUES FROM FILES
C     ========================================================================
C     INPUTS (which are outputs itself):
C     |   N1,N2     : Number of elements in the rolling and lateral directions
C     |   H         : Undeformed distance matrix N1xN2 [m]
C     |   x1        : Longitudinal coordinates of the elements in the PCA [m]
C     |   x2        : Lateral coordinates of the elements in the PCA [m]
C     ========================================================================
      INTEGER N1, N2, I, J, IERR
      DOUBLE PRECISION H(N1, N2), X1(N1), X2(N2)
      CHARACTER*80 LINE
```

```
      ! Read vector X1 from x1.txt
      OPEN(UNIT=7, FILE='x1.txt', STATUS='OLD', IOSTAT=IERR)
      IF (IERR /= 0) STOP "Error opening x1.txt"
      DO I = 1, N1
          READ(7, *) X1(I)
      END DO
      CLOSE(7)


      OPEN(UNIT=8, FILE='x2.txt', STATUS='OLD', IOSTAT=IERR)
      IF (IERR /= 0) STOP "Error opening x2.txt"
      DO I = 1, N2
          READ(8, *) X2(I)
      END DO
      CLOSE(8)
      OPEN(UNIT=9, FILE='h.txt', STATUS='OLD', IOSTAT=IERR)
      IF (IERR /= 0) THEN
          PRINT *, 'Error opening h.txt'
          STOP
      END IF
      DO I = 1, N1
          READ(9, *, IOSTAT=IERR) (H(I, J), J = 1, N2)
          IF (IERR /= 0) THEN
              PRINT *, 'Error reading h.txt at row', I
              CLOSE(9)
              STOP
          END IF
      END DO
      CLOSE(9)
    END

      SUBROUTINE CREATION_AES(N1, N2, A, B, NU, E, AES, X1, X2)
C     ======================================================================
C     Aes CREATION
C     ======================================================================
C     INPUTS:
C     |    N1,N2   : Number of elements in the rolling and lateral directions
C     |    a       : Half size of the element - rolling direction [m]
C     |    b       : Half size of the element - lateral direction [m]
C     |    nu      : Poisson`s ratio [·]
C     |    E       : Young modulus [N/m^2]
```

```
C     |    F       : Flexibility coefficient
C     |    Aes     : Influence coefficient`s matrix for each element [N1xN2]
C     |    x1      : Longitudinal coordinates of the elements in the PCA [m]
C     |    x2      : Lateral coordinates of the elements in the PCA [m]
C     ========================================================================
C     WHAT IS BEING CALCULATED?:
C     |    Aes     : Influence coefficient`s matrix for each element [N1xN2]
C     ========================================================================
        INTEGER N1, N2, I, J
        DOUBLE PRECISION A, B, NU, E, F, X, Y, AUX
        DOUBLE PRECISION AES(N1, N2), X1(N1), X2(N2)
        DOUBLE PRECISION PI

        PI = ACOS(-1.0)
        AUX = 2.0 * ((1.0 - NU**2) / (PI * E))
        DO I = 1, N1
          X = X1(I) - X1(1)
          DO J = 1, N2
              Y = X2(J) - X2(1)
              CALL CT_BC(X, Y, A, B, F)
              AES(I, J) = F * AUX
          END DO
        END DO
      END

      SUBROUTINE CT_BC(X, Y, A, B, F)
C     ========================================================================
C     FLEXIBILITY COEFFICIENTS CALCULATION
C     ========================================================================
C     Calculate the flexibility coeff. A, which relates the displacement
C     at (x,y) due to an applied unitary load on a 2ax2b origin-centered
C     rectangle. Love (1929), Johnson page 54
C     Note: Multiply by (1-poisson^2)/pi/E
C     ========================================================================
C     INPUTS:
C     |    x       : x-coord where displacement is measured [m]
C     |    y       : y-coord where displacement is measured [m]
C     |    a       : Half size of the element - rolling direction [m]
C     |    b       : Half size of the element - lateral direction [m]
C     ========================================================================
C     OUTPUT:
```

```
C     |    F      : Flexibility coefficient
C     ================================================================================
        double precision X, Y, A, B, F

       F = (X + A) * LOG(((Y + B) + SQRT((Y + B)**2 + (X + A)**2)) /
     & ((Y - B) + SQRT((Y - B)**2 + (X + A)**2))) +
     &   (Y + B) * LOG(((X + A) + SQRT((Y + B)**2 + (X + A)**2)) /
     & ((X - A) + SQRT((Y + B)**2 + (X - A)**2))) +
     &   (X - A) * LOG(((Y - B) + SQRT((Y - B)**2 + (X - A)**2)) /
     & ((Y + B) + SQRT((Y + B)**2 + (X - A)**2))) +
     &   (Y - B) * LOG(((X - A) + SQRT((Y - B)**2 + (X - A)**2)) /
     & ((X + A) + SQRT((Y - B)**2 + (X + A)**2)))
       END

       SUBROUTINE RC_CALC(N1, N2, NE, RC)
C     ================================================================================
C     ROW - COLUMN MATRIX
C     ================================================================================
C     INPUTS:
C     |    N1      : Number of elements in the rolling direction
C     |    N2      : Number of elements in the lateral direction
C     |    NE      : Dimension of the problem
C     |    RC      : Matrix that stores the position of PCA elements [NEx2]
C     ================================================================================
C     WHAT IS BEING CALCULATED?:
C     |    RC      : Matrix that stores the position of PCA elements [NEx2]
C     ================================================================================
        INTEGER N1, N2, NE, I, J, V
        DOUBLE PRECISION RC(NE, 2)

        V = 1
        DO I = 1, N2
          DO J = 1, N1
            IF (V .LE. NE) THEN
              RC(V, 1) = J
              RC(V, 2) = I
              V = V + 1
            END IF
          END DO
        END DO
       END
```

```
      SUBROUTINE A_FORMAT(NE, N1, N2, A, AES, RC)
C     ============================================================================
C     A CREATION
C     ============================================================================
C     INPUTS:
C     |    NE      : Dimension of the problem
C     |    N2      : Number of elements in the lateral direction
C     |    A       : Matrix containing the influence between all elements of the PCA [NExNE]
C     |    Aes     : Influence coefficient`s matrix for each element [N1xN2]
C     |    RC      : Matrix that stores the position of PCA elements [NEx2]
C     ============================================================================
C     WHAT IS BEING CALCULATED?:
C     |    A       : Matrix containing the influence between all elements of the PCA [NExNE]
C     ============================================================================
        INTEGER NE, N2, I, J, Ro, Co
        DOUBLE PRECISION A(NE, NE), AES(N1, N2), RC(NE, 2)

        DO I = 1, NE
          DO J = 1, NE
            Ro = ABS(RC(J, 1) - RC(I, 1))+1
            Co = ABS(RC(J, 2) - RC(I, 2))+1
            A(I, J) = AES(Ro, Co)
          END DO
        END DO
      END

      SUBROUTINE CGR_METHOD(M, A, H, P, R, VK, AVK, T, IT)
c     //=========================================================================
c     //  | CONJUGATE GRADIENT METHOD
c     //=========================================================================
c     //  | This code aims to caclulate the solucion P of a linear system of
c     //  | equations like AP=H using the Conjugate Gradeint Method.
c     //
c     //  | Conditions needed to found a solution or being able to use the code:
c     //  | - A must be a symmetric matrix -> A = A^T
c     //  | - A must be possitive definite -> (P^T)AP > 0 ∀ P ∈ R
c     //=========================================================================
c     //  | INPUTS:
c     //  |    M        : Dimension of the problem
c     //  |    A        : Matrix which contains the influence between all elements of the PCA
(also the A matrix of the linear system)
```

```
c     //  |    H        : Vector representing the right-hand side of the linear system
c     //  |    P        : Solution of the linear system
c     //  |    R         : Residual vector used to track the difference between the current
solution and the true solution
c     //  |    VK       : Direction vector along which the solution is improved
c     //  |    AVK      : Result of multiplying the matrix A by the direction vector Vk
c     //  |    T        : Threshold below which the solution is considered accurate enough
c     //  |    IT       : The number of iterations used
c     //=========================================================================
c     //  | WHAT IS BEING CALCULATED?:
c     //  |    P        : Solution of the linear system
c     //=========================================================================
        INTEGER M, IT
        DOUBLE PRECISION ALPHA, BETA, RSOLD, T, PROD, BB
        DOUBLE PRECISION A(M, M), H(M), P(M)
        DOUBLE PRECISION R(M), VK(M), AVK(M)

        CALL MATVEC(AVK, P, M, A)
        CALL VEC_SUB(R, H, AVK, M)
        CALL VEC_COPY(VK, R, M)
        CALL DOT_PRODUCT(R, R, M, PROD)
        CALL DOT_PRODUCT(H, H, M, BB)  ! Calcula bb = H'*H
        RSOLD=PROD
        IT=0
        DO WHILE (PROD .GT. T**2 * BB)
          ! Código del bucle que se ejecutará mientras la condición se cumpla
          CALL MATVEC(AVK, VK, M, A)
          CALL DOT_PRODUCT(VK, AVK, M, PROD)
          ALPHA = RSOLD / PROD
          CALL VEC_ADD(P, P, VK, M, ALPHA)
          CALL VEC_ADD(R, R, AVK, M, (-1*ALPHA))
          CALL DOT_PRODUCT(R, R, M, PROD)
          BETA = PROD / RSOLD
          CALL VEC_ADD(VK, R, VK, M, BETA)
          RSOLD = PROD
          IT=IT+1
      END DO
      END

      SUBROUTINE DOT_PRODUCT(X, Y, SIZE, PROD)
        INTEGER SIZE, I
        DOUBLE PRECISION X(SIZE), Y(SIZE), PROD
```

```
  PROD = 0.0
  DO I = 1, SIZE
    PROD = PROD + X(I) * Y(I)
  END DO
END

SUBROUTINE MATVEC(Y, X, SIZE, A)
  INTEGER SIZE, I, J
  DOUBLE PRECISION Y(SIZE), X(SIZE), A(SIZE, SIZE)
  DO I = 1, SIZE
    Y(I) = 0.0D0
    DO J = 1, SIZE
      Y(I) = Y(I) + A(I, J) * X(J)
    END DO
  END DO

END

SUBROUTINE VEC_ADD(RESULT, X, Y, SIZE, ALPHA)

  INTEGER SIZE, I
  DOUBLE PRECISION RESULT(SIZE), X(SIZE), Y(SIZE)
  DOUBLE PRECISION ALPHA
  DO I = 1, SIZE
    RESULT(I) = X(I) + ALPHA * Y(I)
  END DO

END

SUBROUTINE VEC_SUB(RESULT, X, Y, SIZE)

  INTEGER SIZE, I
  DOUBLE PRECISION RESULT(SIZE), X(SIZE), Y(SIZE)
  DO I = 1, SIZE
    RESULT(I) = X(I) - Y(I)
  END DO
END

SUBROUTINE VEC_COPY(DEST, SRC, SIZE)

  INTEGER SIZE, I
```

```
        DOUBLE PRECISION DEST(SIZE), SRC(SIZE)
        DO I = 1, SIZE
          DEST(I) = SRC(I)
        END DO
      END

      SUBROUTINE ZEROS_VEC(SIZE, v)
C     ============================================================================
C     INITIALIZATION OF VECTORS
C     ============================================================================
C     INPUTS:
C     |    size    : Dimension of the problem
C     |    v       : Vector to be initialized
C     ============================================================================
C     WHAT IS BEING CALCULATED?:
C     |    v       : Vector initialized
C     ============================================================================
        INTEGER SIZE, I
        DOUBLE PRECISION v(SIZE)

        DO I = 1, SIZE
          v(I) = 0.0
        END DO
      END

      SUBROUTINE ZEROS_MAT(SIZE1,SIZE2, MAT)
C     ============================================================================
C     INITIALIZATION OF MATRICES
C     ============================================================================
C     INPUTS:
C     |    size1   : Rows of the matrix
C     |    size2   : Columns of the matrix
C     |    MAT     : Matrix to be initialized
C     ============================================================================
C     WHAT IS BEING CALCULATED?:
C     |    MAT     : Matrix initialized
C     ============================================================================
        INTEGER SIZE1, SIZE2, J, I
        DOUBLE PRECISION MAT(SIZE1,SIZE2)

        DO I = 1, SIZE1
```

```
          DO J=1, SIZE2
            MAT(I,J) = 0.0
          END DO
        END DO
      END

      SUBROUTINE JEC_VALUE(TOL, RUN, JEC, NE, T, PD, EC)
C     ================================================================================
C     JEC`S VALUE CALCULATION
C     ================================================================================
C     INPUTS:
C     |    TOL      : Tolerance to differentiate elements in contact against those not in
contact
C     |    RUN      : Logical value (0: The real contact area has been found. 1: The contact
area must be re-calulated)
C     |    JEC      : Number of elements in contact in the real contact area
C     |    NE       : Total dimension of the problem
C     |    T        : Matrix that stores (with unitary value) the positions to be used in
matrices and vectors to calculate the new values after updating JEC
C     |    PD       : Normal traction vector NEx1 [N/m^2]
C     |    EC       : Vector that stores the position in Influence coefficients` matrix A to
be considered based on PD`s value.
C     ================================================================================
C     WHAT IS BEING CALCULATED?:
C     |    RUN      : Logical value (0: The real contact area has been found. 1: The contact
area must be re-calulated)
C     |    JEC      : Number of elements in contact in the real contact area
C     |    T        : Matrix that stores (with unitary value) the positions to be used in
matrices and vectors to calculate the new values after updating JEC
C     |    EC       : Vector that stores the position in Influence coefficients` matrix A to
be considered based on PD`s value.
C     ================================================================================
        INTEGER RUN, JEC, NE, N
        DOUBLE PRECISION TOL, T(NE,NE), PD(NE), EC(NE)

        JEC=0
        DO N = 1, NE
          IF (PD(N) .LT. TOL) THEN
              IF ((PD(N) .LT. (-TOL)) .AND.
     & (RUN .EQ. 0)) THEN
                RUN = 1
              END IF
          ELSE
```

```
                JEC=JEC + 1
                T(N,JEC)= 1.0
                EC(JEC)=N
            END IF
          END DO
        END

        SUBROUTINE MULTIPLY_MAT_VEC(R, C, RE, V, A)
C       ============================================================================
C       MULTIPLY A VECTOR AND A MATRIX (RE=A*V)
C       ============================================================================
C       INPUTS:
C       |    c       : Columns
C       |    r       : Rows
C       |    Re      : Result of the multiplication
C       |    v       : vector to multiply
C       |    A       : matrix to multiply
C       ============================================================================
C       WHAT IS BEING CALCULATED?:
C       |    Re      : Result of the multiplication
C       ============================================================================
          INTEGER C, R, I, J
          DOUBLE PRECISION RE(R), V(C), A(R, C)

          DO I = 1, R
            DO J = 1, C
              RE(I) = RE(I) + A(I, J) * V(J)
            END DO
          END DO
        END

        SUBROUTINE MULTIPLY_MAT_VEC2(R, C, RE, V, A)
C       ============================================================================
C       MULTIPLY A VECTOR AND A MATRIX (RE=A^T*V)
C       ============================================================================
C       INPUTS:
C       |    c       : Columns
C       |    r       : Rows
C       |    Re      : Result of the multiplication
C       |    v       : vector to multiply
C       |    A       : matrix to multiply
```

```
C     ========================================================================
C     WHAT IS BEING CALCULATED?:
C     |    Re     : Result of the multiplication
C     ========================================================================
        INTEGER C, R, I, J
        DOUBLE PRECISION RE(C), V(R), A(R, C)

        DO I = 1, R
          DO J = 1, C
            RE(J) = RE(J) + A(I, J) * V(I)
          END DO
        END DO
      END

      SUBROUTINE MAKEA2(JEC, NE, A, A2,EC)
C     ========================================================================
C     CALCULATION OF THE A2 MATRIX
C     ========================================================================
C     INPUTS:
C     |    jec     : New dimension of the problem (after tension value evaluation)
C     |    NE      : Dimension of the problem
C     |    T       : Matrix containing the positions of A to consider (after tension value
evaluation) [NExNE]
C     |    A       : Matrix containing the influence coefficients between all elements of the
PCA [NExNE]
C     |    A2      : Matrix containing the influence coefficients between all elements of the
updated PCA (after tension value evaluation) [jec x jec]
C     |    EC      : Vector that stores the position in Influence coefficients` matrix A to
be considered in A2 based on PD`s value. (Elements in contact area)
C     ========================================================================
C     WHAT IS BEING CALCULATED?:
C     |    A2      : Matrix containing the influence between all elements of the updated PCA
(after tension value evaluation) [jec x jec]
C     ========================================================================
        INTEGER JEC, NE, I, J
        DOUBLE PRECISION A(NE, NE), A2(JEC, JEC)
        double precision EC(NE)
        DO J =1,JEC
          DO I=1,JEC
            A2(I,J)=A(INT(EC(I)),INT(EC(J)))
          END DO
        END DO
      END
```

```
      SUBROUTINE FORCE_CALC(N1, N2, PD, P, X1, X2, X1P,
     & X2P, NF, A, B)
C     ================================================================================
C     CALCULATION OF THE TOTAL FORCE
C     ================================================================================
C     INPUTS:
C     |    N1,N2   : Number of elements in the rolling and lateral directions
C     |    PD      : Normal traction vector NEx1 [N/m^2]
C     |    P       : Normal traction N1xN2 matrix [N/m^2]
C     |    X1      : Longitudinal coordinates of the elements in the PCA [m]
C     |    X2      : Lateral coordinates of the elements in the PCA [m]
C     |    X1P,X2P : Coordinates of the centre of pressure [m]
C     |    NF      : Total normal force [N]
C     |    A       : Half size of the element - rolling direction [m]
C     |    B       : Half size of the element - lateral direction [m]
C     ================================================================================
C     WHAT IS BEING CALCULATED?:
C     |    P       : Normal traction N1xN2 matrix [N/m^2]
C     |    NF      : Total normal force [N]
C     |    X1P,X2P : Coordinates of the centre of pressure [m]
C     ================================================================================
        INTEGER N1, N2, I, J, M, N
        DOUBLE PRECISION PD(NE), P(N1, N2)
        DOUBLE PRECISION X1(N1), X2(N2),SIGMA
        DOUBLE PRECISION X1P, X2P, SUMA, NF, A, B

        SIGMA=0.0
        DO N = 1, N2
          DO M = 1, N1
            P(M, N) = PD((N - 1) * N1 + M)
          END DO
        END DO
C Addition of all elements in pa to calculate the total traction
        DO N=1,N2
          DO M=1,N1
            SIGMA=SIGMA+P(M,N)
          END DO
        END DO
C Calculate X1P
        SUMA = 0.0
```

```
        DO I = 1, N1
          DO J = 1, N2
              SUMA = SUMA + P(I, J)
          END DO
            X1P = X1P + SUMA * X1(I)
        END DO
        X1P = X1P / SIGMA
C Calculate X2P
        SUMA = 0.0
        DO I = 1, N2
          DO J = 1, N1
              SUMA = SUMA + P(J, I)
          END DO
              X2P = X2P + SUMA * X2(I)
        END DO
        X2P = X2P / SIGMA
C Calculate NF
      NF = SIGMA * 4 * A * B
      END

      SUBROUTINE COMPARE_MATRICES(MATRIX, REF_MATRIX, N1, N2)
C     ======================================================================
C     COMPARE TWO MATRICES AND CALCULATE ERRORS
C     ======================================================================
C     INPUTS:
C     |    MATRIX        : Matrix to be compared [n1 x n2]
C     |    REF_MATRIX    : Reference matrix to compare against [n1 x n2]
C     |    N1, N2        : Dimensions of the matrices
C     OUTPUTS:
C     |    MAX_ABS_ERROR : Maximum absolute error between the two matrices
C     |    MAX_REL_ERROR : Maximum relative error between the two matrices
C     FUNCTIONALITY:
C     |    Reads the reference matrix from a file (REF.TXT), compares it
C     |    element by element with the provided matrix, and computes the
C     |    maximum absolute and relative errors.
C     ======================================================================
        integer n1, n2, i, j
        DOUBLE PRECISION matrix(n1, n2), ref_matrix(n1, n2)
        DOUBLE PRECISION max_abs_error, max_rel_error, abs_error, rel_error

        max_abs_error = 0.0d0
```

```
      max_rel_error = 0.0d0
C     Read the reference matrix from the file REF.TXT
      open(unit=10, file='ref.txt', status='old')
      do i = 1, n1
         read(10,*) (ref_matrix(i, j), j = 1, n2)
      enddo
      close(10)
C     Compare the matrices
      do i = 1, n1
         do j = 1, n2
            abs_error = abs(matrix(i,j) - ref_matrix(i,j))
            if (ref_matrix(i,j) .ne. 0.0d0) then
               rel_error = abs_error / abs(ref_matrix(i,j))
            else
               rel_error = 0.0d0
            endif
            if (abs_error > max_abs_error) max_abs_error = abs_error
            if (rel_error > max_rel_error) max_rel_error = rel_error
         enddo
      enddo
C     Print the maximum absolute and relative errors
      print*, 'Max absolute error:', max_abs_error/1d+6
      print*, 'Max relative error:', max_rel_error
      return
      END
```

1.4. NORM+E/P

```
      PROGRAM NORMAL_CONTACT
C     The following code computes the normal traction distribution. Non-Hertzian model.
C     This is the fastest version
C     ========================================================================
C     INPUTS:
C     |    E       : Young modulus [N/m^2]
C     |    NU      : Poisson`s ratio [·]
C     |    N1,N2   : Number of elements in the rolling and lateral directions
C     |    H       : Undeformed distance matrix N1xN2 [m]
C     |    X1      : Longitudinal coordinates of the elements in the PCA [m]
C     |    X2      : Lateral coordinates of the elements in the PCA [m]
C     |    A       : Half size of the element - rolling direction [m]
C     |    B       : Half size of the element - lateral direction [m]
C
```

```
C     OUTPUTS:
C     |    P     : Normal traction N1xN2 matrix [N/m^2]
C     |    NF      : Total normal force [N]
C     |    X1P,X2P : Coordinates of the centre of pressure [m]
C     ================================================================================
C     PCA Diagram:
C                (Lateral)
C                    A X2
C     ---------------|------------------
C     |  |  |  |  |  |  |  |  |  |  |44|
C     ---------------------------------
C     |  |  |  |  |  |  |  |  |  |  |  |
C     ------------------------------------> X1  (Rolling)
C     |12|  |  |  |  |  |  |  |  |  |  |
C     ---------------------------------
C     | 1| 2| 3| 4| 5| 6| 7| 8| 9|10|11|
C     ---------------------------------
C
C     ================================================================================
        REAL TIME_ARRAY(2)
        DOUBLE PRECISION E, NU, A, B, NF, X1P, X2P, TOL, TOL2
        DOUBLE PRECISION CPU_TIME_USED, ERR
        INTEGER N1, N2, NE, NESQR, RUN, JEC, TT, IX1, IX2
        INTEGER IRC, IH, IAES, IT, IA, IPD, IH2, ITER
        INTEGER IA2, IX, IP, IR, IVK, IAVK, IEC, IEND
        INTEGER ILS, ILAMBDA, MAX_ITER_NUM,DIM,G
        INTEGER IQT_RHT,IQTAQ,IQ,STRP,ITEMP,ITEMP2
        LOGICAL YAW
        DOUBLE PRECISION PI
        REAL(KIND=16) R(15500000)
        PARAMETER (PI = acos(-1.0d0))
        DOUBLE PRECISION TIME_START2, TIME_END2,WALLCLOCK
        INTEGER start_count, end_count, count_rate, count_max

        TOL = 1.0E-9
        TOL2 = 1.0E-13
        MAX_ITER_NUM=55
        YAW=.true.
        G=4
        PRINT *, "//=======================================//"
        PRINT *, "//=========NORM_V16_WEAR_GE.F CODE=========//"
```

```
PRINT *, "//============== Paper used: ==============//"
PRINT *, "//==J Giner - A fast version of 'CONTACT'==//"
PRINT *, "//=====for normal problem calculations=====//"
PRINT *, "//=========================================//"
IF (.NOT. YAW)THEN
PRINT *, "//====ELLIPTICAL TRACTION DISTRIBUTION=====//"
ELSE
PRINT *, "//====POLYNOMICAL TRACTION DISTRIBUTION====//"
END IF
PRINT *, "//=========================================//"
PRINT *, "//===========CLASSIC NORM + E/P============//"

CALL READ_SCALAR_VARIABLES(E, NU, A, B, N1, N2)

NE = N1 * N2
NESQR = NE ** 2
IH = 1
IX1 = IH + NE
IX2 = IX1 + N1
IRC = IX2 + N2
IAES = IRC + NE * 2
IT = IAES + NE
IA = IT + NESQR
IPD = IA + NESQR
IH2 = IPD + NE
IA2 = IH2 + NE
IX = IA2 + NESQR
IP = IX + NE
IEC=IP+NE
IQ=IEC+NE
ILS=IQ+NESQR
IQTAQ=ILS+N2
IQT_RHT=IQTAQ+NE*15
ILAMBDA=IQT_RHT+NE
ITEMP=ILAMBDA+NE
IVEC=ITEMP+NESQR
ITEMP2=IVEC+NE
IEND = ITEMP2 + NESQR

PRINT*, 'Memory used:',IEND, 'elements'
IF (IEND .LT. 15500000) THEN
```

```
          WRITE (*,*) 'Correct Allocation'
        ELSE
          WRITE (*,*) 'IEND is greater than maximum capacity for R.'
        END IF
        CALL ZEROS_VEC(IEND,R)
        CALL READ_VECTOR_AND_MATRIX(N1, N2, R(IH), R(IX1), R(IX2))
c       ¡CODE TIMMER STARTS!
        TIME_START2 = ETIME(TIME_ARRAY)
        CALL SYSTEM_CLOCK(COUNT_RATE=count_rate, COUNT_MAX=count_max)
        CALL SYSTEM_CLOCK(COUNT=start_count)
C       AES CREATION
        CALL CREATION_AES(N1, N2, A, B, NU, E, R(IAES), R(IX1), R(IX2))
C       MATRIX ROW-COLUMN CREATION
        CALL RC_CALC(N1, N2, NE, R(IRC))
C       A MATRIX CREATION
        CALL A_FORMAT(NE, N1, N2, R(IA), R(IAES), R(IRC))
C       P ELLIPTICAL DISTRIBUTION CALCULATION
        CALL VALUE_TO_VEC(N2, R(ILS),N1)
        IF (.NOT. YAW)THEN
          CALL CALC_Q_E(DIM,NE,N2,R(ILS),R(IQ))
          CALL COMPUTE_QT_A_Q(DIM, NE, R(IQ), R(IA), R(IQTAQ),
     & R(ITEMP))
        ELSE
          CALL CALC_Q_P(G,DIM,NE,N2,A,R(ILS),R(IQ))
          CALL COMPUTE_QT_A_Q(DIM, NE, R(IQ), R(IA), R(IQTAQ),
     & R(ITEMP))
        END IF
        CALL MULTIPLY_MAT_VEC2(NE, DIM, R(IQT_RHT), R(IH), R(IQ))
C       CONJUGATE GRADIENT METHOD
        CALL GE_PIVOT(DIM, R(IQTAQ), R(IQT_RHT), R(ILAMBDA),R(ITEMP2),
     & R(IVEC))
        CALL MULTIPLY_MAT_VEC(NE, DIM, R(IPD), R(ILAMBDA), R(IQ))
        RUN = 1
        TT = 0
C       NEW PD AND PCA CALCULATION LOOP
  10  CONTINUE
        IF (RUN .EQ. 0) GOTO 20
        RUN = 0
        IF (TT .EQ. 1) THEN
          CALL ZEROS_MAT(NE,JEC, R(IT))
          CALL ZEROS_MAT(JEC,DIM, R(IQ))
```

```
          CALL ZEROS_MAT(DIM,DIM, R(IQTAQ))
          CALL ZEROS_VEC(JEC, R(IX))
          CALL ZEROS_MAT(JEC,JEC, R(IA2))
          CALL ZEROS_VEC(JEC, R(IH2))
          CALL ZEROS_VEC(DIM, R(IQT_RHT))
          CALL ZEROS_MAT(JEC,DIM, R(ITEMP))
        ELSE
          CALL ZEROS_VEC(DIM, R(IQT_RHT))
          CALL ZEROS_MAT(NE,DIM, R(IQ))
          CALL ZEROS_MAT(DIM,DIM, R(IQTAQ))
          CALL ZEROS_MAT(NE,DIM, R(ITEMP))
        END IF
        DIM=0
        TT = 1
        CALL JEC_VALUE(TOL, RUN, N2, STRP, JEC, NE, R(IT), R(IPD),
     & R(IEC),R(ILS))
        CALL ZEROS_VEC(NE, R(IPD))
        CALL MULTIPLY_MAT_VEC2(NE, JEC, R(IH2), R(IH), R(IT))
        CALL MAKEA2(JEC, NE, R(IA), R(IA2),R(IEC))
C       P ELLIPTICAL DISTRIBUTION CALCULATION
        IF (.not. YAW) THEN
          CALL CALC_Q_E(DIM,JEC,STRP,R(ILS),R(IQ))
          CALL COMPUTE_QT_A_Q(DIM, JEC, R(IQ), R(IA2), R(IQTAQ),
     & R(ITEMP))
        ELSE
          CALL CALC_Q_P(G,DIM,JEC,STRP,A,R(ILS),R(IQ))
          CALL COMPUTE_QT_A_Q(DIM, JEC, R(IQ), R(IA2), R(IQTAQ),
     & R(ITEMP))
        END IF
        CALL MULTIPLY_MAT_VEC2(JEC, DIM, R(IQT_RHT), R(IH2), R(IQ))
C       GAUSS ELIMINATION
        CALL GE_PIVOT(DIM, R(IQTAQ), R(IQT_RHT), R(ILAMBDA),R(ITEMP2),
     & R(IVEC))
        CALL MULTIPLY_MAT_VEC(JEC, DIM, R(IX), R(ILAMBDA), R(IQ))
        CALL MULTIPLY_MAT_VEC(NE, JEC, R(IPD), R(IX), R(IT))
        IF (RUN .EQ. 1) GOTO 10
  20  CONTINUE
        NF = 0.0D0
        X1P = 0.0D0
        X2P = 0.0D0
        CALL FORCE_CALC(N1, N2, R(IPD), R(IP), R(IX1), R(IX2), X1P,
```

```
     &      X2P, NF, A, B)
            ERR = (ABS(NF - 5.003728385561319D+4)/5.003728385561319D+4)*
     & 100.0D0
            TIME_END2 = ETIME(TIME_ARRAY)
            CALL SYSTEM_CLOCK(COUNT=end_count)
c           CALCULATE WALL-CLOCK TIME AND CPU TIME
            WALLCLOCK = REAL(end_count - start_count) / REAL(count_rate)
            CPU_TIME_USED = TIME_END2 - TIME_START2
            write ( *, * ) 'Wall-clock time: ', WALLCLOCK, 'seconds'
            write ( *, * ) 'CPU time used: ', CPU_TIME_USED ,
     & 'seconds'
            write ( *, * ) "//========================================//"
            write ( *, * ) "Total Normal Force (N) value is:", NF,"N"
            write ( *, * ) "//========================================//"
            write ( *, * ) "Relative error (NORM + E/P + GE ",
     & "vs. NORM (Matlab))"
            write ( *, * )  ERR, "%"
            write ( *, * ) "//========================================//"
            CALL compare_matrices(R(IP), R(IH), N1, N2)

            CALL SYSTEM('PAUSE')
          END

          SUBROUTINE READ_SCALAR_VARIABLES(E, NU, A, B, N1, N2)
C     ================================================================================
C     READING OF SCALAR VARIABLES` VALUES FROM FILES
C     ================================================================================
C     INPUTS (which are outputs itself):
C     |   E         : Young modulus [N/m^2]
C     |   nu        : Poisson`s ratio [·]
C     |   N1,N2     : Number of elements in the rolling and lateral directions
C     |   a         : Half size of the element - rolling direction [m]
C     |   b         : Half size of the element - lateral direction [m]
C     ================================================================================
            DOUBLE PRECISION E, NU, A, B
            INTEGER N1, N2
            INTEGER IERR

            OPEN(UNIT=1, FILE='E.txt', STATUS='OLD', IOSTAT=IERR)
            IF (IERR .ne. 0) STOP "Error opening E.txt"
            READ(1, *) E
```

```
      CLOSE(1)
      OPEN(UNIT=2, FILE='nu.txt', STATUS='OLD', IOSTAT=IERR)
      IF (IERR .ne. 0) STOP "Error opening nu.txt"
      READ(2, *) NU
      CLOSE(2)
      OPEN(UNIT=3, FILE='a.txt', STATUS='OLD', IOSTAT=IERR)
      IF (IERR .ne. 0) STOP "Error opening a.txt"
      READ(3, *) A
      CLOSE(3)
      OPEN(UNIT=4, FILE='b.txt', STATUS='OLD', IOSTAT=IERR)
      IF (IERR .ne. 0) STOP "Error opening b.txt"
      READ(4, *) B
      CLOSE(4)
      OPEN(UNIT=5, FILE='N1.txt', STATUS='OLD', IOSTAT=IERR)
      IF (IERR .ne. 0) STOP "Error opening N1.txt"
      READ(5, *) N1
      CLOSE(5)
      OPEN (UNIT=20, FILE='N2.txt', STATUS='OLD', IOSTAT=IERR)
      IF (IERR .ne. 0) STOP "Error opening N1.txt"
      READ(20, *) N2
      CLOSE(20)
      END

      SUBROUTINE READ_VECTOR_AND_MATRIX(N1, N2, H, X1, X2)
C     ========================================================================
C     READING OF VECTORS AND MATRICES VALUES FROM FILES
C     ========================================================================
C     INPUTS (which are outputs itself):
C     |   N1,N2     : Number of elements in the rolling and lateral directions
C     |   H         : Undeformed distance matrix N1xN2 [m]
C     |   x1        : Longitudinal coordinates of the elements in the PCA [m]
C     |   x2        : Lateral coordinates of the elements in the PCA [m]
C     ========================================================================
      INTEGER N1, N2, I, J, IERR
      DOUBLE PRECISION H(N1, N2), X1(N1), X2(N2)
      CHARACTER*80 LINE

      ! Read vector X1 from x1.txt
      OPEN(UNIT=7, FILE='x1.txt', STATUS='OLD', IOSTAT=IERR)
      IF (IERR /= 0) STOP "Error opening x1.txt"
      DO I = 1, N1
```

```
          READ(7, *) X1(I)
        END DO
        CLOSE(7)
        OPEN(UNIT=8, FILE='x2.txt', STATUS='OLD', IOSTAT=IERR)
        IF (IERR /= 0) STOP "Error opening x2.txt"
        DO I = 1, N2
          READ(8, *) X2(I)
        END DO
        CLOSE(8)
        OPEN(UNIT=9, FILE='h.txt', STATUS='OLD', IOSTAT=IERR)
        IF (IERR /= 0) THEN
          PRINT *, 'Error opening h.txt'
          STOP
        END IF
        DO I = 1, N1
          READ(9, *, IOSTAT=IERR) (H(I, J), J = 1, N2)
          IF (IERR /= 0) THEN
            PRINT *, 'Error reading h.txt at row', I
            CLOSE(9)
            STOP
          END IF
        END DO
        CLOSE(9)
      END

      SUBROUTINE CREATION_AES(N1, N2, A, B, NU, E, AES, X1, X2)
C     ======================================================================
C     Aes CREATION
C     ======================================================================
C     INPUTS:
C     |    N1,N2   : Number of elements in the rolling and lateral directions
C     |    a       : Half size of the element - rolling direction [m]
C     |    b       : Half size of the element - lateral direction [m]
C     |    nu      : Poisson`s ratio [·]
C     |    E       : Young modulus [N/m^2]
C     |    F       : Flexibility coefficient
C     |    Aes     : Influence coefficient`s matrix for each element [N1xN2]
C     |    x1      : Longitudinal coordinates of the elements in the PCA [m]
C     |    x2      : Lateral coordinates of the elements in the PCA [m]
C     ======================================================================
C     WHAT IS BEING CALCULATED?:
```

```
C     |    Aes     : Influence coefficient`s matrix for each element [N1xN2]
C     ========================================================================
        INTEGER N1, N2, I, J
        DOUBLE PRECISION A, B, NU, E, F, X, Y, AUX
        DOUBLE PRECISION AES(N1, N2), X1(N1), X2(N2)
        DOUBLE PRECISION PI

        PI = ACOS(-1.0D0)
        AUX = 2.0 * ((1.0 - NU**2) / (PI * E))
        DO I = 1, N1
          X = X1(I) - X1(1)
          DO J = 1, N2
              Y = X2(J) - X2(1)
              CALL CT_BC(X, Y, A, B, F)
              AES(I, J) = F * AUX
          END DO
        END DO
      END

      SUBROUTINE CT_BC(X, Y, A, B, F)
C     ========================================================================
C     FLEXIBILITY COEFFICIENTS CALCULATION
C     ========================================================================
C     Calculate the flexibility coeff. A, which relates the displacement
C     at (x,y) due to an applied unitary load on a 2ax2b origin-centered
C     rectangle. Love (1929), Johnson page 54
C     Note: Multiply by (1-poisson^2)/pi/E
C     ========================================================================
C     INPUTS:
C     |    x      : x-coord where displacement is measured [m]
C     |    y      : y-coord where displacement is measured [m]
C     |    a      : Half size of the element - rolling direction [m]
C     |    b      : Half size of the element - lateral direction [m]
C     ========================================================================
C     OUTPUT:
C     |    F      : Flexibility coefficient
C     ========================================================================
        DOUBLE PRECISION X, Y, A, B, F

       F = (X + A) * LOG(((Y + B) + SQRT((Y + B)**2 + (X + A)**2)) /
     & ((Y - B) + SQRT((Y - B)**2 + (X + A)**2))) +
```

```
     &   (Y + B) * LOG(((X + A) + SQRT((Y + B)**2 + (X + A)**2)) /
     & ((X - A) + SQRT((Y + B)**2 + (X - A)**2))) +
     &   (X - A) * LOG(((Y - B) + SQRT((Y - B)**2 + (X - A)**2)) /
     & ((Y + B) + SQRT((Y + B)**2 + (X - A)**2))) +
     &   (Y - B) * LOG(((X - A) + SQRT((Y - B)**2 + (X - A)**2)) /
     & ((X + A) + SQRT((Y - B)**2 + (X + A)**2)))
      END

      SUBROUTINE RC_CALC(N1, N2, NE, RC)
C     ================================================================================
C     ROW - COLUMN MATRIX
C     ================================================================================
C     INPUTS:
C     |    N1      : Number of elements in the rolling direction
C     |    N2      : Number of elements in the lateral direction
C     |    NE      : Dimension of the problem
C     |    RC      : Matrix that stores the position of PCA elements [NEx2]
C     ================================================================================
C     WHAT IS BEING CALCULATED?:
C     |    RC      : Matrix that stores the position of PCA elements [NEx2]
C     ================================================================================
        INTEGER N1, N2, NE, I, J, V
        DOUBLE PRECISION RC(NE, 2)

        V = 1
        DO I = 1, N2
          DO J = 1, N1
            IF (V .LE. NE) THEN
              RC(V, 1) = J
              RC(V, 2) = I
              V = V + 1
            END IF
          END DO
        END DO
      END

      SUBROUTINE A_FORMAT(NE, N1, N2, A, AES, RC)
C     ================================================================================
C     CREATION OF MATRIX A
C     ================================================================================
C     INPUTS:
```

```
C     |    NE      : Dimension of the problem
C     |    N2      : Number of elements in the lateral direction
C     |    A       : Matrix containing the influence between all elements of the PCA [NExNE]
C     |    Aes     : Influence coefficient`s matrix for each element [N1xN2]
C     |    RC      : Matrix that stores the position of PCA elements [NEx2]
C     ========================================================================
C     WHAT IS BEING CALCULATED?:
C     |    A       : Matrix containing the influence between all elements of the PCA [NExNE]
C     ========================================================================
      INTEGER NE, N2, I, J, Ro, Co
      DOUBLE PRECISION A(NE, NE), AES(N1, N2), RC(NE, 2)

      DO I = 1, NE
        DO J = 1, NE
          Ro = ABS(RC(J, 1) - RC(I, 1))+1
          Co = ABS(RC(J, 2) - RC(I, 2))+1
          A(I, J) = AES(Ro, Co)
        END DO
      END DO
      END

      SUBROUTINE ZEROS_VEC(SIZE, v)
C     ========================================================================
C     INITIALIZATION OF VECTORS
C     ========================================================================
C     INPUTS:
C     |    size    : Dimension of the problem
C     |    v       : Vector to be initialized
C     ========================================================================
C     WHAT IS BEING CALCULATED?:
C     |    v       : Vector initialized
C     ========================================================================
      INTEGER SIZE, I
      DOUBLE PRECISION v(SIZE)

      DO I = 1, SIZE
        v(I) = 0.0
      END DO
      END

      SUBROUTINE ZEROS_MAT(SIZE1,SIZE2, MAT)
```

```
C     ===========================================================================
C     INITIALIZATION OF MATRICES
C     ===========================================================================
C     INPUTS:
C     |    size1   : Rows of the matrix
C     |    size2   : Columns of the matrix
C     |    MAT     : Matrix to be initialized
C     ===========================================================================
C     WHAT IS BEING CALCULATED?:
C     |    MAT     : Matrix initialized
C     ===========================================================================
        INTEGER SIZE1, SIZE2, J, I
        DOUBLE PRECISION MAT(SIZE1,SIZE2)

        DO I = 1, SIZE1
          DO J=1, SIZE2
            MAT(I,J) = 0.0
          END DO
        END DO
      END

      SUBROUTINE JEC_VALUE(TOL, RUN, N2,STRP, JEC, NE, T, PD, EC,LS)
C     ===========================================================================
C     JEC`S VALUE CALCULATION
C     ===========================================================================
C     INPUTS:
C     |     TOL      : Tolerance to differentiate elements in contact against those not in
contact
C     |    RUN     : Logical value (0: The real contact area has been found. 1: The contact
area must be re-calulated)
C     |    JEC     : Number of elements in contact in the real contact area
C     |    NE      : Total dimension of the problem
C     |     T        : Matrix that stores (with unitary value) the positions to be used in
matrices and vectors to calculate the new values after updating JEC
C     |    PD      : Normal traction vector NEx1 [N/m^2]
C     |    EC      : Vector that stores the position in Influence coefficients` matrix A to
be considered based on PD`s value.
C     ===========================================================================
C     WHAT IS BEING CALCULATED?:
C     |    RUN     : Logical value (0: The real contact area has been found. 1: The contact
area must be re-calulated)
C     |    JEC     : Number of elements in contact in the real contact area
```

```
C     |    T       : Matrix that stores (with unitary value) the positions to be used in
matrices and vectors to calculate the new values after updating JEC
C     |    EC      : Vector that stores the position in Influence coefficients` matrix A to
be considered based on PD`s value.
C     ========================================================================
        INTEGER RUN, JEC, NE, N,CHK,STRP,N2
        DOUBLE PRECISION TOL, T(NE,NE), PD(NE), EC(NE)
        DOUBLE PRECISION LS(N2)

        JEC=0
        CHK=0
        STRP=0
        DO N = 1, NE
          IF (PD(N) .LT. TOL) THEN
            CHK=0
              IF ((PD(N) .LT. (-TOL)) .AND.
     & (RUN .EQ. 0)) THEN
                RUN = 1
              END IF
          ELSE
            IF ((CHK .EQ. 0) .AND. (STRP .LT. N2)) THEN
              STRP=STRP+1
              CHK=1
              LS(STRP)=1;
            ELSE IF ((CHK .EQ. 1) .AND. (STRP .LE. N2)) THEN
                LS(STRP)=LS(STRP)+1;
                IF (LS(STRP) .EQ. N1)THEN
                CHK=0
                END IF
            END IF
                  JEC=JEC + 1
                  T(N,JEC)= 1.0
                  EC(JEC)=N
          END IF
        END DO
      END

      SUBROUTINE MULTIPLY_MAT_VEC(R, C, RE, V, A)
C     ========================================================================
C     MULTIPLY A VECTOR AND A MATRIX (RE=A*V)
C     ========================================================================
C     INPUTS:
```

```
C     |    c       : Columns
C     |    r       : Rows
C     |    Re      : Result of the multiplication
C     |    v       : vector to multiply
C     |    A       : matrix to multiply
C     ========================================================================
C     WHAT IS BEING CALCULATED?:
C     |    Re      : Result of the multiplication
C     ========================================================================
        INTEGER C, R, I, J
        DOUBLE PRECISION RE(R), V(C), A(R, C)

        DO I = 1, R
          DO J = 1, C
            RE(I) = RE(I) + A(I, J) * V(J)
          END DO
        END DO
      END

      SUBROUTINE MULTIPLY_MAT_VEC2(R, C, RE, V, A)
C     ========================================================================
C     MULTIPLY A VECTOR AND A MATRIX (RE=AT*V)
C     ========================================================================
C     INPUTS:
C     |    c      : Columns
C     |    r      : Rows
C     |    Re     : Result of the multiplication
C     |    v      : vector to multiply
C     |    A      : matrix to multiply
C     ========================================================================
C     WHAT IS BEING CALCULATED?:
C     |    Re     : Result of the multiplication of A transposed per v
C     ========================================================================
        INTEGER C, R, I, J
        DOUBLE PRECISION RE(C), V(R), A(R, C)
        DO I = 1, R
          DO J = 1, C
            RE(J) = RE(J) + A(I, J) * V(I)
          END DO
        END DO
      END
```

```
      SUBROUTINE MAKEA2(JEC, NE, A, A2,EC)
C     ================================================================================
C     CALCULATION OF THE A2 MATRIX
C     ================================================================================
C     INPUTS:
C     |    jec     : New dimension of the problem (after tension value evaluation)
C     |    NE      : Dimension of the problem
C     |    T       : Matrix containing the positions of A to consider (after tension value
evaluation) [NExNE]
C     |    A       : Matrix containing the influence coefficients between all elements of the
PCA [NExNE]
C     |    A2      : Matrix containing the influence coefficients between all elements of the
updated PCA (after tension value evaluation) [jec x jec]
C     |    EC      : Vector that stores the position in Influence coefficients` matrix A to
be considered in A2 based on PD`s value. (Elements in contact area)
C     ================================================================================
C     WHAT IS BEING CALCULATED?:
C     |    A2      : Matrix containing the influence between all elements of the updated PCA
(after tension value evaluation) [jec x jec]
C     ================================================================================
        INTEGER JEC, NE, I, J
        DOUBLE PRECISION A(NE, NE), A2(JEC, JEC)
        DOUBLE PRECISION EC(NE)

        DO J =1,JEC
          DO I=1,JEC
            A2(I,J)=A(INT(EC(I)),INT(EC(J)))
          END DO
        END DO
      END

      SUBROUTINE FORCE_CALC(N1, N2, PD, P, X1, X2, X1P,
     & X2P, NF, A, B)
C     ================================================================================
C     CALCULATION OF THE TOTAL FORCE
C     ================================================================================
C     INPUTS:
C     |    N1,N2   : Number of elements in the rolling and lateral directions
C     |    PD      : Normal traction vector NEx1 [N/m^2]
C     |    P     : Normal traction N1xN2 matrix [N/m^2]
C     |    X1      : Longitudinal coordinates of the elements in the PCA [m]
C     |    X2      : Lateral coordinates of the elements in the PCA [m]
```

```
C     |    X1P,X2P : Coordinates of the centre of pressure [m]
C     |    NF      : Total normal force [N]
C     |    A       : Half size of the element - rolling direction [m]
C     |    B       : Half size of the element - lateral direction [m]
C     ======================================================================
C     WHAT IS BEING CALCULATED?:
C     |    P     : Normal traction N1xN2 matrix [N/m^2]
C     |    NF      : Total normal force [N]
C     |    X1P,X2P : Coordinates of the centre of pressure [m]
C     ======================================================================
        INTEGER N1, N2, I, J, M, N
        DOUBLE PRECISION PD(NE), P(N1, N2)
        DOUBLE PRECISION X1(N1), X2(N2),SIGMA
        DOUBLE PRECISION X1P, X2P, SUMA, NF, A, B

        SIGMA=0.0
        DO N = 1, N2
          DO M = 1, N1
            P(M, N) = PD((N - 1) * N1 + M)
          END DO
        END DO
C Addition of all elements in pa to calculate the total traction
        DO N=1,N2
          DO M=1,N1
            SIGMA=SIGMA+P(M,N)
          END DO
        END DO
C Calculate X1P
        SUMA = 0.0
        DO I = 1, N1
          DO J = 1, N2
              SUMA = SUMA + P(I, J)
          END DO
            X1P = X1P + SUMA * X1(I)
        END DO
        X1P = X1P / SIGMA
C Calculate X2P
        SUMA = 0.0
        DO I = 1, N2
          DO J = 1, N1
              SUMA = SUMA + P(J, I)
```

```
          END DO
              X2P = X2P + SUMA * X2(I)
        END DO
        X2P = X2P / SIGMA
C Calculate NF
        NF = SIGMA * 4 * A * B
      END

      SUBROUTINE CALC_Q_E(DIM,SIZE,STRP,LS,QE)
C     ======================================================================
C     CALCULATION OF MATRIX Q (ELLIPTICAL BASIS)
C     ======================================================================
C     INPUTS:
C     |    DIM    : Dimension of the matrix Q
C     |    SIZE   : Size of the matrix Q
C     |    STRP   : Number of strips (subdivisions)
C     |    LS     : Array containing the lengths of the strips
C
C     OUTPUTS:
C     |    Q     : Matrix containing the elliptical basis
C                   for the stress distribution
C     ======================================================================

        INTEGER DIM, SIZE,I1,I2,I3,J,STRP
        DOUBLE PRECISION LS(STRP), QE(SIZE,DIM)
        DOUBLE PRECISION XES
        NUM=0;
        I3=0;

        DO J=1,STRP
          I2=INT(LS(J))
          DO I1=1,I2
            XES=-1*(2 * (I2/2 - I1 + 1) - 1)
            QE(I1+I3,2*J-1)=1
            QE(I1+I3,2*J)=SQRT(1 - (XES / I2)**2)
          END DO
          I3=I3+INT(LS(J))
        END DO
        DIM=2*STRP
      END
```

```
      SUBROUTINE COMPUTE_QT_A_Q(DIM, SIZE, QE, A, QT_A_Q,TEMP)
C     ======================================================================
C     COMPUTATION OF MATRIX QT_A_Q
C     ======================================================================
C     INPUTS:
C     |    DIM       : Dimension of matrices
C     |    SIZE      : Size of matrices
C     |    QE        : Matrix containing the elliptical or polynomial basis
C                      functions
C     |    A         : Matrix A of the system
C
C     OUTPUTS:
C     |    QT_A_Q    : Matrix result of QT * A * Q
C     |    TEMP      : Temporary matrix used for the calculation of QT * A * Q
C     ======================================================================
        INTEGER SIZE, DIM, I, J, K,L
        DOUBLE PRECISION QE(SIZE, DIM), A(SIZE, SIZE)
        DOUBLE PRECISION QT_A_Q(DIM, DIM),TEMP(SIZE,DIM)

C Compute TEMP = A * Q
        DO K = 1, SIZE
          DO J = 1, DIM
            IF (QE(K, J) .NE. 0.0D0) THEN
              DO L = 1, SIZE
                  TEMP(L, J) = TEMP(L, J) + A(L, K) * QE(K, J)
              END DO
            END IF
          END DO
        END DO
C Compute QT_A_Q = Q_T * TEMP
        DO L = 1, SIZE
          DO I = 1, DIM
            IF (QE(L, I) .NE. 0.0D0) THEN
              DO J = 1, DIM
                IF (TEMP(L, J) .NE. 0.0D0) THEN
                  QT_A_Q(I, J) = QT_A_Q(I, J) + QE(L, I) *
     & TEMP(L, J)
                END IF
              END DO
            END IF
          END DO
```

```
        END DO
      END

      SUBROUTINE VALUE_TO_VEC(SIZE,V,VALUE)
C     ======================================================================
C     ASSIGN CONSTANT VALUE TO VECTOR V
C     ======================================================================
C     INPUTS:
C     |    SIZE      : Size of the vector
C     |    VALUE     : Constant value to assign to all elements
C
C     OUTPUTS:
C     |    V         : Vector of size SIZE with all elements assigned VALUE
C     ======================================================================
        INTEGER VALUE,I,SIZE
        DOUBLE PRECISION V(SIZE)

        DO I=1,SIZE
          V(I)=VALUE
        END DO
      END

      SUBROUTINE CALC_Q_P(G,DIM,SIZE,STRP,A,LS,QP)
C     ======================================================================
C     CALCULATION OF MATRIX B (POLYNOMIAL BASIS)
C     ======================================================================
C     INPUTS:
C     |    G         : Degree of the polynomial
C     |    DIM       : Dimension of the matrix B
C     |    SIZE      : Size of the matrix B
C     |    STRP      : Number of strips (subdivisions)
C     |    A         : Scaling factor for polynomial basis functions
C     |    LS        : Array containing the lengths of the strips
C
C     OUTPUTS:
C     |    B         : Matrix containing the polynomial basis functions
C     ======================================================================
        INTEGER DIM, SIZE,I1,I2,I3,J,STRP,G,M
        DOUBLE PRECISION LS(STRP),XES
        DOUBLE PRECISION QP(SIZE,DIM), A
```

```
        I3=0;
        DO J=1,STRP
          I2=INT(LS(J))
          DO I1=1,I2
            XES=-1*(2 * (I2/2 - I1 + 1) - 1)
            DO M=0,G
              QP(I1+I3,(G+1)*J-(G-M))=(XES*A)**M;
            END DO
          END DO
          I3=I3+INT(LS(J))
        END DO
        DIM=(G+1)*STRP
      END

      SUBROUTINE GE_PIVOT(M, A, H, P, TEMP, VEC)
c     //=====================================================================
c     //  | GAUSS ELIMINATION WITH PARTIAL PIVOTING - OPTIMIZED
c     //=====================================================================
c     //  | This subroutine solves the linear system A * P = H using the Gauss
c     //  | Elimination method with partial pivoting, followed by back substitution.
c     //  | The code has been optimized to reduce execution time and improve
c     //  | numerical stability.
c     //
c     //  | INPUTS:
c     //  |    M        : Dimension of the problem (size of the matrix)
c     //  |    A        : Matrix containing the coefficients of the system
c     //  |    H        : Vector representing the right-hand side of the system
c     //  |    TEMP     : Temporary matrix used to store a copy of A
c     //  |    VEC      : Temporary vector used to store a copy of H
c     //
c     //  | OUTPUTS:
c     //  |    P        : Solution vector of the system
c     //=====================================================================
      INTEGER M, I, J, K, MAX_ROW
      DOUBLE PRECISION A(M, M), H(M), P(M)
      DOUBLE PRECISION TEMP(M,M), VEC(M)
      DOUBLE PRECISION FACTOR, MAX_VAL, TT

c     Copy A and H into TEMP and VEC to preserve the original matrices
      VEC = H
      TEMP = A
```

```
c     Forward Elimination with Partial Pivoting
      DO K = 1, M-1
c         Find the row with the largest pivot element in column K
          MAX_ROW = K
          MAX_VAL = ABS(A(K, K))
          DO I = K+1, M
              IF (ABS(A(I, K)) .GT. MAX_VAL) THEN
                  MAX_ROW = I
                  MAX_VAL = ABS(A(I, K))
              END IF
          END DO
c         Swap rows in matrix A and vector H if necessary
          IF (MAX_ROW .NE. K) THEN
              DO J = 1, M
                  TT = A(K, J)
                  A(K, J) = A(MAX_ROW, J)
                  A(MAX_ROW, J) = TT
              END DO
              TT = H(K)
              H(K) = H(MAX_ROW)
              H(MAX_ROW) = TT
          END IF
c         Perform elimination to convert A into an upper triangular matrix
          DO I = K+1, M
              FACTOR = A(I, K) / A(K, K)
              DO J = K+1, M
                  A(I, J) = A(I, J) - FACTOR * A(K, J)
              END DO
              H(I) = H(I) - FACTOR * H(K)
          END DO
      END DO
c     Back Substitution to find the solution vector P
      P(M) = H(M) / A(M, M)
      DO I = M-1, 1, -1
          P(I) = H(I)
          DO J = I+1, M
              P(I) = P(I) - A(I, J) * P(J)
          END DO
          P(I) = P(I) / A(I, I)
      END DO
c     Restore the original matrices A and H
```

```
      A = TEMP
      H = VEC
      RETURN
      END

      SUBROUTINE COMPARE_MATRICES(MATRIX, REF_MATRIX, N1, N2)
C     ======================================================================
C     COMPARE TWO MATRICES AND CALCULATE ERRORS
C     ======================================================================
C     INPUTS:
C     |    MATRIX        : Matrix to be compared [n1 x n2]
C     |    REF_MATRIX    : Reference matrix to compare against [n1 x n2]
C     |    N1, N2        : Dimensions of the matrices
C
C     OUTPUTS:
C     |    MAX_ABS_ERROR : Maximum absolute error between the two matrices
C     |    MAX_REL_ERROR : Maximum relative error between the two matrices
C
C     FUNCTIONALITY:
C     |    Reads the reference matrix from a file (REF.TXT), compares it
C     |    element by element with the provided matrix, and computes the
C     |    maximum absolute and relative errors.
C     ======================================================================

        integer n1, n2, i, j
        DOUBLE PRECISION matrix(n1, n2), ref_matrix(n1, n2)
        DOUBLE PRECISION max_abs_error, max_rel_error, abs_error, rel_error

        max_abs_error = 0.0d0
        max_rel_error = 0.0d0
C       Read the reference matrix from the file REF.TXT
        open(unit=10, file='ref.txt', status='old')
        do i = 1, n1
           read(10,*) (ref_matrix(i, j), j = 1, n2)
        enddo
        close(10)
C       Compare the matrices
        do i = 1, n1
           do j = 1, n2
              abs_error = abs(matrix(i,j) - ref_matrix(i,j))
              if (ref_matrix(i,j) .ne. 0.0d0) then
```

```
                rel_error = abs_error / abs(ref_matrix(i,j))
              else
                rel_error = 0.0d0
              endif
              if (abs_error > max_abs_error) max_abs_error = abs_error
              if (rel_error > max_rel_error) max_rel_error = rel_error
            enddo
          enddo
C         Print the maximum absolute and relative errors
          print*, 'Max absolute error:', max_abs_error/1d+6
          print*, 'Max relative error:', max_rel_error
          return
        END
```

1.5. NOMR+E/P+CG

```
        PROGRAM NORMAL_CONTACT
C       The following code computes the normal traction distribution. Non-Hertzian model.
C       This is the fastest version
C       ============================================================================
C       INPUTS:
C       |    E       : Young modulus [N/m^2]
C       |    NU      : Poisson`s ratio [·]
C       |    N1,N2   : Number of elements in the rolling and lateral directions
C       |    H       : Undeformed distance matrix N1xN2 [m]
C       |    X1      : Longitudinal coordinates of the elements in the PCA [m]
C       |    X2      : Lateral coordinates of the elements in the PCA [m]
C       |    A       : Half size of the element - rolling direction [m]
C       |    B       : Half size of the element - lateral direction [m]
C       OUTPUTS:
C       |    P     : Normal traction N1xN2 matrix [N/m^2]
C       |    NF      : Total normal force [N]
C       |    X1P,X2P : Coordinates of the centre of pressure [m]
C       ============================================================================
C       PCA Diagram:
C                  (Lateral)
C                      A X2
C       ---------------|------------------
C       |  |  |  |  |  |  |  |  |  |  |44|
C       ---------------------------------
C       |  |  |  |  |  |  |  |  |  |  |  |
C       ------------------------------------> X1  (Rolling)
```

```
C     |12|  |  |  |  |  |  |  |  |  |
C     ---------------------------------
C     | 1| 2| 3| 4| 5| 6| 7| 8| 9|10|11|
C     ---------------------------------
C
C     ========================================================================
        REAL TIME_ARRAY(2)
        DOUBLE PRECISION E, NU, A, B, NF, X1P, X2P, TOL, TOL2
        DOUBLE PRECISION CPU_TIME_USED, ERR
        INTEGER N1, N2, NE, NESQR, RUN, JEC, TT, IX1, IX2
        INTEGER IRC, IH, IAES, IT, IA, IPD, IH2
        INTEGER IA2, IX, IP, IR, IVK, IAVK, IEC, IEND
        INTEGER ILS, ILAMBDA, MAX_ITER_NUM,DIM
        INTEGER IQT_RHT,IQTAQ,IQ,STRP,ITEMP,G,iter
        LOGICAL YAW
        DOUBLE PRECISION PI
        REAL(KIND=16) R(12900000)
        PARAMETER (PI = acos(-1.0d0))
        DOUBLE PRECISION TIME_START2, TIME_END2,WALLCLOCK
        INTEGER start_count, end_count, count_rate, count_max

        TOL = 1.0E-9
        TOL2 = 1.0E-13
        MAX_ITER_NUM=55
        YAW=.TRUE.
        G=4
        PRINT *, "//========================================//"
        PRINT *, "//=========NORM_V16_WEAR_CG.F CODE=========//"
        PRINT *, "//============== Paper used: ==============//"
        PRINT *, "//==J Giner - A fast version of 'CONTACT'==//"
        PRINT *, "//=====for normal problem calculations=====//"
        PRINT *, "//========================================//"
        IF (.NOT. YAW)THEN
        PRINT *, "//====ELLIPTICAL TRACTION DISTRIBUTION=====//"
        ELSE
        PRINT *, "//====POLYNOMICAL TRACTION DISTRIBUTION====//"
        END IF
        PRINT *, "//========================================//"
        PRINT *, "//============NORM + CG + E/P=============//"

        CALL READ_SCALAR_VARIABLES(E, NU, A, B, N1, N2)
```

```
      NE = N1 * N2
      NESQR = NE ** 2
      IH = 1
      IX1 = IH + NE
      IX2 = IX1 + N1
      IRC = IX2 + N2
      IAES = IRC + NE * 2
      IT = IAES + NE
      IA = IT + NESQR
      IPD = IA + NESQR
      IH2 = IPD + NE
      IA2 = IH2 + NE
      IX = IA2 + NESQR
      IP = IX + NE
      IR = IP + NE
      IVK = IR + NE
      IAVK = IVK + NE
      IEC=IAVK+NE
      IQ=IEC+NE
      ILS=IQ+NESQR
      IQTAQ=ILS+N2
      IQT_RHT=IQTAQ+NE*15
      ILAMBDA=IQT_RHT+NE
      ITEMP=ILAMBDA+NE
      IEND = ITEMP + NESQR
      PRINT*, 'Memory used:',IEND, 'elements'
      IF (IEND .LT. 12900000) THEN
        WRITE (*,*) 'Correct Allocation'
      ELSE
        WRITE (*,*) 'IEND is greater than maximum capacity for R.'
      END IF
      CALL ZEROS_VEC(IEND,R)
      CALL READ_VECTOR_AND_MATRIX(N1, N2, R(IH), R(IX1), R(IX2))
c     ¡CODE TIMMER STARTS!
      TIME_START2 = ETIME(TIME_ARRAY)
      CALL SYSTEM_CLOCK(COUNT_RATE=count_rate, COUNT_MAX=count_max)
      CALL SYSTEM_CLOCK(COUNT=start_count)
C     AES CREATION
      CALL CREATION_AES(N1, N2, A, B, NU, E, R(IAES), R(IX1), R(IX2))
C     MATRIX ROW-COLUMN CREATION
```

```
        CALL RC_CALC(N1, N2, NE, R(IRC))
C       A MATRIX CREATION
        CALL A_FORMAT(NE, N1, N2, R(IA), R(IAES), R(IRC))
C       P ELLIPTICAL DISTRIBUTION CALCULATION
        CALL VALUE_TO_VEC(N2, R(ILS),N1)
        IF (.NOT. YAW)THEN
          CALL CALC_Q_E(DIM,NE,N2,R(ILS),R(IQ))

          CALL COMPUTE_QT_A_Q(DIM, NE, R(IQ), R(IA), R(IQTAQ),
     & R(ITEMP))
        ELSE
          CALL CALC_Q_P(G,DIM,NE,N2,A,R(ILS),R(IQ))
          CALL COMPUTE_QT_A_Q(DIM, NE, R(IQ), R(IA), R(IQTAQ),
     & R(ITEMP))
        END IF
        CALL MULTIPLY_MAT_VEC2(NE, DIM, R(IQT_RHT), R(IH), R(IQ))
C       CONJUGATE GRADIENT METHOD
        CALL CGR_METHOD(DIM, R(IQTAQ), R(IQT_RHT), R(ILAMBDA), R(IR),
     & R(IVK), R(IAVK), TOL2, MAX_ITER_NUM)
        CALL MULTIPLY_MAT_VEC(NE, DIM, R(IPD), R(ILAMBDA), R(IQ))
        RUN = 1
        TT = 0
C       NEW PD AND PCA CALCULATION LOOP
  10  CONTINUE
        IF (RUN .EQ. 0) GOTO 20
        RUN = 0
        IF (TT .EQ. 1) THEN
          CALL ZEROS_MAT(NE,JEC, R(IT))
          CALL ZEROS_MAT(JEC,DIM, R(IQ))
          CALL ZEROS_MAT(DIM,DIM, R(IQTAQ))
          CALL ZEROS_VEC(JEC, R(IX))
          CALL ZEROS_MAT(JEC,JEC, R(IA2))
          CALL ZEROS_VEC(JEC, R(IH2))
          CALL ZEROS_VEC(DIM, R(IQT_RHT))
          CALL ZEROS_MAT(JEC,DIM, R(ITEMP))
        ELSE
          CALL ZEROS_VEC(DIM, R(IQT_RHT))
          CALL ZEROS_MAT(NE,DIM, R(IQ))
          CALL ZEROS_MAT(DIM,DIM, R(IQTAQ))
          CALL ZEROS_MAT(NE,DIM, R(ITEMP))
        END IF
```

```
        DIM=0
        TT = 1
        CALL JEC_VALUE(TOL, RUN, N2, STRP, JEC, NE, R(IT), R(IPD),
     & R(IEC),R(ILS))
        CALL ZEROS_VEC(NE, R(IPD))
        CALL MULTIPLY_MAT_VEC2(NE, JEC, R(IH2), R(IH), R(IT))
        CALL MAKEA2(JEC, NE, R(IA), R(IA2),R(IEC))
C       P ELLIPTICAL DISTRIBUTION CALCULATION
        IF (.not. YAW) THEN
          CALL CALC_Q_E(DIM,JEC,STRP,R(ILS),R(IQ))
          CALL COMPUTE_QT_A_Q(DIM, JEC, R(IQ), R(IA2), R(IQTAQ),
     & R(ITEMP))
        ELSE
          CALL CALC_Q_P(G,DIM,JEC,STRP,A,R(ILS),R(IQ))
          CALL COMPUTE_QT_A_Q(DIM, JEC, R(IQ), R(IA2), R(IQTAQ),
     & R(ITEMP))
        END IF
        CALL MULTIPLY_MAT_VEC2(JEC, DIM, R(IQT_RHT), R(IH2), R(IQ))
C       CONJUGATE GRADIENT METHOD
        CALL CGR_METHOD(DIM, R(IQTAQ), R(IQT_RHT), R(ILAMBDA), R(IR),
     & R(IVK), R(IAVK), TOL2, MAX_ITER_NUM)
        CALL MULTIPLY_MAT_VEC(JEC, DIM, R(IX), R(ILAMBDA), R(IQ))
        CALL MULTIPLY_MAT_VEC(NE, JEC, R(IPD), R(IX), R(IT))
        IF (RUN .EQ. 1) GOTO 10
  20  CONTINUE
        NF = 0.0D0
        X1P = 0.0D0
        X2P = 0.0D0
        CALL FORCE_CALC(N1, N2, R(IPD), R(IP), R(IX1), R(IX2), X1P,
     &      X2P, NF, A, B)
        ERR = (ABS(NF - 5.003728385561319D+4)/5.003728385561319D+4)*
     & 100.0D0
        TIME_END2 = ETIME(TIME_ARRAY)
        CALL SYSTEM_CLOCK(COUNT=end_count)
c       CALCULATE WALL-CLOCK TIME AND CPU TIME
        WALLCLOCK = REAL(end_count - start_count) / REAL(count_rate)
        CPU_TIME_USED = TIME_END2 - TIME_START2
        write ( *, * ) 'Wall-clock time: ', WALLCLOCK, 'seconds'
        write ( *, * ) 'CPU time used: ', CPU_TIME_USED ,
     & 'seconds'
        write ( *, * ) "//=======================================//"
```

```
        write ( *, * ) "Total Normal Force (N) value is:", NF, "N"
        write ( *, * ) "//=======================================//"
        write ( *, * ) "Relative error (NROM + E/P +",
     & "CG vs. NORM (Matlab))"
        write ( *, * )  ERR, "%"
        write ( *, * ) "//=======================================//"
        CALL compare_matrices(R(IP), R(IH), N1, N2)
        CALL SYSTEM('PAUSE')
      END

      SUBROUTINE READ_SCALAR_VARIABLES(E, NU, A, B, N1, N2)
C     ================================================================================
C     READING OF SCALAR VARIABLES` VALUES FROM FILES
C     ================================================================================
C     INPUTS (which are outputs itself):
C     |   E         : Young modulus [N/m^2]
C     |   nu        : Poisson`s ratio [·]
C     |   N1,N2     : Number of elements in the rolling and lateral directions
C     |   a         : Half size of the element - rolling direction [m]
C     |   b         : Half size of the element - lateral direction [m]
C     ================================================================================
        DOUBLE PRECISION E, NU, A, B
        INTEGER N1, N2
        INTEGER IERR

        OPEN(UNIT=1, FILE='E.txt', STATUS='OLD', IOSTAT=IERR)
        IF (IERR .ne. 0) STOP "Error opening E.txt"
        READ(1, *) E
        CLOSE(1)
        OPEN(UNIT=2, FILE='nu.txt', STATUS='OLD', IOSTAT=IERR)
        IF (IERR .ne. 0) STOP "Error opening nu.txt"
        READ(2, *) NU
        CLOSE(2)
        OPEN(UNIT=3, FILE='a.txt', STATUS='OLD', IOSTAT=IERR)
        IF (IERR .ne. 0) STOP "Error opening a.txt"
        READ(3, *) A
        CLOSE(3)
        OPEN(UNIT=4, FILE='b.txt', STATUS='OLD', IOSTAT=IERR)
        IF (IERR .ne. 0) STOP "Error opening b.txt"
        READ(4, *) B
        CLOSE(4)
```

```
      OPEN(UNIT=5, FILE='N1.txt', STATUS='OLD', IOSTAT=IERR)
      IF (IERR .ne. 0) STOP "Error opening N1.txt"
      READ(5, *) N1
      CLOSE(5)
      OPEN (UNIT=20, FILE='N2.txt', STATUS='OLD', IOSTAT=IERR)
      IF (IERR .ne. 0) STOP "Error opening N1.txt"
      READ(20, *) N2
      CLOSE(20)
    END

    SUBROUTINE READ_VECTOR_AND_MATRIX(N1, N2, H, X1, X2)
C     ======================================================================
C     READING OF VECTORS AND MATRICES VALUES FROM FILES
C     ======================================================================
C     INPUTS (which are outputs itself):
C     |   N1,N2     : Number of elements in the rolling and lateral directions
C     |   H         : Undeformed distance matrix N1xN2 [m]
C     |   x1        : Longitudinal coordinates of the elements in the PCA [m]
C     |   x2        : Lateral coordinates of the elements in the PCA [m]
C     ======================================================================
      INTEGER N1, N2, I, J, IERR
      DOUBLE PRECISION H(N1, N2), X1(N1), X2(N2)
      CHARACTER*80 LINE

      ! Read vector X1 from x1.txt
      OPEN(UNIT=7, FILE='x1.txt', STATUS='OLD', IOSTAT=IERR)
      IF (IERR /= 0) STOP "Error opening x1.txt"
      DO I = 1, N1
          READ(7, *) X1(I)
      END DO
      CLOSE(7)
      OPEN(UNIT=8, FILE='x2.txt', STATUS='OLD', IOSTAT=IERR)
      IF (IERR /= 0) STOP "Error opening x2.txt"
      DO I = 1, N2
          READ(8, *) X2(I)

      END DO
      CLOSE(8)
      OPEN(UNIT=9, FILE='h.txt', STATUS='OLD', IOSTAT=IERR)
      IF (IERR /= 0) THEN
          PRINT *, 'Error opening h.txt'
```

```
            STOP
          END IF
          DO I = 1, N1
              READ(9, *, IOSTAT=IERR) (H(I, J), J = 1, N2)
              IF (IERR /= 0) THEN
                  PRINT *, 'Error reading h.txt at row', I
                  CLOSE(9)
                  STOP
              END IF
          END DO
          CLOSE(9)
      END

      SUBROUTINE CREATION_AES(N1, N2, A, B, NU, E, AES, X1, X2)
C     ======================================================================
C     Aes CREATION
C     ======================================================================
C     INPUTS:
C     |    N1,N2   : Number of elements in the rolling and lateral directions
C     |    a       : Half size of the element - rolling direction [m]
C     |    b       : Half size of the element - lateral direction [m]
C     |    nu      : Poisson`s ratio [·]
C     |    E       : Young modulus [N/m^2]
C     |    F       : Flexibility coefficient
C     |    Aes     : Influence coefficient`s matrix for each element [N1xN2]
C     |    x1      : Longitudinal coordinates of the elements in the PCA [m]
C     |    x2      : Lateral coordinates of the elements in the PCA [m]
C     ======================================================================
C     WHAT IS BEING CALCULATED?:
C     |    Aes     : Influence coefficient`s matrix for each element [N1xN2]
C     ======================================================================
          INTEGER N1, N2, I, J
          DOUBLE PRECISION A, B, NU, E, F, X, Y, AUX
          DOUBLE PRECISION AES(N1, N2), X1(N1), X2(N2)
          DOUBLE PRECISION PI

          PI = ACOS(-1.0D0)
          AUX = 2.0 * ((1.0 - NU**2) / (PI * E))
          DO I = 1, N1
            X = X1(I) - X1(1)
            DO J = 1, N2
```

```
            Y = X2(J) - X2(1)
            CALL CT_BC(X, Y, A, B, F)
            AES(I, J) = F * AUX
          END DO
        END DO
      END

      SUBROUTINE CT_BC(X, Y, A, B, F)
C     ================================================================================
C     FLEXIBILITY COEFFICIENTS CALCULATION
C     ================================================================================
C     Calculate the flexibility coeff. A, which relates the displacement
C     at (x,y) due to an applied unitary load on a 2ax2b origin-centered
C     rectangle. Love (1929), Johnson page 54
C     Note: Multiply by (1-poisson^2)/pi/E
C     ================================================================================
C     INPUTS:
C     |    x      : x-coord where displacement is measured [m]
C     |    y      : y-coord where displacement is measured [m]
C     |    a      : Half size of the element - rolling direction [m]
C     |    b      : Half size of the element - lateral direction [m]
C     ================================================================================
C     OUTPUT:
C     |    F      : Flexibility coefficient
C     ================================================================================
        DOUBLE PRECISION X, Y, A, B, F

       F = (X + A) * LOG(((Y + B) + SQRT((Y + B)**2 + (X + A)**2)) /
     & ((Y - B) + SQRT((Y - B)**2 + (X + A)**2))) +
     &   (Y + B) * LOG(((X + A) + SQRT((Y + B)**2 + (X + A)**2)) /
     & ((X - A) + SQRT((Y + B)**2 + (X - A)**2))) +
     &   (X - A) * LOG(((Y - B) + SQRT((Y - B)**2 + (X - A)**2)) /
     & ((Y + B) + SQRT((Y + B)**2 + (X - A)**2))) +
     &   (Y - B) * LOG(((X - A) + SQRT((Y - B)**2 + (X - A)**2)) /
     & ((X + A) + SQRT((Y - B)**2 + (X + A)**2)))
      END

      SUBROUTINE RC_CALC(N1, N2, NE, RC)
C     ================================================================================
C     ROW - COLUMN MATRIX
C     ================================================================================
```

```
C     INPUTS:
C     |    N1      : Number of elements in the rolling direction
C     |    N2      : Number of elements in the lateral direction
C     |    NE      : Dimension of the problem
C     |    RC      : Matrix that stores the position of PCA elements [NEx2]
C     ========================================================================
C     WHAT IS BEING CALCULATED?:
C     |    RC      : Matrix that stores the position of PCA elements [NEx2]
C     ========================================================================
        INTEGER N1, N2, NE, I, J, V
        DOUBLE PRECISION RC(NE, 2)

        V = 1
        DO I = 1, N2
          DO J = 1, N1
            IF (V .LE. NE) THEN
              RC(V, 1) = J
              RC(V, 2) = I
              V = V + 1
            END IF
          END DO
        END DO
      END

      SUBROUTINE A_FORMAT(NE, N1, N2, A, AES, RC)
C     ========================================================================
C     CREATION OF MATRIX A
C     ========================================================================
C     INPUTS:
C     |    NE      : Dimension of the problem
C     |    N2      : Number of elements in the lateral direction
C     |    A       : Matrix containing the influence between all elements of the PCA [NExNE]
C     |    Aes     : Influence coefficient`s matrix for each element [N1xN2]
C     |    RC      : Matrix that stores the position of PCA elements [NEx2]
C     ========================================================================
C     WHAT IS BEING CALCULATED?:
C     |    A       : Matrix containing the influence between all elements of the PCA [NExNE]
C     ========================================================================
        INTEGER NE, N2, I, J, Ro, Co
        DOUBLE PRECISION A(NE, NE), AES(N1, N2), RC(NE, 2)
```

```
        DO I = 1, NE
          DO J = 1, NE
            Ro = ABS(RC(J, 1) - RC(I, 1))+1
            Co = ABS(RC(J, 2) - RC(I, 2))+1
            A(I, J) = AES(Ro, Co)
          END DO
        END DO
      END

      SUBROUTINE CGR_METHOD(M, A, H, P, R, VK, AVK, T, IT)
c     //=====================================================================
c     //  | CONJUGATE GRADIENT METHOD
c     //=====================================================================
c     //  | This code aims to caclulate the solucion P of a linear system of
c     //  | equations like AP=H using the Conjugate Gradeint Method.
c     //
c     //  | Conditions needed to found a solution or being able to use the code:
c     //  | - A must be a symmetric matrix -> A = A^T
c     //  | - A must be possitive definite -> (P^T)AP > 0 ∀ P ∈ R
c     //=====================================================================
c     //  | INPUTS:
c     //  |    M        : Dimension of the problem
c     //  |    A        : Matrix which contains the influence between all elements of the PCA
(also the A matrix of the linear system)
c     //  |    H        : Vector representing the right-hand side of the linear system
c     //  |    P        : Solution of the linear system
c     //  |    R        : Residual vector used to track the difference between the current
solution and the true solution
c     //  |    VK       : Direction vector along which the solution is improved
c     //  |    AVK      : Result of multiplying the matrix A by the direction vector Vk
c     //  |    T        : Threshold below which the solution is considered accurate enough
c     //  |    IT       : The number of iterations used
c     //=====================================================================
c     //  | WHAT IS BEING CALCULATED?:
c     //  |    P        : Solution of the linear system
c     //=====================================================================
        INTEGER M, IT
        DOUBLE PRECISION ALPHA, BETA, RSOLD, T, PROD, BB
        DOUBLE PRECISION A(M, M), H(M), P(M)
        DOUBLE PRECISION R(M), VK(M), AVK(M)

        CALL MATVEC(AVK, P, M, A)
```

```
  CALL VEC_SUB(R, H, AVK, M)
  CALL VEC_COPY(VK, R, M)
  CALL DOT_PRODUCT(R, R, M, PROD)
  CALL DOT_PRODUCT(H, H, M, BB)  ! Calcula bb = H'*H
  RSOLD=PROD
  IT=0
  DO WHILE (PROD .GT. T**2 * BB)
    ! Código del bucle que se ejecutará mientras la condición se cumpla
    CALL MATVEC(AVK, VK, M, A)
    CALL DOT_PRODUCT(VK, AVK, M, PROD)
    ALPHA = RSOLD / PROD
    CALL VEC_ADD(P, P, VK, M, ALPHA)
    CALL VEC_ADD(R, R, AVK, M, (-1*ALPHA))
    CALL DOT_PRODUCT(R, R, M, PROD)
    BETA = PROD / RSOLD
    CALL VEC_ADD(VK, R, VK, M, BETA)
    RSOLD = PROD
    IT=IT+1
END DO
END

SUBROUTINE DOT_PRODUCT(X, Y, SIZE, PROD)
  INTEGER SIZE, I
  DOUBLE PRECISION X(SIZE), Y(SIZE)
  DOUBLE PRECISION PROD
  PROD = 0.0
  DO I = 1, SIZE
    PROD = PROD + X(I) * Y(I)
  END DO
END

SUBROUTINE MATVEC(Y, X, SIZE, A)
  INTEGER SIZE, I, J
  DOUBLE PRECISION  Y(SIZE), X(SIZE), A(SIZE, SIZE)
  DO I = 1, SIZE
    Y(I) = 0.0D0
    DO J = 1, SIZE
      Y(I) = Y(I) + A(I, J) * X(J)
    END DO
  END DO
END
```

```
      SUBROUTINE VEC_ADD(RESULT, X, Y, SIZE, ALPHA)
        INTEGER SIZE, I
        DOUBLE PRECISION  RESULT(SIZE), X(SIZE), Y(SIZE)
        DOUBLE PRECISION ALPHA
        DO I = 1, SIZE
          RESULT(I) = X(I) + ALPHA * Y(I)
        END DO
      END

      SUBROUTINE VEC_SUB(RESULT, X, Y, SIZE)
        INTEGER SIZE, I
        DOUBLE PRECISION RESULT(SIZE), X(SIZE), Y(SIZE)
        DO I = 1, SIZE
          RESULT(I) = X(I) - Y(I)
        END DO
      END

      SUBROUTINE VEC_COPY(DEST, SRC, SIZE)
        INTEGER SIZE, I
        DOUBLE PRECISION DEST(SIZE), SRC(SIZE)
        DO I = 1, SIZE
          DEST(I) = SRC(I)
        END DO
      END

      SUBROUTINE ZEROS_VEC(SIZE, v)
C     ============================================================================
C     INITIALIZATION OF VECTORS
C     ============================================================================
C     INPUTS:
C     |    size    : Dimension of the problem
C     |    v       : Vector to be initialized
C     ============================================================================
C     WHAT IS BEING CALCULATED?:
C     |    v       : Vector initialized
C     ============================================================================
        INTEGER SIZE, I
        DOUBLE PRECISION v(SIZE)

        DO I = 1, SIZE
```

```
          v(I) = 0.0
        END DO
      END

      SUBROUTINE ZEROS_MAT(SIZE1,SIZE2, MAT)
C     ================================================================================
C     INITIALIZATION OF MATRICES
C     ================================================================================
C     INPUTS:
C     |    size1   : Rows of the matrix
C     |    size2   : Columns of the matrix
C     |    MAT     : Matrix to be initialized
C     ================================================================================
C     WHAT IS BEING CALCULATED?:
C     |    MAT     : Matrix initialized
C     ================================================================================


        INTEGER SIZE1, SIZE2, J, I
        DOUBLE PRECISION MAT(SIZE1,SIZE2)
        DO I = 1, SIZE1
          DO J=1, SIZE2
            MAT(I,J) = 0.0
          END DO
        END DO
      END

      SUBROUTINE JEC_VALUE(TOL, RUN, N2,STRP, JEC, NE, T, PD, EC,LS)
C     ================================================================================
C     JEC`S VALUE CALCULATION
C     ================================================================================
C     INPUTS:
C      |     TOL      : Tolerance to differentiate elements in contact against those not in
contact
C     |    RUN     : Logical value (0: The real contact area has been found. 1: The contact
area must be re-calulated)
C     |    JEC     : Number of elements in contact in the real contact area
C     |    NE      : Total dimension of the problem
C      |     T        : Matrix that stores (with unitary value) the positions to be used in
matrices and vectors to calculate the new values after updating JEC
C     |    PD      : Normal traction vector NEx1 [N/m^2]
```

```
C     |    EC      : Vector that stores the position in Influence coefficients` matrix A to
be considered based on PD`s value.
C     ========================================================================
C     WHAT IS BEING CALCULATED?:
C     |    RUN     : Logical value (0: The real contact area has been found. 1: The contact
area must be re-calulated)
C     |    JEC     : Number of elements in contact in the real contact area
C     |    T       : Matrix that stores (with unitary value) the positions to be used in
matrices and vectors to calculate the new values after updating JEC
C     |    EC      : Vector that stores the position in Influence coefficients` matrix A to
be considered based on PD`s value.
C     ========================================================================
        INTEGER RUN, JEC, NE, N,CHK,STRP,N2
        DOUBLE PRECISION TOL
        DOUBLE PRECISION LS(N2),T(NE,NE), PD(NE), EC(NE)


        JEC=0
        CHK=0
        STRP=0
        DO N = 1, NE
          IF (PD(N) .LT. TOL) THEN
            CHK=0
              IF ((PD(N) .LT. (-TOL)) .AND.
     & (RUN .EQ. 0)) THEN
                RUN = 1
              END IF
          ELSE
            IF ((CHK .EQ. 0) .AND. (STRP .LT. N2)) THEN
              STRP=STRP+1
              CHK=1
              LS(STRP)=1;
            ELSE IF ((CHK .EQ. 1) .AND. (STRP .LE. N2)) THEN
                LS(STRP)=LS(STRP)+1;
                IF (LS(STRP) .EQ. N1)THEN
                CHK=0
                END IF
            END IF
                  JEC=JEC + 1
                  T(N,JEC)= 1.0
                  EC(JEC)=N
          END IF
        END DO
```

```
      END

      SUBROUTINE MULTIPLY_MAT_VEC(R, C, RE, V, A)
C     ============================================================================
C     MULTIPLY A VECTOR AND A MATRIX (RE=A*V)
C     ============================================================================
C     INPUTS:
C     |    c       : Columns
C     |    r       : Rows
C     |    Re      : Result of the multiplication
C     |    v       : vector to multiply
C     |    A       : matrix to multiply
C     ============================================================================
C     WHAT IS BEING CALCULATED?:
C     |    Re      : Result of the multiplication
C     ============================================================================
        INTEGER C, R, I, J
        DOUBLE PRECISION RE(R), V(C), A(R, C)
        DO I = 1, R
          DO J = 1, C
            RE(I) = RE(I) + A(I, J) * V(J)
          END DO
        END DO
      END

      SUBROUTINE MULTIPLY_MAT_VEC2(R, C, RE, V, A)
C     ============================================================================
C     MULTIPLY A VECTOR AND A MATRIX (RE=AT*V)
C     ============================================================================
C     INPUTS:
C     |    c      : Columns
C     |    r      : Rows
C     |    Re     : Result of the multiplication
C     |    v      : vector to multiply
C     |    A      : matrix to multiply
C     ============================================================================
C     WHAT IS BEING CALCULATED?:
C     |    Re     : Result of the multiplication of A transposed per v
C     ============================================================================
        INTEGER C, R, I, J
        DOUBLE PRECISION RE(C), V(R), A(R, C)
```

```
        DO I = 1, R
          DO J = 1, C
            RE(J) = RE(J) + A(I, J) * V(I)
          END DO
        END DO
      END

      SUBROUTINE MAKEA2(JEC, NE, A, A2,EC)
C     ================================================================================
C     CALCULATION OF THE A2 MATRIX
C     ================================================================================
C     INPUTS:
C     |    jec     : New dimension of the problem (after tension value evaluation)
C     |    NE      : Dimension of the problem
C     |    T       : Matrix containing the positions of A to consider (after tension value
evaluation) [NExNE]
C     |    A       : Matrix containing the influence coefficients between all elements of the
PCA [NExNE]
C     |    A2      : Matrix containing the influence coefficients between all elements of the
updated PCA (after tension value evaluation) [jec x jec]
C     |    EC      : Vector that stores the position in Influence coefficients` matrix A to
be considered in A2 based on PD`s value. (Elements in contact area)
C     ================================================================================
C     WHAT IS BEING CALCULATED?:
C     |    A2      : Matrix containing the influence between all elements of the updated PCA
(after tension value evaluation) [jec x jec]
C     ================================================================================
        INTEGER JEC, NE, I, J
        DOUBLE PRECISION A(NE, NE), A2(JEC, JEC)
        DOUBLE PRECISION EC(NE)

        DO J =1,JEC
          DO I=1,JEC
            A2(I,J)=A(INT(EC(I)),INT(EC(J)))
          END DO
        END DO
      END

      SUBROUTINE FORCE_CALC(N1, N2, PD, P, X1, X2, X1P,
     & X2P, NF, A, B)
C     ================================================================================
C     CALCULATION OF THE TOTAL FORCE
```

```
C     ======================================================================
C     INPUTS:
C     |    N1,N2   : Number of elements in the rolling and lateral directions
C     |    PD      : Normal traction vector NEx1 [N/m^2]
C     |    P     : Normal traction N1xN2 matrix [N/m^2]
C     |    X1      : Longitudinal coordinates of the elements in the PCA [m]
C     |    X2      : Lateral coordinates of the elements in the PCA [m]
C     |    X1P,X2P : Coordinates of the centre of pressure [m]
C     |    NF      : Total normal force [N]
C     |    A       : Half size of the element - rolling direction [m]
C     |    B       : Half size of the element - lateral direction [m]
C     ======================================================================
C     WHAT IS BEING CALCULATED?:
C     |    P     : Normal traction N1xN2 matrix [N/m^2]
C     |    NF      : Total normal force [N]
C     |    X1P,X2P : Coordinates of the centre of pressure [m]
C     ======================================================================
        INTEGER N1, N2, I, J, M, N
        DOUBLE PRECISION PD(NE), P(N1, N2)
        DOUBLE PRECISION X1(N1), X2(N2)
        DOUBLE PRECISION X1P, X2P, NF, A, B
        DOUBLE PRECISION SUMA,SIGMA

        SIGMA=0.0
        DO N = 1, N2
          DO M = 1, N1
            P(M, N) = PD((N - 1) * N1 + M)
          END DO
        END DO
C Addition of all elements in pa to calculate the total traction
        DO N=1,N2
          DO M=1,N1
            SIGMA=SIGMA+P(M,N)
          END DO
        END DO
C Calculate X1P
        SUMA = 0.0
        DO I = 1, N1
          DO J = 1, N2
              SUMA = SUMA + P(I, J)
          END DO
```

```
            X1P = X1P + SUMA * X1(I)
          END DO
          X1P = X1P / SIGMA
C Calculate X2P
          SUMA = 0.0
          DO I = 1, N2
            DO J = 1, N1
                SUMA = SUMA + P(J, I)
            END DO
                X2P = X2P + SUMA * X2(I)
          END DO
          X2P = X2P / SIGMA
C Calculate NF
          NF = SIGMA * 4 * A * B
        END

        SUBROUTINE CALC_Q_E(DIM,SIZE,STRP,LS,QE)
C       ======================================================================
C       CALCULATION OF MATRIX Q (ELLIPTICAL BASIS)
C       ======================================================================
C       INPUTS:
C       |    DIM    : Dimension of the matrix Q
C       |    SIZE   : Size of the matrix Q
C       |    STRP   : Number of strips (subdivisions)
C       |    LS     : Array containing the lengths of the strips
C
C       OUTPUTS:
C       |    Q     : Matrix containing the elliptical basis
C                      for the stress distribution
C       ======================================================================
          INTEGER DIM, SIZE,I1,I2,I3,J,STRP
          DOUBLE PRECISION LS(STRP), QE(SIZE,DIM)
          DOUBLE PRECISION XES

          NUM=0;
          I3=0;
          DO J=1,STRP
            I2=INT(LS(J))
            DO I1=1,I2
              XES=-1*(2 * (I2/2 - I1 + 1) - 1)
              QE(I1+I3,2*J-1)=1
```

```
            QE(I1+I3,2*J)=SQRT(1 - (XES / I2)**2)
          END DO
          I3=I3+INT(LS(J))
        END DO
        DIM=2*STRP
      END

      SUBROUTINE COMPUTE_QT_A_Q(DIM, SIZE, QE, A, QT_A_Q,TEMP)
C     ======================================================================
C     COMPUTATION OF MATRIX QT_A_Q
C     ======================================================================
C     INPUTS:
C     |    DIM       : Dimension of matrices
C     |    SIZE      : Size of matrices
C     |    QE        : Matrix containing the elliptical or polynomial basis
C                      functions
C     |    A         : Matrix A of the system
C
C     OUTPUTS:
C     |    QT_A_Q    : Matrix result of QT * A * Q
C     |    TEMP      : Temporary matrix used for the calculation of QT * A * Q
C     ======================================================================
        INTEGER SIZE, DIM, I, J, K,L
        DOUBLE PRECISION QE(SIZE, DIM), A(SIZE, SIZE)
        DOUBLE PRECISION QT_A_Q(DIM, DIM),TEMP(SIZE,DIM)

C Compute TEMP = A * Q
        DO K = 1, SIZE
          DO J = 1, DIM
            IF (QE(K, J) .NE. 0.0D0) THEN
              DO L = 1, SIZE
                  TEMP(L, J) = TEMP(L, J) + A(L, K) * QE(K, J)
              END DO
            END IF
          END DO
        END DO
C Compute QT_A_Q = Q_T * TEMP
        DO L = 1, SIZE
          DO I = 1, DIM
            IF (QE(L, I) .NE. 0.0D0) THEN
              DO J = 1, DIM
```

```
            IF (TEMP(L, J) .NE. 0.0D0) THEN
              QT_A_Q(I, J) = QT_A_Q(I, J) + QE(L, I) *
     & TEMP(L, J)
            END IF
          END DO
        END IF
      END DO
    END DO
  END

  SUBROUTINE VALUE_TO_VEC(SIZE,V,VALUE)
C     ======================================================================
C     ASSIGN CONSTANT VALUE TO VECTOR V
C     ======================================================================
C     INPUTS:
C     |    SIZE      : Size of the vector
C     |    VALUE     : Constant value to assign to all elements
C
C     OUTPUTS:
C     |    V         : Vector of size SIZE with all elements assigned VALUE
C     ======================================================================
      INTEGER VALUE,I,SIZE
      DOUBLE PRECISION V(SIZE)

      DO I=1,SIZE
        V(I)=VALUE
      END DO
  END

  SUBROUTINE CALC_Q_P(G,DIM,SIZE,STRP,A,LS,QP)
C     ======================================================================
C     CALCULATION OF MATRIX B (POLYNOMIAL BASIS)
C     ======================================================================
C     INPUTS:
C     |    G         : Degree of the polynomial
C     |    DIM       : Dimension of the matrix B
C     |    SIZE      : Size of the matrix B
C     |    STRP      : Number of strips (subdivisions)
C     |    A         : Scaling factor for polynomial basis functions
C     |    LS        : Array containing the lengths of the strips
C     OUTPUTS:
```

```
C     |    B         : Matrix containing the polynomial basis functions
C     ======================================================================
        INTEGER DIM, SIZE,I1,I2,I3,J,STRP,G,M
        DOUBLE PRECISION XES,A
        DOUBLE PRECISION QP(SIZE,DIM), LS(STRP)

        I3=0;
        DO J=1,STRP
          I2=INT(LS(J))
          DO I1=1,I2
            XES=-1*(2 * (I2/2 - I1 + 1) - 1)
            DO M=0,G
              QP(I1+I3,(G+1)*J-(G-M))=(XES*A)**M;
            END DO
          END DO
          I3=I3+INT(LS(J))
        END DO
        DIM=(G+1)*STRP
      END

      SUBROUTINE COMPARE_MATRICES(MATRIX, REF_MATRIX, N1, N2)
C     ======================================================================
C     COMPARE TWO MATRICES AND CALCULATE ERRORS
C     ======================================================================
C     INPUTS:
C     |    MATRIX        : Matrix to be compared [n1 x n2]
C     |    REF_MATRIX    : Reference matrix to compare against [n1 x n2]
C     |    N1, N2        : Dimensions of the matrices
C
C     OUTPUTS:
C     |    MAX_ABS_ERROR : Maximum absolute error between the two matrices
C     |    MAX_REL_ERROR : Maximum relative error between the two matrices
C
C     FUNCTIONALITY:
C     |    Reads the reference matrix from a file (REF.TXT), compares it
C     |    element by element with the provided matrix, and computes the
C     |    maximum absolute and relative errors.
C     ======================================================================
        integer n1, n2, i, j
        DOUBLE PRECISION matrix(n1, n2), ref_matrix(n1, n2)
        DOUBLE PRECISION max_abs_error, max_rel_error, abs_error, rel_error
```

```
      max_abs_error = 0.0d0
      max_rel_error = 0.0d0
C     Read the reference matrix from the file REF.TXT
      open(unit=10, file='ref.txt', status='old')
      do i = 1, n1
         read(10,*) (ref_matrix(i, j), j = 1, n2)
      enddo
      close(10)
C     Compare the matrices
      do i = 1, n1
         do j = 1, n2
            abs_error = abs(matrix(i,j) - ref_matrix(i,j))
            if (ref_matrix(i,j) .ne. 0.0d0) then
               rel_error = abs_error / abs(ref_matrix(i,j))
            else
               rel_error = 0.0d0
            endif
            if (abs_error > max_abs_error) max_abs_error = abs_error
            if (rel_error > max_rel_error) max_rel_error = rel_error
         enddo
      enddo
C     Print the maximum absolute and relative errors
      print*, 'Max absolute error:', max_abs_error/1d+6
      print*, 'Max relative error:', max_rel_error
      return
    end
```

1.6. BCCG (K)

```
    PROGRAM NORMAL_CONTACT
C   The following code computes the normal traction distribution. Non-Hertzian model.
C   This is the fastest version
C   ========================================================================
C   INPUTS:
C   |   E       : Young modulus [N/m^2]
C   |   NU      : Poisson`s ratio [·]
C   |   N1,N2   : Number of elements in the rolling and lateral directions
C   |   H       : Undeformed distance matrix N1xN2 [m]
C   |   X1      : Longitudinal coordinates of the elements in the PCA [m]
C   |   X2      : Lateral coordinates of the elements in the PCA [m]
C   |   A       : Half size of the element - rolling direction [m]
```

```
C     |    B       : Half size of the element - lateral direction [m]
C     OUTPUTS:
C     |    P     : Normal traction N1xN2 matrix [N/m^2]
C     |    NF      : Total normal force [N]
C     |    X1P,X2P : Coordinates of the centre of pressure [m]
C     ========================================================================
C     PCA Diagram:
C                     (Lateral)
C                     A X2
C     ---------------|------------------
C     |  |  |  |  |  |  |  |  |  |44|
C     ---------------------------------
C     |  |  |  |  |  |  |  |  |  |  |
C     ---------------------------------------> X1 (Rolling)
C     |12|  |  |  |  |  |  |  |  |  |
C     ---------------------------------
C     | 1| 2| 3| 4| 5| 6| 7| 8| 9|10|11|
C     ---------------------------------
C
C     ========================================================================
        REAL TIME_ARRAY(2)
        double precision E, NU, A, B, NF, X1P, X2P, TOL, TOL2,TOL3
        DOUBLE PRECISION CPU_TIME_USED,CPU_TIME_USED1, ERR
        INTEGER N1, N2, NE, NESQR, RUN, JEC, TT, IX1, IX2
        INTEGER IRC, IH, IAES, IA, IPD, IXBND,IMSKBND
        INTEGER IP, IR, IV, IXP, IRP, IEND,IUPD
        INTEGER MAX_ITER_NUM, IG,K_INN,IQ
        DOUBLE PRECISION PI, R(2600000)
        PARAMETER (PI = 3.141592653589793D0)
        DOUBLE PRECISION TIME_START2, TIME_END2
        INTEGER start_count, end_count, count_rate, count_max
        DOUBLE PRECISION WALLCLOCK

        TOL = 1.0E-9
        TOL2 = 1.0E-13
        MAX_ITER_NUM=20
        K_INN=2
        TOL3=1.0E-6
        PRINT *, "//=========================================//"
        PRINT *, "//============ BCCG_V16.F CODE ============//"
        PRINT *, "//============== Paper used: ==============//"
```

```
      PRINT *, "//=E.A.H. Vollebregt-The Bound-Constrained=//"
      PRINT *, "//======Conjugate Gradient Method for======//"
      PRINT *, "//==========Non-negative Matrices==========//"
      PRINT *, "//=========================================//"
      PRINT *, "//======== BCCG(2) ===== MAX_in=20 ========//"

     CALL READ_SCALAR_VARIABLES(E, NU, A, B, N1, N2)

      NE = N1 * N2
      NESQR = NE ** 2
      IH = 1
      IX1 = IH + NE
      IX2 = IX1 + N1
      IRC = IX2 + N2
      IAES = IRC + NE * 2
      IA = IAES + NE
      IPD = IA + NESQR
      IR = IPD + NE
      IV = IR + NE
      IG = IV + NE
      IQ=IG + NE
      IP = IQ + NE
      IUPD=IP+NE
      IXP = IUPD + MAX_ITER_NUM
      IRP = IXP + NE
      IXBND = IRP + NE
      IMSKBND= IXBND + NE
      IXREL=IMSKBND+NE
      IEND = IXREL + NE

      PRINT*, 'Memory used:',IEND, 'elements'
      IF (IEND .LT. 2600000) THEN
        WRITE (*,*) 'Correct Allocation'
      ELSE
        WRITE (*,*) 'IEND is greater than maximum capacity for R.'
      END IF
      CALL READ_VECTOR_AND_MATRIX(N1, N2, R(IH), R(IX1), R(IX2))
c     ¡CODE TIMMER STARTS!
      TIME_START2 = ETIME(TIME_ARRAY)
      CALL SYSTEM_CLOCK(COUNT_RATE=count_rate, COUNT_MAX=count_max)
      CALL SYSTEM_CLOCK(COUNT=start_count)
```

```
C       AES CREATION
        CALL CREATION_AES(N1, N2, A, B, NU, E, R(IAES), R(IX1), R(IX2))
C       MATRIX ROW-COLUMN CREATION
        CALL RC_CALC(N1, N2, NE, R(IRC))
C       A MATRIX CREATION
        CALL A_FORMAT(NE, N1, N2, R(IA), R(IAES), R(IRC))
C       CONJUGATE GRADIENT METHOD
         CALL BCCG(K_INN, R(IA), R(IH), R(IPD), R(IG), R(IR),
     & R(IV), R(IQ), R(IXP), R(IRP), TOL3, MAX_ITER_NUM,
     & R(IUPD), R(IXBND), R(IMSKBND), R(IXREL), NE)
        NF = 0.0
        X1P = 0.0
        X2P = 0.0
        CALL FORCE_CALC(N1, N2, R(IPD), R(IP), R(IX1), R(IX2), X1P,
     &      X2P, NF, A, B)
        ERR = (ABS(NF - 5.003728385561319D+4)/5.003728385561319D+4)*
     & 100.0D0
        TIME_END2 = ETIME(TIME_ARRAY)
        CALL SYSTEM_CLOCK(COUNT=end_count)
c       CALCULATE WALL-CLOCK TIME AND CPUT TIME
        WALLCLOCK = REAL(end_count - start_count) / REAL(count_rate)
        CPU_TIME_USED = TIME_END2 - TIME_START2
        write ( *, * ) 'Wall-clock time: ', WALLCLOCK, 'seconds'
        write ( *, * ) 'CPU time used: ', CPU_TIME_USED ,
     & 'seconds'
        write ( *, * ) "//=======================================//"
        write ( *, * ) "Total Normal Force (N) value is:", NF, "N"
        write ( *, * ) "//=======================================//"
        write ( *, * ) "Relative error ",
     & "(BCCG (Fortran) vs. NORM (Matlab))"
        write ( *, * )  ERR, "%"
        write ( *, * ) "//=======================================//"
        CALL compare_matrices(R(IP), R(IH), N1, N2)
        CALL SYSTEM('PAUSE')
      END

      SUBROUTINE READ_SCALAR_VARIABLES(E, NU, A, B, N1, N2)
C     ========================================================================
C     READING OF SCALAR VARIABLES` VALUES FROM FILES
C     ========================================================================
C     INPUTS (which are outputs itself):
```

```
C     |   E         : Young modulus [N/m^2]
C     |   nu        : Poisson`s ratio [·]
C     |   N1,N2     : Number of elements in the rolling and lateral directions
C     |   a         : Half size of the element - rolling direction [m]
C     |   b         : Half size of the element - lateral direction [m]
C     ======================================================================
        DOUBLE PRECISION E, NU, A, B
        INTEGER N1, N2
        INTEGER IERR

        OPEN(UNIT=1, FILE='E.txt', STATUS='OLD', IOSTAT=IERR)
        IF (IERR .ne. 0) STOP "Error opening E.txt"
        READ(1, *) E
        CLOSE(1)
        OPEN(UNIT=2, FILE='nu.txt', STATUS='OLD', IOSTAT=IERR)
        IF (IERR .ne. 0) STOP "Error opening nu.txt"
        READ(2, *) NU
        CLOSE(2)
        OPEN(UNIT=3, FILE='a.txt', STATUS='OLD', IOSTAT=IERR)
        IF (IERR .ne. 0) STOP "Error opening a.txt"
        READ(3, *) A
        CLOSE(3)
        OPEN(UNIT=4, FILE='b.txt', STATUS='OLD', IOSTAT=IERR)
        IF (IERR .ne. 0) STOP "Error opening b.txt"
        READ(4, *) B
        CLOSE(4)
        OPEN(UNIT=5, FILE='N1.txt', STATUS='OLD', IOSTAT=IERR)
        IF (IERR .ne. 0) STOP "Error opening N1.txt"
        READ(5, *) N1
        CLOSE(5)
        OPEN (UNIT=20, FILE='N2.txt', STATUS='OLD', IOSTAT=IERR)
        IF (IERR .ne. 0) STOP "Error opening N1.txt"
        READ(20, *) N2
        CLOSE(20)
      END

      SUBROUTINE READ_VECTOR_AND_MATRIX(N1, N2, H, X1, X2)
C     ======================================================================
C     READING OF VECTORS AND MATRICES VALUES FROM FILES
C     ======================================================================
C     INPUTS (which are outputs itself):
```

```
C     |   N1,N2     : Number of elements in the rolling and lateral directions
C     |   H         : Undeformed distance matrix N1xN2 [m]
C     |   x1        : Longitudinal coordinates of the elements in the PCA [m]
C     |   x2        : Lateral coordinates of the elements in the PCA [m]
C     ======================================================================
        INTEGER N1, N2, I, J, IERR
        DOUBLE PRECISION H(N1, N2), X1(N1), X2(N2)

        ! Read vector X1 from x1.txt
        OPEN(UNIT=7, FILE='x1.txt', STATUS='OLD', IOSTAT=IERR)
        IF (IERR /= 0) STOP "Error opening x1.txt"
        DO I = 1, N1
            READ(7, *) X1(I)
        END DO
        CLOSE(7)
        OPEN(UNIT=8, FILE='x2.txt', STATUS='OLD', IOSTAT=IERR)
        IF (IERR /= 0) STOP "Error opening x2.txt"
        DO I = 1, N2
            READ(8, *) X2(I)

        END DO
        CLOSE(8
        OPEN(UNIT=9, FILE='h.txt', STATUS='OLD', IOSTAT=IERR)
        IF (IERR /= 0) THEN
            PRINT *, 'Error opening h.txt'
            STOP
        END IF
        DO I = 1, N1
            READ(9, *, IOSTAT=IERR) (H(I, J), J = 1, N2)  ! Adjust J range as needed
            IF (IERR /= 0) THEN
                PRINT *, 'Error reading h.txt at row', I
                CLOSE(9)
                STOP
            END IF
        END DO
        CLOSE(9)
      END

      SUBROUTINE CREATION_AES(N1, N2, A, B, NU, E, AES, X1, X2)
C     ======================================================================
C     Aes CREATION
```

```
C     ======================================================================
C     INPUTS:
C     |    N1,N2   : Number of elements in the rolling and lateral directions
C     |    a       : Half size of the element - rolling direction [m]
C     |    b       : Half size of the element - lateral direction [m]
C     |    nu      : Poisson`s ratio [·]
C     |    E       : Young modulus [N/m^2]
C     |    F       : Flexibility coefficient
C     |    Aes     : Influence coefficient`s matrix for each element [N1xN2]
C     |    x1      : Longitudinal coordinates of the elements in the PCA [m]
C     |    x2      : Lateral coordinates of the elements in the PCA [m]
C     ======================================================================
C     WHAT IS BEING CALCULATED?:
C     |    Aes     : Influence coefficient`s matrix for each element [N1xN2]
C     ======================================================================
      INTEGER N1, N2, I, J
      DOUBLE PRECISION A, B, NU, E, F, X, Y, AUX
      DOUBLE PRECISION AES(N1, N2), X1(N1), X2(N2)
      DOUBLE PRECISION PI

      PI = ACOS(-1.0)
      AUX = 2.0 * ((1.0 - NU**2) / (PI * E))
      DO I = 1, N1
        X = X1(I) - X1(1)
        DO J = 1, N2
            Y = X2(J) - X2(1)
            CALL CT_BC(X, Y, A, B, F)
            AES(I, J) = F * AUX
        END DO
      END DO
      END

      SUBROUTINE CT_BC(X, Y, A, B, F)
C     ======================================================================
C     FLEXIBILITY COEFFICIENTS CALCULATION
C     ======================================================================
C     Calculate the flexibility coeff. A, which relates the displacement
C     at (x,y) due to an applied unitary load on a 2ax2b origin-centered
C     rectangle. Love (1929), Johnson page 54
C     Note: Multiply by (1-poisson^2)/pi/E
C     ======================================================================
```

```
C     INPUTS:
C     |    x      : x-coord where displacement is measured [m]
C     |    y      : y-coord where displacement is measured [m]
C     |    a      : Half size of the element - rolling direction [m]
C     |    b      : Half size of the element - lateral direction [m]
C     ========================================================================
C     OUTPUT:
C     |    F      : Flexibility coefficient
C     ========================================================================
        double precision X, Y, A, B, F

       F = (X + A) * LOG(((Y + B) + SQRT((Y + B)**2 + (X + A)**2)) /
     & ((Y - B) + SQRT((Y - B)**2 + (X + A)**2))) +
     &   (Y + B) * LOG(((X + A) + SQRT((Y + B)**2 + (X + A)**2)) /
     & ((X - A) + SQRT((Y + B)**2 + (X - A)**2))) +
     &   (X - A) * LOG(((Y - B) + SQRT((Y - B)**2 + (X - A)**2)) /
     & ((Y + B) + SQRT((Y + B)**2 + (X - A)**2))) +
     &   (Y - B) * LOG(((X - A) + SQRT((Y - B)**2 + (X - A)**2)) /
     & ((X + A) + SQRT((Y - B)**2 + (X + A)**2)))
       END

       SUBROUTINE RC_CALC(N1, N2, NE, RC)
C     ========================================================================
C     ROW - COLUMN MATRIX
C     ========================================================================
C     INPUTS:
C     |    N1      : Number of elements in the rolling direction
C     |    N2      : Number of elements in the lateral direction
C     |    NE      : Dimension of the problem
C     |    RC      : Matrix that stores the position of PCA elements [NEx2]
C     ========================================================================
C     WHAT IS BEING CALCULATED?:
C     |    RC      : Matrix that stores the position of PCA elements [NEx2]
C     ========================================================================
        INTEGER N1, N2, NE, I, J, V
        DOUBLE PRECISION RC(NE, 2)

        V = 1
        DO I = 1, N2
          DO J = 1, N1
            IF (V .LE. NE) THEN
```

```
            RC(V, 1) = J
            RC(V, 2) = I
            V = V + 1
          END IF
        END DO
      END DO
    END

    SUBROUTINE A_FORMAT(NE, N1, N2, A, AES, RC)
C   ========================================================================
C   A CREATION
C   ========================================================================
C   INPUTS:
C   |   NE      : Dimension of the problem
C   |   N2      : Number of elements in the lateral direction
C   |   A       : Matrix containing the influence between all elements of the PCA [NExNE]
C   |   Aes     : Influence coefficient`s matrix for each element [N1xN2]
C   |   RC      : Matrix that stores the position of PCA elements [NEx2]
C   ========================================================================
C   WHAT IS BEING CALCULATED?:
C   |   A       : Matrix containing the influence between all elements of the PCA [NExNE]
C   ========================================================================
      INTEGER NE, N2, I, J, Ro, Co
      DOUBLE PRECISION A(NE, NE), AES(N1, N2), RC(NE, 2)

      DO I = 1, NE
        DO J = 1, NE
          Ro = ABS(RC(J, 1) - RC(I, 1))+1
          Co = ABS(RC(J, 2) - RC(I, 2))+1
          A(I, J) = AES(Ro, Co)
        END DO
      END DO
    END

    SUBROUTINE ZEROS_VEC(SIZE, v)
C   ========================================================================
C   INITIALIZATION OF VECTORS
C   ========================================================================
C   INPUTS:
C   |   size    : Dimension of the problem
C   |   v       : Vector to be initialized
```

```
C     ======================================================================
C     WHAT IS BEING CALCULATED?:
C     |    v       : Vector initialized
C     ======================================================================
        INTEGER SIZE, I
        DOUBLE PRECISION v(SIZE)

        DO I = 1, SIZE
          v(I) = 0.0
        END DO
      END

      SUBROUTINE ZEROS_MAT(SIZE1,SIZE2, MAT)
C     ======================================================================
C     INITIALIZATION OF MATRICES
C     ======================================================================
C     INPUTS:
C     |    size1   : Rows of the matrix
C     |    size2   : Columns of the matrix
C     |    MAT     : Matrix to be initialized
C     ======================================================================
C     WHAT IS BEING CALCULATED?:
C     |    MAT     : Matrix initialized
C     ======================================================================
        INTEGER SIZE1, SIZE2, J, I
        DOUBLE PRECISION MAT(SIZE1,SIZE2)

        DO I = 1, SIZE1
          DO J=1, SIZE2
            MAT(I,J) = 0.0
          END DO
        END DO
      END

      SUBROUTINE JEC_VALUE(TOL, RUN, JEC, NE, T, PD)
C     ======================================================================
C     JEC`S VALUE CALCULATION
C     ======================================================================
C     INPUTS:
C      |     TOL      : Tolerance to differentiate elements in contact against those not in
contact
```

```
C     |    RUN     : Logical value (0: The real contact area has been found. 1: The contact
area must be re-calulated)
C     |    JEC     : Number of elements in contact in the real contact area
C     |    NE      : Total dimension of the problem
C     |    T       : Matrix that stores (with unitary value) the positions to be used in
matrices and vectors to calculate the new values after updating JEC
C     |    PD      : Normal traction vector NEx1 [N/m^2]
C     |    EC      : Vector that stores the position in Influence coefficients` matrix A to
be considered based on PD`s value.
C     ========================================================================
C     WHAT IS BEING CALCULATED?:
C     |    RUN     : Logical value (0: The real contact area has been found. 1: The contact
area must be re-calulated)
C     |    JEC     : Number of elements in contact in the real contact area
C     |    T       : Matrix that stores (with unitary value) the positions to be used in
matrices and vectors to calculate the new values after updating JEC
C     |    EC      : Vector that stores the position in Influence coefficients` matrix A to
be considered based on PD`s value.
C     ========================================================================
        INTEGER RUN, JEC, NE, N
        DOUBLE PRECISION TOL, T(NE,NE), PD(NE)

        JEC=0
        DO N = 1, NE
          IF (PD(N) .LT. TOL) THEN
              IF ((PD(N) .LT. (-TOL)) .AND.
     & (RUN .EQ. 0)) THEN
                RUN = 1
              END IF
          ELSE
                  JEC=JEC + 1
                  T(N,JEC)= 1.0
          END IF
        END DO
      END

      SUBROUTINE MULTIPLY_MAT_VEC(R, C, RE, V, A)
C     ========================================================================
C     MULTIPLY A VECTOR AND A MATRIX (RE=A*V)
C     ========================================================================
C     INPUTS:
C     |    c       : Columns
C     |    r       : Rows
```

```
C     |    Re      : Result of the multiplication
C     |    v       : vector to multiply
C     |    A       : matrix to multiply
C     ===========================================================================
C     WHAT IS BEING CALCULATED?:
C     |    Re      : Result of the multiplication
C     ===========================================================================
        INTEGER C, R, I, J
        DOUBLE PRECISION RE(R), V(C), A(R, C)

        DO I = 1, R
          DO J = 1, C
            RE(I) = RE(I) + A(I, J) * V(J)
          END DO
        END DO
      END

      SUBROUTINE MULTIPLY_MAT_VEC2(R, C, RE, V, A)
C     ===========================================================================
C     MULTIPLY A VECTOR AND A MATRIX (RE=AT*V)
C     ===========================================================================
C     INPUTS:
C     |    c       : Columns
C     |    r       : Rows
C     |    Re      : Result of the multiplication
C     |    v       : vector to multiply
C     |    A       : matrix to multiply
C     ===========================================================================
C     WHAT IS BEING CALCULATED?:
C     |    Re      : Result of the multiplication
C     ===========================================================================
        INTEGER C, R, I, J
        DOUBLE PRECISION RE(C), V(R), A(R, C)

        DO I = 1, R
          DO J = 1, C
            RE(J) = RE(J) + A(I, J) * V(I)
          END DO
       END DO
      END
```

```
      SUBROUTINE FORCE_CALC(N1, N2, PD, P, X1, X2, X1P,
     & X2P, NF, A, B)
C     ================================================================================
C     CALCULATION OF THE TOTAL FORCE
C     ================================================================================
C     INPUTS:
C     |    N1,N2   : Number of elements in the rolling and lateral directions
C     |    PD      : Normal traction vector NEx1 [N/m^2]
C     |    P       : Normal traction N1xN2 matrix [N/m^2]
C     |    X1      : Longitudinal coordinates of the elements in the PCA [m]
C     |    X2      : Lateral coordinates of the elements in the PCA [m]
C     |    X1P,X2P : Coordinates of the centre of pressure [m]
C     |    NF      : Total normal force [N]
C     |    A       : Half size of the element - rolling direction [m]
C     |    B       : Half size of the element - lateral direction [m]
C     ================================================================================
C     WHAT IS BEING CALCULATED?:
C     |    P     : Normal traction N1xN2 matrix [N/m^2]
C     |    NF      : Total normal force [N]
C     |    X1P,X2P : Coordinates of the centre of pressure [m]
C     ================================================================================
        INTEGER N1, N2, I, J, M, N
        DOUBLE PRECISION PD(NE), P(N1, N2)
        DOUBLE PRECISION X1(N1), X2(N2),SIGMA
        DOUBLE PRECISION X1P, X2P, SUMA, NF, A, B

        SIGMA=0.0
        DO N = 1, N2
          DO M = 1, N1
            P(M, N) = PD((N - 1) * N1 + M)
          END DO
        END DO
C Addition of all elements in pa to calculate the total traction
        DO N=1,N2
          DO M=1,N1
            SIGMA=SIGMA+P(M,N)
          END DO
        END DO
C Calculate X1P
        SUMA = 0.0
        DO I = 1, N1
```

```
          DO J = 1, N2
              SUMA = SUMA + P(I, J)
          END DO
            X1P = X1P + SUMA * X1(I)
        END DO
        X1P = X1P / SIGMA
C Calculate X2P
        SUMA = 0.0
        DO I = 1, N2
          DO J = 1, N1
              SUMA = SUMA + P(J, I)
          END DO
              X2P = X2P + SUMA * X2(I)
        END DO
        X2P = X2P / SIGMA
C Calculate NF
      NF = SIGMA * 4 * A * B
      END

      SUBROUTINE BCCG(K_INN,A,B,X,G,R,V,Q,
     & X_PREV,R_PREV,TOL,MAXIT,UPD,IX_BND,
     & IX_REL,MSK_BND, NE)
C     ======================================================================
C     BOUND-CONSTRAINED CONJUGATE GRADIENT ALGORITHM
C     ======================================================================
C     INPUTS:
C     |    K_INN    : Maximum number of inner iterations
C     |    A        : Coefficient matrix A [NE x NE]
C     |    B        : Right-hand side vector B [NE]
C     |    X        : Solution vector X [NE]
C     |    G        : Gradient vector G [NE]
C     |    R        : Residual vector R [NE]
C     |    V        : Search direction vector V [NE]
C     |    Q        : Temporary vector for A * V product [NE]
C     |    X_PREV   : Solution vector X from previous iteration [NE]
C     |    R_PREV   : Residual vector R from previous iteration [NE]
C     |    TOL      : Tolerance for stopping criterion
C     |    MAXIT    : Maximum number of iterations
C     |    UPD      : Array to track the relative updates [NE]
C     |    IX_BND   : Indices of bound-constrained variables [NE]
C     |    IX_REL   : Indices of released variables [NE]
```

```
C     |    MSK_BND  : Mask array indicating bound-constrained variables [NE]
C     |    NE       : Number of elements
C     OUTPUTS:
C     |    X        : Updated solution vector [NE]
C     |    G        : Updated gradient vector [NE]
C     |    UPD      : Array tracking relative updates in X [NE]
C     |    IX_BND   : Updated indices of bound-constrained variables [NE]
C     |    MSK_BND  : Updated mask for bound-constrained variables [NE]
C     FUNCTIONALITY:
C     |    Implements the bound-constrained conjugate gradient (BCCG)
C     |    algorithm to solve the system A * X = B with partial constraints
C     |    on the solution X. The algorithm iterates until convergence or
C     |    the maximum number of iterations is reached.
C     ======================================================================
      INTEGER K_INN, MAXIT, NE, NFREE, I, IT, IT_INN
      DOUBLE PRECISION TOL, SMALL, GARBAGE, ALPHA
      DOUBLE PRECISION BETA_PR, DLTA,BETA,TEMP_SQRT
      DOUBLE PRECISION DLTA_PREV, RMS_XK, RMS_UPD
      DOUBLE PRECISION X(NE), B(NE), G(NE), R(NE), V(NE), Q(NE)
      DOUBLE PRECISION UPD(NE), X_PREV(NE), R_PREV(NE), A(NE,NE)
      DOUBLE PRECISION MSK_BND(NE),TEMP
      DOUBLE PRECISION IX_BND(NE), IX_PRJ(NE), IX_REL(NE)
      INTEGER NBOUND,NXPRJ
      LOGICAL OUTER_IT, CHANGED, STOP,USE_NORMCG,USE_PLAIN

C     0. initialize:
      DO I = 1, NE
        X(I) = MAX(0.0, X(I))
        MSK_BND(I) = 0.0D0
      END DO
      CALL MATVEC(A, X, G, NE)
      DO I = 1, NE
        G(I) = G(I) - B(I)
        IF (X(I) .LE. 0.0D0 .AND. G(I) .GE. 0.0D0) THEN
          MSK_BND(I) = 1.0D0
        END IF
      END DO
      CALL FIND(MSK_BND, IX_BND, NE,NBOUND)
      NFREE = NE - NBOUND
      IT = 0
      IT_INN = 0
```

```
      STOP = .false.
      USE_NORMCG=.FALSE.
      CHANGED = .false.
      SMALL = 1.0E-14
      GARBAGE = 12345.67
      USE_PLAIN=.false.
      DO WHILE (.NOT. STOP)
C     1. given the bound set msk_bnd, iterate x and its gradient g
        IT = IT + 1
        IT_INN = IT_INN + 1
        CALL COPY(X, X_PREV, NE)
        IF (IT .GT. 1) THEN
          CALL COPY(R, R_PREV, NE)
          DLTA_PREV = DLTA
        END IF
C     3. construct a search direction that is zero in bound variables
        CALL COPY(-G, R, NE)
        CALL APPLYMASK(R, IX_BND, 0.0D0,NE, NBOUND)
        CALL DOT_PRODUCT(R, R, NE, DLTA)
      IF (USE_PLAIN .OR. USE_NORMCG) THEN
        IF ((IT .LE. 1) .OR. CHANGED) THEN
          CALL COPY(R, V, NE)
          BETA = 0.0D0
        ELSE
          BETA = RHO / RHO_PREV
          CALL ADDVEC(R, BETA, V, V, NE)
        END IF
      ELSE
        IF (IT .LE. 1) THEN
            CALL COPY(R, V, NE)
            BETA_PR = 0.0D0
          ELSE
            CALL DOT_PRODUCT(R, R_PREV, NE, TEMP)
            BETA_PR = MAX(0.0D0,((DLTA - TEMP) / DLTA_PREV))
            CALL ADDVEC(R, BETA_PR, V, V, NE)
        END IF
      END IF
        CALL APPLYMASK(V, IX_BND, 0.0D0, NE,NBOUND)
C     4. compute tildex, optimization ignoring the bounds
        CALL MATVEC(A, V, Q, NE)
        IF (IT_INN .LT. K_INN) THEN
```

```
          CALL APPLYMASK(Q, IX_BND, GARBAGE, NE, NBOUND)
        END IF
        CALL DOT_PRODUCT(R, V, NE, TEMP)
        CALL DOT_PRODUCT(V, Q, NE, ALPHA)
        ALPHA = TEMP / ALPHA
        CALL ADDVEC(X, ALPHA, V, X, NE)
C     7. check convergence of inner iteration
        TEMP_SQRT = DSQRT(DBLE(NFREE))
        CALL NORM(X, NE, RMS_XK)
        RMS_XK = RMS_XK / TEMP_SQRT
        CALL NORM_DIFF(X, X_PREV, NE, RMS_UPD)
        RMS_UPD = RMS_UPD / TEMP_SQRT
        UPD(IT) = RMS_UPD / RMS_XK
C     5. / 2. project x onto feasible domain, bind variables
        CHANGED = .false.
        OUTER_IT = ((IT_INN .GE. K_INN) .OR. (UPD(IT) .LT. TOL))
        IF (OUTER_IT) THEN
            CALL FIND_NEGATIVE(X, -SMALL, IX_PRJ, NE,NXPRJ)
          IF (NXPRJ .GT. 0) THEN
            CALL APPLYMASK(X, IX_PRJ, 0.0D0, NE,NXPRJ)
            CALL APPLYMASK_LOGICAL(MSK_BND, IX_PRJ, 1.0D0, NE,NXPRJ)
            CHANGED = .true.
          END IF
        END IF
C     6. update or recompute the gradient
        IF (CHANGED .OR. (OUTER_IT .AND. (K_INN .GT. 1))) THEN
          CALL MATVEC(A, X, G, NE)
          CALL SUBTRACT(G, B, NE)
        ELSE
          CALL ADDVEC(G, ALPHA, Q, G, NE)
        END IF
C     2. if desired: release constraints with negative gradient
        IF (OUTER_IT .AND. ((.NOT. USE_NORMCG).OR.
     & (.NOT. CHANGED))) THEN
          CALL FIND_NEGATIVE(G, -SMALL, IX_REL, NE,NXPRJ)
          IF (NXPRJ .GT. 0) THEN
          CALL APPLYMASK_LOGICAL(MSK_BND, IX_REL, 0.0D0, NE,NXPRJ)
          CHANGED = .true.
          END IF
        END IF
        CALL FIND(MSK_BND, IX_BND, NE,NBOUND)
```

```
        NFREE = NE - NBOUND
C     7. check convergence
        IF (OUTER_IT) THEN
          IT_INN = 0
        END IF
        IF (IT .GE. MAXIT .OR. (OUTER_IT .AND.
     & (.NOT. CHANGED) .AND. UPD(IT) .LT. TOL)) THEN
          STOP = .true.
        END IF
      END DO
      END

C     Subroutine to perform matrix-vector multiplication: G = A * X
      SUBROUTINE MATVEC(A, X, G, N)
      INTEGER N, I, J
      DOUBLE PRECISION A(N,N), X(N), G(N)

      DO I = 1, N
        G(I)=0.0D0
        DO J = 1, N
          G(I) = G(I) + A(I,J) * X(J)
        END DO
      END DO
      END

C     Subroutine to copy one vector to another: Y = X
      SUBROUTINE COPY(X, Y, N)
      INTEGER N, I
      DOUBLE PRECISION X(N), Y(N)
      DO I = 1, N
        Y(I) = X(I)
      END DO
      END

C     Subroutine to add two vectors: Z = X + BETA * Y
      SUBROUTINE ADDVEC(X, BETA, Y, Z, N)
      INTEGER N, I
      DOUBLE PRECISION X(N), Y(N), Z(N), BETA
      DO I = 1, N
        Z(I) = X(I) + BETA * Y(I)
      END DO
```

```
      END

C     Subroutine to apply a mask to a vector: X(IX_BND) = VAL
      SUBROUTINE APPLYMASK(X, IX, VAL, N,NB)
      INTEGER N, I
      DOUBLE PRECISION IX(NB)
      DOUBLE PRECISION X(N), VAL
         DO I = 1, NB
            X(INT(IX(I))) = VAL
         END DO
      END

C     Subroutine to apply a logical mask to a vector: MSK_BND(IX) = VAL
      SUBROUTINE APPLYMASK_LOGICAL(MSK_BND, IX, VAL, N,NL)
      INTEGER N, I
      DOUBLE PRECISION IX(NL)
      DOUBLE PRECISION MSK_BND(N), VAL
      DO I = 1, NL
          MSK_BND(INT(IX(I))) = VAL
      END DO
      END

C     Subroutine to find indices where a logical condition is true
      SUBROUTINE FIND(MSK_BND, IX_BND, N,COUNT)
      INTEGER N, I, COUNT
      DOUBLE PRECISION MSK_BND(N), IX_BND(N)
      COUNT = 0
      DO I = 1, N
        IF (MSK_BND(I) .NE. 0.0D0) THEN
          COUNT = COUNT + 1
          IX_BND(COUNT) = I
        END IF
      END DO
      END

C     Subroutine to find indices where a vector is less than a value
      SUBROUTINE FIND_NEGATIVE(X, VAL, IX_PRJ, N,COUNT)
      INTEGER N, I, COUNT
      DOUBLE PRECISION X(N), VAL, IX_PRJ(N)
      COUNT = 0
      DO I = 1, N
```

```
        IF (X(I) .LT. VAL) THEN
          COUNT = COUNT + 1
          IX_PRJ(COUNT) = I
        END IF
      END DO
      END

C     Subroutine to calculate the norm of a vector
      SUBROUTINE NORM(X, N, RESULT)
      INTEGER N, I
      DOUBLE PRECISION X(N), SUM, RESULT
      SUM = 0.0D0
      DO I = 1, N
        SUM = SUM + X(I)**2
      END DO
      RESULT = DSQRT(SUM)
      END

C     Subroutine to calculate the difference norm between two vectors
      SUBROUTINE NORM_DIFF(X, Y, N, RESULT)
      INTEGER N, I
      DOUBLE PRECISION X(N), Y(N), SUM, RESULT
      SUM = 0.0D0
      DO I = 1, N
        SUM = SUM + (X(I) - Y(I))**2
      END DO
      RESULT = DSQRT(SUM)
      RETURN
      END

C     Subroutine to subtract two vectors: X = X - Y
      SUBROUTINE SUBTRACT(X, Y, N)
      INTEGER N, I
      DOUBLE PRECISION X(N), Y(N)
      DO I = 1, N
        X(I) = X(I) - Y(I)
      END DO
      RETURN
      END

      SUBROUTINE DOT_PRODUCT(X, Y, SIZE, PROD)
```

```
      INTEGER SIZE, I
      DOUBLE PRECISION X(SIZE), Y(SIZE), PROD
      PROD = 0.0
      DO I = 1, SIZE
        PROD = PROD + X(I) * Y(I)
      END DO
     END

      SUBROUTINE COMPARE_MATRICES(MATRIX, REF_MATRIX, N1, N2)
C     ======================================================================
C     COMPARE TWO MATRICES AND CALCULATE ERRORS
C     ======================================================================
C     INPUTS:
C     |    MATRIX        : Matrix to be compared [n1 x n2]
C     |    REF_MATRIX    : Reference matrix to compare against [n1 x n2]
C     |    N1, N2        : Dimensions of the matrices
C     OUTPUTS:
C     |    MAX_ABS_ERROR : Maximum absolute error between the two matrices
C     |    MAX_REL_ERROR : Maximum relative error between the two matrices
C     FUNCTIONALITY:
C     |    Reads the reference matrix from a file (REF.TXT), compares it
C     |    element by element with the provided matrix, and computes the
C     |    maximum absolute and relative errors.
C     ======================================================================
      integer n1, n2, i, j
      double precision matrix(n1, n2), ref_matrix(n1, n2)
      double precision max_abs_error, max_rel_error, abs_error, rel_error

      max_abs_error = 0.0d0
      max_rel_error = 0.0d0
      ! Read the reference matrix from the file REF.TXT
      open(unit=10, file='ref.txt', status='old')
      do i = 1, n1
         read(10,*) (ref_matrix(i, j), j = 1, n2)
      enddo
      close(10)
      ! Compare the matrices
      do i = 1, n1
         do j = 1, n2
            abs_error = abs(matrix(i,j) - ref_matrix(i,j))
            if (ref_matrix(i,j) .ne. 0.0d0) then
```

```
            rel_error = abs_error / abs(ref_matrix(i,j))
         else
            rel_error = 0.0d0
         endif
         if (abs_error > max_abs_error) max_abs_error = abs_error
         if (rel_error > max_rel_error) max_rel_error = rel_error
      enddo
   enddo
   ! Print the maximum absolute and relative errors
   print*, 'Max absolute error:', max_abs_error/1d+6
   print*, 'Max relative error:', max_rel_error
   return
   end
```