UNIVERSITAT POLITÈCNICA DE VALÈNCIA
Departamento de Ingeniería
Mecánica y de Materiales

Máster en Ingeniería Mecánica y Materiales

MASTER THESIS

Adaptation of a library, to be released as free software, for the improvement of the finite element solution using the MLS-C recovery technique

---

Presented by: D. Mohammed Ouallal
Supervised by: Dr. Juan José Ródenas García and
Dr. Enrique Nadal Soriano
Valencia, April 2014

Acknowledgements

First, I would like to express my sincere gratitude to my tutor Juanjo for his support, help several times and especially his patience during these months of work, and offer the opportunity for further research in this area, also thanks to him this work has been completed successfully.

I want to thank Enrique who, besides being busy, he was able to attend and devote all the time.

Resumen

El objetivo principal de la tesis es adaptar una librería desarrollada en Matlab que implementa la tecnica de reconstrucción de tensiones MLS-C para que en el futuro pueda ser, bajo licencia GNU LGPL, de libre disposición para usuarios.

La librería es una herramienta para evaluar los campos de tensiones recuperadas apartir de los resultados obtenidos por diferentes analisis numericos como el MEF o metodos sin malla.

Para ilustrar el uso de la libreria MLS-C, se crearà una interface con el codigo comercial Ansys para que la libreria se llame directamente desde Ansys para postprocesar el campo discontinuo de tensiones en 2D proporcionado por este codigo.

La librería presenta una alternativa de alta calidad al promediado nodal usado por Ansys para obtener campos continuos de tensiones.

Para publicar la libreria MLS-C, hemos elegido la licencia GNU GPL introducida por Richard Stallman que a por objetivo dar a los usuarios la libertad de modificar, compartir, distribuir y redistribuir.

En es tesis, la seccion 2 va a describir la licencia GNU, la seccion 3 va a presentar un resumen de la tecnica MLS-C. La seccion 4 describirà las subrutinas principales usada para la implementacion de la tecnica MLS-C en Matlab, y subrutinas creadas usando fichero macro de Ansys para llamar a la subrutina MLS. Seccion 5 presentarà ejemplos numericos ilustrativos. El material presente en esa tesis, de la seccion 2 a la seccion 5 se va a publicar bajo la licencia GNU. Finalemente, la seccion 6 presentarà la conclusion de este trabajo con un anexo con toda la subrutina implementada en Matlab en la seccion 8.

Palabras clave: GNU LPGL, Moving Least Squares, Elementos finitos, reconstrucción, alisado

Abstract

The main objective of this master thesis is to adapt a library developed in Matlab [4] which implements the MLS-C technique to make it freely available for other researchers as a tool to evaluate recovered stress fields from the results obtained with different numerical analysis techniques like the FEM or the meshless methods.

To illustrate the use of this MLS-C library, an interface with the commercial code Ansys [5] will also be created so that the library can be directly invoked from Ansys to postprocess the discontinuous 2D stress fields provided by this code. This would then represent a high quality alternative technique to the nodal averaging technique used by Ansys to obtain continuous stress fields.

We have choosen to publish the MLS-C library under the GNU General Public License [6]introduced by Richard Stallman which aims to give computer users the freedom to modify, share, and even distribute it again.

In this thesis Section 2 will describe the GNU License, Section 3 will present a resume of the Moving Least Squares Recovery Technique with constraints (MLS-C)[3]. Section 4 will describe the main subroutines used for the Matlab implementation of the MLS-C technique and the subroutines created using the Ansys macros language to invoke the MLS-C subroutine. Section 5 will present illustrative numerical examples. Most of the material presented in this thesis in Sections 2 to 5 will be published together with the software under the GNU license. Finally Section 6 will present the conclusions of this work with an anex with all Matlab code in the section 8.

Key words: GNU LPGL, Moving Least Squares, Finite elements, reconstruction, smoothing.

Resum

L'objectiu principal de la tesi és adaptar una llibreria desenrotllada en Matlab que implementa la tècnica de reconstrucció de tensions MLS-C perquè en el futur puga ser, baix llicència GNU LGPL, de lliure disposició per a usuaris. La llibreria és una ferramenta per a avaluar els camps de tensions recuperades apartir dels resultats obtinguts per diferents anàlisis numèriques com el MEF o fiquedos sense malla.

Per a il·lustrar l'ús de la llibreria MLS-C, es crearà una interfície amb el codi comercial Ansys perquè la llibreria es cride directament des d'Ansys per a postprocesar el camp discontinu de tensions en 2D proporcionat per este codi.

La llibreria presenta una alternativa d'alta qualitat a l'amitjanat nodal usat per Ansys per a obtindre camps continus de tensions.

Per a publicar la llibreria MLS-C, hem triat la llicència GNU GPL introduïda per Richard Stallman que a per objectiu donar als usuaris la llibertat de modificar, compartir, distribuir i redistribuir.

En és tesi, la secció 2 descriurà la llicència GNU, la secció 3 presentarà un resum de la tècnica MLS-C. La secció 4 describirà les subrutines principals usada per a la implementació de la tècnica MLS-C en Matlab, i subrutines creades usant fitxer macro d'Ansys per a cridar a la subrutina MLS. Secció 5 presentarà exemples numèrics il·lustratius. El material present en eixa tesi, de la secció 2 a la secció 5 es va a publicar davall la llicència GNU.

Fina-liment, la secció 6 presentarà la conclusió d'este treball amb un annex amb tota la subrutina implementada en Matlab en la secció 8.

Paraules clau: GNU LPGL, Moving Least Squares, Elements finits, reconstrucció, allisat"

Contents

1. Introduction

During the last decades, the Finite Element Method (FEM) has been the most widely used tool for the simulation of mechanical phenomena. The FEM is a powerful numerical analysis tool for the evaluation of approximate solutions of boundary value problems defined by partial differential equations.

It is applied to a variety of problems of all types, such as fluid dynamics, electromagnetism, etc. For instance, in structural mechanics, the method owes its success to the power and simplicity it provides. A good measure of this success is the fact that thousand papers on the subject have been published in journals and international symposia. The research team of the Research Centre in Mechanical Engineering (CIIM) formerly called Research Centre in Vehicles Technology (CITV) formed by staff of Department of Mechanical and Materials Engineering (DIMM) at the Universitat Politècnica de València (UPV) is currently carrying out advanced research regarding this subject.

The FEM provides an approximated solution depending on the discretization used to evaluate it. Recent works carried out by the team at CIIM are devoted to provide a better solution than the one obtained by the FEM by postprocessing the FEM solution. These postprocessing techniques are usually called recovery techniques. The post processed or recovered solution can have two different uses:

- *Solution improvement.* The recovered solution is usually of higher accuracy than the FE solution, then it can be used to substitute de FE solution.
- *Error estimation*. Recovery-based error estimation techniques make use of the recovered solution, that can be seen as an approximation to the exact solution, to evaluate an estimation of the error in energy norm. This error estimation is not only used to

quantify the quality of the numerical solution but also to guide adaptive mesh refinement processes whose objective is to reduce the discretization error of the FE analysis.

One of the most important recovery procedures is the Superconvergent Patch Recovery technique SPR, proposed by Zienkiewicz and Zhu [1].
In the SPR the stress field at each patch (group of elements connected to a vertex node) is described by a polynomial whose coefficients are evaluated by means of a least squares fitting to the stress values provided by the FEM at the super convergence points. As the information used to fit these polynomials corresponds to stress values with higher accuracy than the standard solution, the fitted polynomial will also maintain this higher accuracy. The polynomial obtained in each patch will then be particularized at the patch assembly nodes. These nodal values of recovered stress will then be interpolated to the interior of the elements using the shape functions used to interpolate displacement field. Following Zienkiewicz and Zhu's ideas many authors have published enhancements of the SPR technique. One of these enhancements is the SPR-C technique developed for the case of linear elasticity problems by Dr. Ródenas and his collaborators. The SPR-C technique [2] imposes constrain equations using the Lagrange Multipliers technique to enforce the local (patch-wise) satisfaction of the equilibrium and compatibility equations. The solution provided by the SPR-C technique is considerably better than that obtained by the original SPR technique.

The SPR-type techniques are well suited for standard finite element implementations. However, certain FEM implementaions use Multi Point Constraints (MPC) to enforce continuity in implementations of the FEM where element splitting is used for mesh refinement. This type of mesh refinement techniques results quite interesting as it significantly simplifies the mesh refinement process. However SPR-type techniques require special implementations to account for the MPCs. Other numerical methods commonly used in numerical analysis are not suited for the SPR technique. As the SPR technique is based on an element structure, it cannot be applied if the mesh is missing, like in the case of the meshless methods. To overcome these problems, Dr. Ródenas and his collaborators also developed the MLS-C recovery technique [3] .This technique is based on the use of the Moving Least Squares technique, commonly used in the reconstruction of surfaces from discrete data. In the MLS-C technique, these discrete data corresponding to the FE stress values, evaluated at the superconvergent points, are used to reconstruct (recover) a continuous stress field. As in the case of the SPR-C technique, constrain equations are also used in the MLS-C technique to account for the local satisfaction of the equilibrium equations. The MLS-C technique is therefore an interesting tool to obtain enhanced representations of the solution provided by various numerical analysis techniques.

The main objective of this master thesis is to adapt a library developed in Matlab [4] which implements the MLS-C technique to make it freely available in the future as Open Source software for other researchers as a tool to evaluate recovered stress fields from the results obtained with different numerical analysis techniques like the FEM or the meshless methods. To illustrate the use of this MLS-C library, an interface with the commercial code Ansys [5](.) will also be created so that the library can be directly invoked from Ansys to postprocess the discontinuous 2D stress fields provided by the commercial software. This will then represent a high quality alternative technique to the nodal averaging technique used by Ansys to postprocess the FEM stress field. We have choosen to publish the MLS-C library under the GNU General Public License [6] introduced by Richard Stallman which aims to give to the computer users the freedom to modify, share, and even distribute it again.

In this work Section 2 will describe the GNU License, Section 3 will present a resume of the Moving Least Squares Recovery Technique with constraints (MLS-C) [3]. Section 4 will describe the main subroutines used for the Matlab implementation of the MLS-C technique and the subroutines created using the Ansys macros language to invoke the MLS-C subroutine. Section 5 will present illustrative numerical examples. Most of the material presented in this thesis in Sections 2 to 5 will be published together with the software under the GNU license. Finally Section 6 will present the conclusions of this work.

2. GNU General Public License – GNU

2.1. What is the GNU?

The GNU General Public License or GNU GPL is the most widely used license in the software world, guarantees to users as persons, organizations and companies, the freedom to use, study, share, copy and modify the software which is being published under the named license.

Its purpose is to claim that the software covered by that license is free software and protect it of several appropriations which eventually could restrict these freedoms to users. The GPL GNU has been created originally by Richard Stallman, the founder of the Free Software Foundation (FSF) for the GNU project.

Stallman define Free Software as "any program whose users enjoy of certain liberties" but beyond a specific concept, things that characterize the free software are the four basic freedoms which grant to the user:

1. Freedom to run the program to any purpose.
2. Freedom to study how the program works and change it so the user can do whatever he wants. This freedom allows the user "to modify the program according to his needs or convenience and use these modifications in a private manner 'without claiming these modifications'".
3. Freedom to reallocate copies so other can benefit of it.
4. Freedom to allocate copies of its modified versions to third.

One of the software aspects which the computer law covers is the contractual field. Well so, into that category we find the contracts relative to the Software, one of them is the license, one of the central materials of this master thesis.

A license is said to be Free Software when this "allows the users to use it freely in their use, reproduce, modify and distribute" (Bain, Gallego, Martínez Ribas y Rius [7]), i.e. that license which reflects the four basics freedoms of free software.

We will analyze the GNU GPL "GNU general public license" which is determinative in the moment of establishment if the license is compatible or not.

During 1980, the beginning of the GNU Project, the license "was linked to each program. Copying a license, changing its titles,… created compatibilities problems" (Vidal [8]).

Facing that situation, Stallman debugged these particular licenses to create the version 1, which could benefit any developer to license his program (Vidal [8]).

In the GPL, the granted rights reflect, obviously, the four fundamental freedoms of the Free Software: in its first clause the GNU GPL allows copying and distributing copies of the original source code, without modifications, which is freedom 2 of free software. Also allows modifying the program or part of it, i.e. the first freedom is also present in the GNU GPL. Besides, the license allows the distribution of the mentioned modifications, which corresponds to the freedom number 3. Finally the GNU GPL grants the right to copy and distribute the program, with or without modifications, right corresponding to the freedom 2 and 3 of the free Software.

As a corollary of the foregoing rights, the GNU GPL imposes certain conditions or restrictions. First, and with respect to the possibility of reproduce and distribute copies without modifying the program, the above license presents a series of requirements, being the most interesting the obligation to grant to the program recipients a copy of the license GPL along with the distributed software.

In the second place, the GPL establish the requirements to accomplish in order to copy and distribute copies of the program with modifications. The most important between them is that the derived work has to be licensed as all and only under the same GNU GPL, this is strongly required.

The Copyleft is considered a central element within the movement of the Free Software, “allows the execution of the program, its copy, modifications and distribution of modified versions, always that any kind of constraint is added a posteriori” (Stallman [10])

In the third point, and to copy and distribute the program (or its derived works), the GPL license requires (beside the accomplishment the anterior requirements) to give the source code of the program.

2.2.Publishing under GNU License

Things that we know so far about "Software license" are the terms the author determines (actually the holder of the right of exploitation) which must be respected in the moment of the distribution of copies.
Speaking in roughly mode, there exist three principal characters in the software distribution:

- The author/s of the works which is being distributed (person who wrote the program).
- The copyright holder/s.
- The user/s to which the program is destined.

In the field of the free software, the author and the copyright holder are often the same person. The main difference between them, in Spain for example, is that authority of the work is irrevocable and unalienable (the author could not transfer to another person) while is the case for the copyright by means of contract.

Anyway, it is the copyright holder concerned to determine the terms of the distribution.
Let's see how this works out, we have this simple example as follows:

```
# helloworld.m - program says Hello World
```

In this simple case we don't know who is the autor and the copyright holder. When this occurs, it doesn't mean that you are not the author, neither the intellectual property law corves it. Conversely, you are the author in the moment of performing the work.

By putting "Written by…" one could solve a lot of problem in the future.

After we modify the program we obtain:

```
# helloworld.m - program says Hello World
#
# Written by Author
```

By including our name in the head of the program we do not obtain any additional right (the authority is inherent) but we do strengthen the legal form of our program: it is more difficult that other person pretends to be the author and includes his.

We cannot see any copyright notification (Copyright (C)). Same thing occurs with the authority, and that doesn't imply that exploitation right could not be ours: There is an implicit copyright notification in the anterior program which indicates that exploitation rights correspond to the author. Specify explicitly who is the master of the exploitation rights makes the work robust legally.

```
# helloworld.m - program says Hello World
#
# Written by Author
# Copyright (C) 1999 Author
```

Now it is clearly and explicit who is the author and the copyright holder. In the following step we specify what are the conditions under the program is being distributed. This usually should be written under the copyright notification, as we have seen in many sites.

What are these conditions? That depends on the intentions of the copyright holder. People whom distribute private software (non-free software) use these conditions to grant certain rights and deny others, like including the notification “all right reserved” and use licenses called EULA’s (End User License Agreement) to grant restricted rights to people who buy it. After, if we would like our program to be private, we could simply write:

```
# helloworld.m - program says Hello World
#
# Written by Author
# Copyright (C) 2005 Author
# All rights reserved.
```

If the author do not include: “All right reserved”, it doesn’t mean that this program is not private. Actually, if the distribution conditions are omitted, what law says is that all rights are reserved. By default none of the rights is granted. This is very important: people tend to think that the programs found in the web which do not specify distribution rights, or do not include copyright notification, are in public domain, this is wrong. By default, the conditions are "all right reserved", the copyright holder is the author as well, and the name of the author is not legally required to be mentioned. This was how the private software is published.

We want our programs to be free. That means the users must enjoy the four freedoms of rights which we already mentioned in a previous section, freedoms maintained by the FSF (*Free Software Foundation*):

- Right to execute the software, for any purpose.
- Right to know how the program works and adapt it for one need.
- Right to distribute copies of the programs.
- Right to perform modifications and redistribute.

Well, at first, we want the users of our programs enjoy all these rights. As we have seen already, we can use our role of copyright holder to grant these freedoms or rights to the users (conversely to the private software, where the copyright holder use to restrict it). So we can simply include:

```
# helloworld.m - program says Hello World
#
# Written by Author
# Copyright (C) 2005 Author
# Users of this software have the right to use it without
restriction.
```

Once this is written and the software distributed, users who got access to it will have the specified rights under copyright notification. However, one problem appears which it was experimented in the first years of the free software: people who write programs (they are also the copyright holders) are usually developers but not lawyers. The problem is that the conditions of exploitation which the developers wrote under the copyright notification would be legally weak. Lawyers know all of this, and in a text like the previous one, they would be able to find a lot of weak points, ambiguities and sentences with a possible double sense. In case of legal conflict, this could mean the loss of some rights.

The solution was the following: Instead of every developer write his own distribution terms in their programs trying to follow the spirit of the definition of the free software, lawyers were paid in order to redact a text specifying these rights in a legally robust form. In that way, developers could copy under their copyright notification. This was the origin of the GPL license and other free license.

It is important that we recognize the role of the free license: they are simply terms of exploitation well redacted.

The way to proceed is very simple. We just have to take the previous example and insert the GPL text under the copyright notification, as follows:

```
# helloworld.m – program says Hello World
#
# Written by Author
# Copyright (C) 2005 Author
# <INSERT THE GPL HERE>
```

It is not an obligation to pay anyone or ask permission to use a free license. The specific free licenses are distributed under conditions of exploitation which allow its use without restriction.

However, some free licenses are too long (as the GPL) and it is not very practical to repeat it in all and every project source file under the copyright notification.

Let's see an *example* using the GPL:

```
/*
 * Software.m - Software Inc. (Software name or File name)
 * Copyright (C) (year) Author (the Copyright holder/author)
 * This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
 * This program is distributed in the hope that it will be useful but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for more details.
 * You should have received a copy of the GNU General Public License along with GNU gv; see the file COPYING. If not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA or see <http://www.gnu.org/licenses/>.
 * Author:
 * Internet: email address
 * Address:
 * Phone:
 * FAX:
 */
```

As for the record of the intellectual property, we have to understand that the registration is not an obligation: the law concedes the authority and the exploitation right without the need of registration.

The question to register or not, it is up to the author. As the program will suffer a lot of changes constantly, in addition, the free software has an excellent register subtitle: Internet. If we upload a program to a web site or similar, we don't have to replicate in thousands of web sites. It is very easy to demonstrate that a person is the owner of his program, so the idea of "Register" is a good option.

Any translation from English to another language of the GPL text is not official. It would be useful to have translations of GPL to different language. There are persons who already did the translations but without any official validation.

A legal document is in certain way as a program. To translate it is like to move a program from a language and operating system to other one. Only a jurist who knows well both languages can do it, and even this way there exist risks of committing some mistakes.

Sending to people non-official translation would mean that it is allowed to do GPL translations, but this is still not valid.

If a non-official translation has to be included for example because it could be useful to understand the GPL text, then it is necessary to include a note like:

> "This translation of the GPL is informal, and not officially approved by the Free Software Foundation as valid. To be completely sure of what is permitted, refer to the original GPL (in English)."

That way, we have seen how easy is licensing a program as free software: identify yourself as the author, put your copyright notification, choose the free license you like more, GPL, LGPL or AGPL etc. And put a note under the copyright notification specifying you are using that license (the best way to do that is copying some already existing notification).

2.3. GNU Versions

At the current time, there exist three versions of the GNU General Public License, the GNU GPL Version 1 (1989), the GNU GPL Version 2 (1991) and the latest one GNU GPL Version 3 (2007), all launched by Richard Stallman.

Changes between them are mostly minor and made to adopt the current development in software engineering, distribution and execution as well to deal with legal issues like software patents.

The reason for creating so many versions has been to include rights and other aspects not included in previous versions. As simple as it appears, subjects prevented by both, version 1 and version 2, are granted by the version 3 or improved, as the compatibility subject between GPL versions and other free license like Apache License, XFree86 License, hardware restrictions with respect to hardware modifications. And among other modification clarifying how the violations of licenses are managed, and some other points regarding the author's rights.

Taking all these improvements and all these upgrades and in order to protect users' freedom, the FSF has made a better copyleft by releasing the GNU GPL Version 3.

2.4. License Election

As mentioned before, a license is a legal instrument, well redacted terms controlling the use or distribution of software.

The Free Software Foundation itself uses licenses to grant users the four freedoms mentioned in a previous section.

Taking a look at several works published under the GNU license, the publisher must select which is the most advisable one for him.

According to the Free Software Foundation (FSF), there are three different licenses:

- The GNU GPL also named the Ordinary GNU GPL, considered the most used license.
- The GNU Lesser General public License or Library General public License, LGPL especially used to publish libraries (our case), and
- The GNU Affero General Public License or AGPL especially used to publish software that will commonly be run over a network [11] In this work we are not interested in publishing under AGPL.

So the question people ask to theirselves is, what license should I use for my work? Ignoring the AGPL, the remaining licenses are the main licenses for the GNU project, GPL and LGPL. Most libraries usually are published under the LGPL leaving out the publication under GPL.

First, one should know the difference between these two licenses: publishing under GPL makes the library available only for free programs, but publishing under LGPL allows the use of the library in propriety programs. Our library will include code developed with two different programs, Ansys and MATLAB (Private programs). According to the GNU Project [12] it is advisable publishing libraries under the GNU LGPL.

In addition there is a big advantage of using the LGPL: it is compatible with the GPL. In other words, one can combine two different works under two different free licenses.

Remains to know how can we publish a work under LGPL. We have seen in a previous section how to publish a code or software under the ordinary GLP (GNU GPL), is it as simple as it is for the LGPL.

The example become as follows:

```
/*
 * Software.c - Software Inc. (Software name or File name)
 * Copyright (C) (year) Author's name (the Copyright holder/ author)
 * This program is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software
```

```
Foundation; either version 3 of the License, or any later
version.
 * This program is distributed in the hope that it will be
useful but WITHOUT ANY WARRANTY; without even the implied
warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE.  See the GNU General Public License for more
details.
 * You should have received a copy of the GNU General
Public License along with GNU gv; see the file COPYING.  If
not, write to the Free Software Foundation, Inc., 59 Temple
Place - Suite 330, Boston, MA 02111-1307, USA or see
<http://www.gnu.org/licenses/>.
 * Author: Full author name and his professional status
 * Internet:  Email address
 * Address:
 * Phone:
 * FAX:
```

As we have mentioned before, the idea of rigister the program is a good option, for that purpose we have chosen do it using the most commen and best way: Internet.

There are a lot of websites where a free software can be uploaded, we have chosen ***sourceforge.net*** to publish our library.

3. The Moving Least Squares Recovery Technique with constraints (MLS-C) for the FEM

3.1. Introduction

The FEM is only able to provide an approximated solution. Automatic mesh refinement techniques can be used to improve the solution. These techniques require the use of error estimation techniques. Recovery-type error estimators are often preferred by practitioners. One of the reasons is that, apart from providing very accurate error estimations, an enhanced solution is obtained as part of the process.

The Zienkiewicz-Zhu recovery-type error estimator tries to estimate the discretization error in energy norm associated to the finite element analysis using the following expression. To do so it is necessary to evaluate, a recovered stress field $\boldsymbol{\sigma}^*$ from the finite element solution $\boldsymbol{\sigma}^h$, where $\boldsymbol{\sigma}^*$ should be as close as possible to the exact solution $\boldsymbol{\sigma}$.

$$\left\|\mathbf{e}_{ex}\right\|^2 \approx \left\|\mathbf{e}_{es}\right\|^2 = \int_{\Omega} \left(\boldsymbol{\sigma}^* - \boldsymbol{\sigma}^h\right)\mathbf{D}^{-1}\left(\boldsymbol{\sigma}^* - \boldsymbol{\sigma}^h\right)\mathrm{d}\Omega \qquad (1)$$

Numerical integration is used to compute (1), therefore, the values of $\boldsymbol{\sigma}^*$ are required at integration points in each element.

There are different ways to obtain $\boldsymbol{\sigma}^*$. In this case we have considered a Moving Least Squares approach (MLS) to obtain it. For a more accurate evaluation of the recovered stress field $\boldsymbol{\sigma}^*$ the MLS technique has been enhanced such as that, in the evaluation of $\boldsymbol{\sigma}^*$, we will consider equations being satisfied by the exact solution, i.e. the internal and boundary equilibrium equations.

We will here describe the MLS-C technique, implemented in a MATLAB[1] library, used to evaluate the recovered stress fields for 2D linear elasticity problems solved with the Finite Element Method (FEM). This recovered stress field has two main uses: *a)* to improve the FE discontinuous stress solution, and *b)* to estimate the error in energy norm. The mathematical basis used to implement the subroutine can be found in J.J. Ródenas *et al* [1].

3.2. MLS recovery

The MLS technique is a method used for the reconstruction of continuous functions from a set of unorganized point samples via the calculation of a weighted least squares measure biased towards the region around the point at which the reconstructed value is requested.

When the MLS technique is applied to recover a continuous stress field from FE discontinuous results, the components of the recovered stress vector around a given point $\boldsymbol{\sigma}^*(\mathbf{x})$, $(i = xx, yy, xy)$ are obtained from a polynomial expansion, using the following expression:

[1] The names of the main variables used in the subroutine are shown within brackets in the text or on the left hand side of the equations, using `Courier New fonts` so that the subroutine can be easily followed.

`FieldStGP` $\equiv \boldsymbol{\sigma}^*(\mathbf{x})$

`pTerms` $\equiv \mathbf{p}(\mathbf{x})$

`Coefi` $\equiv \mathbf{a}_i(\mathbf{x})$

$$\boldsymbol{\sigma}^*(\mathbf{x}) = \mathbf{p}(\mathbf{x})\mathbf{a}_i(\mathbf{x}) \tag{2}$$

Where $\mathbf{p}(\mathbf{x})$ contains the terms of the polynomial and $\mathbf{a}_i(\mathbf{x})$ is the vector of polynomial unknown coefficients for the i^{th} component of the stress field.

For each stress components in the 2D case, assuming quadratic interpolation polynomials, we would have:

$$\mathbf{p}(\mathbf{x}) = \{1,\ x,\ y,\ x^2,\ xy,\ y^2\} \tag{3}$$

$$\mathbf{a}_i(\mathbf{x}) = \{\mathrm{a}_{0_i},\ \mathrm{a}_{1_i},\ \mathrm{a}_{2_i},\ \mathrm{a}_{3_i},\ \mathrm{a}_{4_i},\ \mathrm{a}_{5_i},\ldots\} \tag{4}$$

$$\boldsymbol{\sigma}^*(\mathbf{x}) = \begin{Bmatrix} \sigma^*_{xx}(\mathbf{x}) \\ \sigma^*_{yy}(\mathbf{x}) \\ \sigma^*_{xy}(\mathbf{x}) \end{Bmatrix} = \mathbf{P}(\mathbf{x})\mathbf{A}(\mathbf{x}) = \begin{bmatrix} \mathbf{p}(\mathbf{x}) & 0 & 0 \\ 0 & \mathbf{p}(\mathbf{x}) & 0 \\ 0 & 0 & \mathbf{p}(\mathbf{x}) \end{bmatrix} \begin{Bmatrix} \mathbf{a}_{xx}(\mathbf{x}) \\ \mathbf{a}_{yy}(\mathbf{x}) \\ \mathbf{a}_{xy}(\mathbf{x}) \end{Bmatrix} \tag{5}$$

The matrix form given by (5) will help us to impose the boundary and internal equilibrium in later calculations.

Let χ be a point within the support denominated $\Omega_{\mathbf{x}}$ corresponding to a point $\mathbf{x}$ defined by a radius $R_{\Omega_{\mathbf{x}}}$ (**RadProb**). The MLS approximation is given by

SIG $\equiv \boldsymbol{\sigma}^{*}(\mathbf{x},\chi)$	$\boldsymbol{\sigma}_i^{*}(\mathbf{x},\chi) = \mathbf{p}(\chi)\mathbf{a}_i(\chi) \quad \forall\chi \in \Omega_{\mathbf{x}}$ (6)

A continuous MLS approximation the following functional is used to obtain the unknown coefficients $\mathbf{A}$.

FieldFEGP $\equiv \boldsymbol{\sigma}^{h}(\mathbf{x})$ **W** $\equiv W(\mathbf{x}-\chi)$	$J(\mathbf{x}) = \int W(\mathbf{x}-\chi)\left[\boldsymbol{\sigma}^{*}(\mathbf{x},\chi) - \boldsymbol{\sigma}^{h}(\mathbf{x},\chi)\right]^2 \mathrm{d}\chi$ (7)

where W is the weighting function.

The coefficients will be evaluated through the linear system $\mathbf{M}(\mathbf{x})\mathbf{A}(\mathbf{x}) = \mathbf{G}(\mathbf{x})$ where

$$\mathbf{M}(\mathbf{x}) = \int_{\Omega_{\mathbf{x}}} W(\mathbf{x}-\chi)\mathbf{P}^{T}(\chi)\mathbf{P}(\chi)\,\mathrm{d}\chi \tag{8}$$

$$\mathbf{G}(\mathbf{x}) = \int_{\Omega_{x}} \mathrm{W}(\mathbf{x}-\chi)\mathbf{P}^{T}(\chi)\boldsymbol{\sigma}^{h}(\chi)\,\mathrm{d}\chi \tag{9}$$

This equation can be numerically evaluated, considering n influence points within the support $\Omega_{\mathbf{x}}$, as follows;

$$J(\mathbf{x}) = \sum_{l=1}^{n} W(\mathbf{x}-\chi_l)\left[\boldsymbol{\sigma}^*(\mathbf{x},\chi_l)-\boldsymbol{\sigma}^h(\mathbf{x},\chi_l)\right]^2 |\mathbf{J}(\chi_l)| H_l \tag{10}$$

NumInflPoints

$\equiv \boldsymbol{n}$

$$\mathbf{M}(\mathbf{x}) = \sum_{l=1}^{n} W(\mathbf{x}-\chi_l)\mathbf{P}^T(\chi_l)\mathbf{P}(\chi_l)|\mathbf{J}(\chi_l)| H_l \tag{11}$$

$$\mathbf{G}(\mathbf{x}) = \sum_{l=1}^{n} W(\mathbf{x}-\chi_l)\mathbf{P}^T(\chi_l)\boldsymbol{\sigma}^h(\chi_l)|\mathbf{J}(\chi_l)| H_l \tag{12}$$

where $|\mathbf{J}(\chi_l)|$ is the Jacobian of the transformation between the global and the local coordinates and H_l is the weight corresponding to point $\boldsymbol{\chi}_l$

The influence points are simply, the integration points used in the FE analysis falling into the inlfuence domain, where the FE stresses are known.

The weighting function in the MLS subroutine implemented is:

$$W(\mathbf{x}) = \begin{cases} 1-6s^2+8s^3+-3s^4 & \text{if } |s|\le 1 \\ 0 & \text{if } |s|>1 \end{cases} \tag{13}$$

where s is the normalized distance function expressed as;

$$s = \frac{\left\| \mathbf{x} - \chi \right\|}{R_{\Omega_{\mathbf{x}}}} \tag{14}$$

3.3. Satisfaction of the boundary equilibrium equation

Constrain equations will be imposed to enforce the satisfaction of the boundary equilibrium equation. In the MLS technique the boundary will smoothly appear in the support of **x** when **x** gets closer to the boundary. The formulation must enforce the satisfaction of the boundary condition when **x** is on the boundary. In order to avoid discontinuities when approaching the boundary, the influence of the boundary conditions must also smoothly appear in the formulation. As described in J.J. Ródenas *et al* [1] the functional in (7) will be modified to account for the satisfaction of the boundary equilibrium equation using a so-called *nearest point approach*. This approach weights the satisfaction of this equation at the points χ_j (within $\Omega_{\mathbf{x}}$) on the contours used to define the boundary closest to **x**. Figure 1 shows a case where 2 nearest points have to be considered because **x** is close to a corner on the boundary.

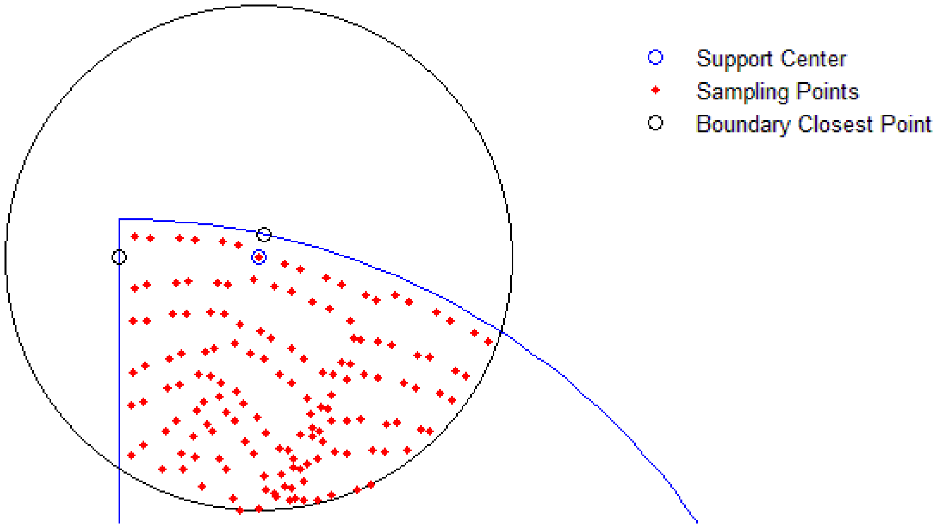


Figure 1: MLS support with boundary conditions applied on the nearest points on the boundary.

The recovered stress field $\boldsymbol{\sigma}^*(\mathbf{x},\chi)$ can be expressed as $\tilde{\boldsymbol{\sigma}}^*(\mathbf{x},\chi)$ in a local coordinate system aligned with the contour using a rotation matrix $\mathbf{r}(\alpha)$, where α is the angle between the horizontal global axis x and the vector normal to the surface at χ_j. This will help to simplify the consideration of the boundary equilibrium equation:

`PtRtSigt` $\equiv \tilde{\boldsymbol{\sigma}}^*(\mathbf{x},\chi)$

`RStress` $\equiv \mathbf{r}(\alpha)$

$$\tilde{\boldsymbol{\sigma}}^*(\mathbf{x},\chi) = \mathbf{r}(\alpha)\boldsymbol{\sigma}^*(\mathbf{x},\chi) \quad (15)$$

where the rotation matrix $\mathbf{r}(\alpha)$ is given by:

$$\mathbf{r}(\alpha)=\begin{bmatrix} \cos^2\alpha & \sin^2\alpha & \sin(2\alpha) \\ \sin^2\alpha & \cos^2\alpha & -\sin^2\alpha \\ -0.5\sin(2\alpha) & 0.5\sin(2\alpha) & \cos(2\alpha) \end{bmatrix} \quad (16)$$

The unknown coefficients $\mathbf{A}$ are evaluated solving a linear system of equations $\mathbf{M}(\mathbf{x})\mathbf{A}(\mathbf{x})=\mathbf{G}(\mathbf{x})$ where the consideration of the boundary equilibrium leads to the following expressions:

```
PtRtRP
```
$\equiv \mathbf{P}^T(\chi_j)\mathbf{r}_{\tilde{i}}^T\mathbf{r}_{\tilde{i}}\mathbf{P}(\chi_j)$

$$\mathbf{M}(\mathbf{x})=\sum_{l=1}^{n}W(\mathbf{x}-\chi_l)\mathbf{P}^T(\chi_l)\mathbf{P}(\chi_l)\left|\mathbf{J}(\chi_l)\right|H_l + \sum_{j=1}^{NBC}\tilde{W}(\mathbf{x}-\chi_j)\mathbf{P}^T(\chi_j)\mathbf{r}_{\tilde{i}}^T\mathbf{r}_{\tilde{i}}\mathbf{P}(\chi_j) \quad (17)$$

```
PtRtSig
```
$\equiv \mathbf{P}^T(\chi_j)\mathbf{r}_{\tilde{i}}^T\boldsymbol{\sigma}_{\tilde{i}}^{ex}\mathbf{P}(\chi_j)$

$$\mathbf{G}(\mathbf{x})=\sum_{l=1}^{n}W(\mathbf{x}-\chi_l)\mathbf{P}^T(\chi_l)\boldsymbol{\sigma}^h(\chi_l)\left|\mathbf{J}(\chi_l)\right|H_l + \sum_{j=1}^{NBC}\tilde{W}(\mathbf{x}-\chi_j)\mathbf{P}^T(\chi_j)\mathbf{r}_{\tilde{i}}^T\boldsymbol{\sigma}_{\tilde{i}}^{ex}\mathbf{P}(\chi_j) \quad (18)$$

`WBP` $\equiv \tilde{W}(\mathbf{x}-\chi_j)$

where the weighting function $\tilde{W}(\mathbf{x}-\chi_j)$ used to consider the boundary equilibrium equation is given by:

$$\tilde{W}(\mathbf{x}-\chi_j)=\frac{W(\mathbf{x}-\chi_j)}{s}=\begin{cases}\frac{1}{s}-6s+8s^2+-3s^3 & \text{if } |s|\leq 1\\ 0 & \text{if } |s|>1\end{cases} \tag{19}$$

In this definition of the weighting factor $\tilde{W}(\mathbf{x}-\chi_j)$, the term $\tilde{W}(\mathbf{x}-\chi_j)$ is in charge of smoothly introducing the effect of the boundary equilibrium whereas the term $1/s$ has the effect of increasing the weight of the term used to consider the boundary equilibrium when $\mathbf{x}\rightarrow\chi_j$, i.e., when $s\rightarrow 0$. Note that $s = 0$ if $\mathbf{x}$ is on the boundary, hence $1/s = \infty$ in (19), In this case, instead of using (19), the boundary equilibrium equation is strongly enforced using the Lagrange Multipliers technique.

3.4. Satisfaction of the internal equilibrium equation.

The Lagrange Multipliers technique is used to impose the internal equilibrium equation given by:

`Sig` $\equiv \boldsymbol{\sigma}^*$

$$\nabla\boldsymbol{\sigma}^*+\mathbf{b}=0 \tag{20}$$

Considering (5), $\nabla\boldsymbol{\sigma}^*$ will be given by:

$$\nabla\boldsymbol{\sigma}^*=(\nabla\mathbf{P})\mathbf{A}+\mathbf{P}(\nabla\mathbf{A}) \tag{21}$$

where $\nabla\mathbf{A}$ is unknown. Differentiating $\mathbf{M}(\mathbf{x})\mathbf{A}(\mathbf{x})=\mathbf{G}(\mathbf{x})$ we obtain:

$$(\nabla\mathbf{M})\mathbf{A}+\mathbf{M}(\nabla\mathbf{A})=\nabla\mathbf{G} \tag{22}$$

The following expressions for the partial derivatives of $\boldsymbol{\sigma}^*$ are obtained evaluating $\nabla\mathbf{A}$ from (22) and substituting in (21):

Minv $\equiv \mathbf{M}^{-1}$

$$\frac{\partial\boldsymbol{\sigma}^*}{\partial x}=\left(\frac{\partial\mathbf{P}}{\partial x}-\mathbf{P}\mathbf{M}^{-1}\frac{\partial\mathbf{M}}{\partial x}\right)\mathbf{A}+\mathbf{P}\mathbf{M}^{-1}\frac{\partial\mathbf{G}}{\partial x} \tag{23}$$

$$\frac{\partial\boldsymbol{\sigma}^*}{\partial y}=\left(\frac{\partial\mathbf{P}}{\partial y}-\mathbf{P}\mathbf{M}^{-1}\frac{\partial\mathbf{M}}{\partial y}\right)\mathbf{A}+\mathbf{P}\mathbf{M}^{-1}\frac{\partial\mathbf{G}}{\partial y} \tag{24}$$

where, for example, for the partial derivatives with respect to x we will have:

$$\begin{aligned}\frac{\partial\mathbf{M}(\mathbf{x})}{\partial\mathbf{x}}=&\sum_{l=1}^{n}\frac{\partial W(\mathbf{x}-\chi_l)}{\partial\mathbf{x}}\mathbf{P}^{\mathrm{T}}(\chi_l)\mathbf{P}(\chi_l)\left|\mathbf{J}(\chi_l)\right|H_l+\\&\sum_{j=1}^{NBC}\frac{\partial\widetilde{W}(\mathbf{x}-\chi_j)}{\partial\mathbf{x}}\mathbf{P}^{\mathrm{T}}(\chi_j)\mathbf{r}_{\tilde{i}}^{T}\mathbf{r}_{\tilde{i}}\mathbf{P}(\chi_j)\end{aligned} \tag{25}$$

and

$$\frac{\partial \mathbf{G}(\mathbf{x})}{\partial \mathbf{x}} = \sum_{l=1}^{n} \frac{\partial W(\mathbf{x}-\boldsymbol{\chi}_l)}{\partial \mathbf{x}} \mathbf{P}^{\mathrm{T}}(\boldsymbol{\chi}_l)\boldsymbol{\sigma}^h(\chi_l)\left|\mathbf{J}(\boldsymbol{\chi}_l)\right|H_l + \sum_{j=1}^{NBC} \frac{\partial \tilde{W}(\mathbf{x}-\boldsymbol{\chi}_j)}{\partial \mathbf{x}} \mathbf{P}^{\mathrm{T}}(\boldsymbol{\chi}_j)\mathbf{r}_{\tilde{i}}^{T}\boldsymbol{\sigma}_{\tilde{i}}^{ex}\mathbf{P}(\boldsymbol{\chi}_j) \tag{26}$$

with

$$\frac{\partial W(\mathbf{x}-\boldsymbol{\chi}_l)}{\partial \mathbf{x}} = \frac{\partial W(\mathbf{x}-\boldsymbol{\chi}_l)}{\partial s}\frac{\partial s}{\partial \mathbf{x}} \tag{27}$$

$$\frac{\partial \tilde{W}(\mathbf{x}-\boldsymbol{\chi}_j)}{\partial \mathbf{x}} = \frac{\tilde{W}(\mathbf{x}-\boldsymbol{\chi}_j)}{\partial s}\frac{\partial s}{\partial \mathbf{x}} \tag{28}$$

Differentiating $s = \frac{\left\|\mathbf{x}-\boldsymbol{\chi}\right\|}{R_{\Omega_{\mathbf{x}}}}$ with respect to $\mathbf{x}$ we obtain $\frac{\partial s}{\partial x}$.

All these terms allow us to evaluate (23) and (24) which will allow us to consider de constraint equation (20) using the Lagrange multipliers technique, leading to the following system of equations to be solved at each point where the recovered stresses have to be evaluated:

$$\begin{bmatrix} \mathbf{M} & \mathbf{C}^T \\ \mathbf{C} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{A} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{G} \\ \mathbf{D} \end{bmatrix} \tag{29}$$

where $\mathbf{C}$ and $\mathbf{D}$ are used to express the constraint equation to impose the satisfaction of the internal equilibrium equation and $\boldsymbol{\lambda}$ is the vector of Lagrange multipliers.

In the evaluation of (22) we considered that the unknown coefficients **A** were evaluated solving $\mathbf{M}(\mathbf{x})\mathbf{A}(\mathbf{x})=\mathbf{G}(\mathbf{x})$. Note that this represents an approximation because (29) shows that **A** is evaluated using:

$$\mathbf{M}\mathbf{A}+\mathbf{C}^T\boldsymbol{\lambda}=\mathbf{G} \tag{30}$$

Therefore, the term $\mathbf{C}^T\boldsymbol{\lambda}$ has not been included when evaluating the derivatives of **A**. This approximation implies that the internal equilibrium equation is not exactly satisfied. If the recovered stress field $\boldsymbol{\sigma}^*$ is conutinuous and satisfies the equilibrium equations, then, the use of $\boldsymbol{\sigma}^*$ in (1) will ensure that $\|\mathbf{e}_{es}\|$ is an upper bound of the exact error $\|\mathbf{e}_{ex}\|$. Hence, in this case the error upper bound property cannot be warrantied, although in most practical cases $\|\mathbf{e}_{es}\|>\|\mathbf{e}_{ex}\|$.

4. Description of the MLS-C library

In this section we will show how to implement the MATLAB MLS subroutine using 2D problems.

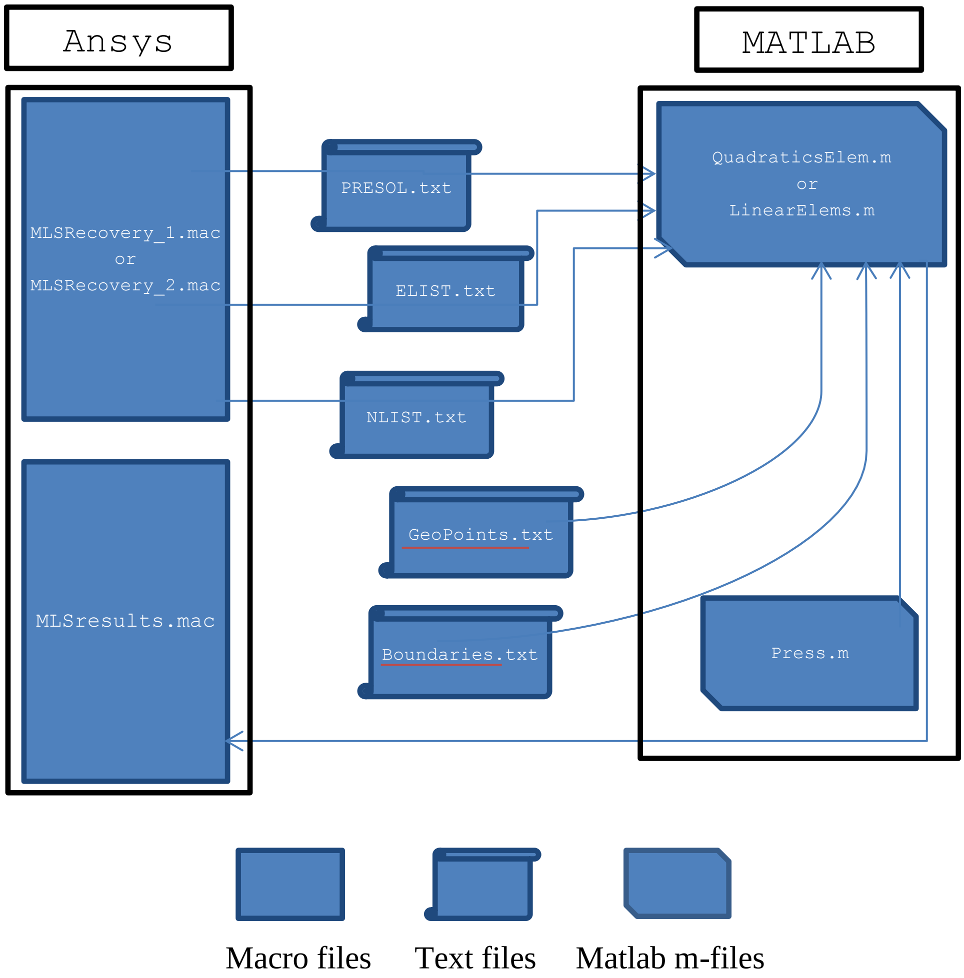


Figure 2. Data flow. Underlined file names indicate files that must be created by the user. The rest of files are already available or automatically created.

The following diagram show the data flow between the different subroutines. The user has to run two different macros within Ansys:

- `MLSRecovery_1.mac`/`MLSRecovery_2.mac`.This subroutine invokes the MLS-C Matlab subroutine that will evaluate the recovered stress field.
- `MLSresults.mac` This subroutine loads in the Ansys data base the smoothing stress field in nodes to visualize the results in Ansys. Before calling this subroutine, Ansys data base containing the original problem must be closed and a new empty data base created. This subroutine contains informations about nodes, elements and necessary results to be visualized.

A macro file called `MLSRecovery.mac` is written to obtain the necessary lists of data from Ansys containing node coordinates, stresses, and topology in text file. Delete blanks and headers from these files so later can be used as input arguments when calling the MLS-C subroutine. Some files used to describe the geometry and boundary conditions must be created specifically for each problem:

- ***Geometry:*** The geometry of the problem is defined using 2 text files: `GeoPoints.txt` and `Boundaries.txt`. The first one, `GeoPoints.txt`, is used to define the points that will later be used to define the boundary curves in the `Boundaries.txt` file. In this release of the subroutine we have considered that the geometry is defined by a combination of straight line segments and arcs of circumference.

- ***Neumann boundary conditions:*** these boundary conditions are defined in the Matlab file `press.m`

To run this subroutine, first thing we do is opening a new data base in Ansys, define the problem to be analyzed. Considering the use of linear elements, once the FE results are available, the recovered solution is obtaining running the macro that calls the Moving Least Squares recovery subroutine. To do this simply run the macro file named `MLSRecovery_1.mac` from the Ansys General Prostproc by typing `MLSRecovery_1` in the command line. Use the `MLSRecovery_2` macro if quadratic elements are used. Another macro file is automatically created called `MLSResults.mac`, to run it, we have to close Ansys and open another empty data base and write in the command line `MLSResults` and visualize the results.

We have described in the section 3 the MLS recovery process in detail, and how the recovered stress field is obtained.

The MLS function **`MLS.m`** is the main subroutine in the library. This subroutine implements the MLS-C technique described in the previous section.

The input variables used in this function are:

- `FieldFEGP`: The finite element stresses at sampling (integration) points.
- `XYZGP`: Coordinates of the sampling (integration) points.
- `XYZout`: Coordinates of the points where the recovered stress field has to be evaluated.

The MLS function returns the recovered stress field as output:

- `FieldStGP`: The recovered stress fields at `XYZout`. The results are in a text file with `.txt` extension that will be used in further operations.

The MLS function invokes other functions:

- `PlotGeo.m`: plots the geometry if required. This is useful if the user want to check that the geometry created for the Matlab subroutine is the right geometry.
- `PlotResults.m`: plots the all the results regarding the recovered stress field with legend.
- `ClosePointInBound.m`: Find the closest point to a given point x over the boundary of the cylinder.
- `polininterp2d.m`: polynomial expansion for each one of the components of the recovered stress field.

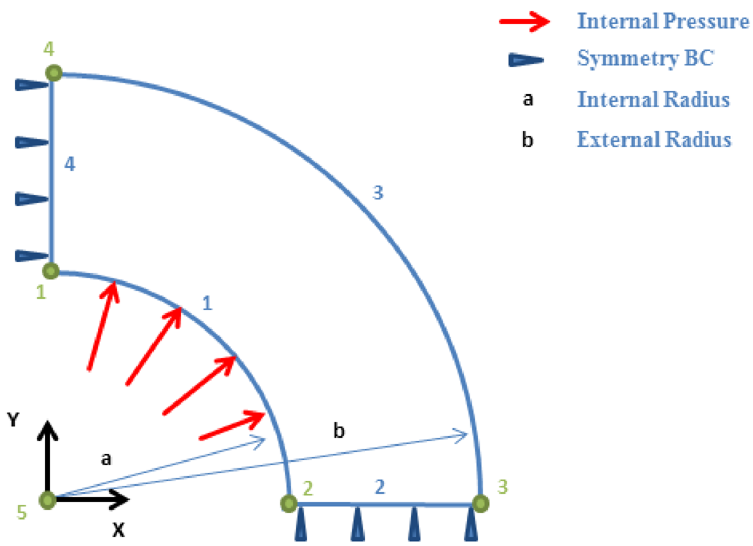
- `ContourStresses.m`: Returns Normal and Tangential stresses applied at the global coordinate system.
- `RotationMatrices.m`: The stress field is being evaluated at the contour in a coordinate system aligned with the same contour which can be rotated

For further information, all the subroutines are included in the Annex.

5. Numerical examples

This sections shows numerical examples used to check the performance of the MLS-C library. For all models a plane strain condition is assumed. The meshes were composed by linear and quadratics triangles, and linear and quadratic quadrilaterals. Meshes with both, triangles and quadrilaterals have also been considered.

Two geometries have been considered. The first one, represented in the following figure, corresponds to a cylinder under internal pressure



Plain strain $a = 5$ $b = 20$ $P = 100$ $E = 2e11$ $\nu = 0.3$

Figure 3: Thick-wall cylinder subjected to internal pressure

This problem has an analytical solution given by the following expressions. For a point of coordinates (x,y) we consider

$$r = \sqrt{x^2 + y^2}$$
$$\theta = \mathrm{arctg}\left(\frac{y}{x}\right)$$
$$k = \frac{b}{a}$$

- Radial diplacement:

$$u_r = \frac{p\,(1+\nu)}{E\,(k^2 - 1)}\left[(1-2\nu)\,r + \frac{b^2}{r}\right]$$

- Stress in cylindric coordinates

$$\sigma_r = \frac{p}{k^2-1}\left(1 - \frac{b^2}{r^2}\right); \qquad \sigma_t = \frac{p}{k^2-1}\left(1 + \frac{b^2}{r^2}\right)$$

- Stress in Cartesian coordinates

$$\sigma_x = \sigma_r \cdot \cos^2(\theta) + \sigma_t \cdot \mathrm{sen}^2(\theta)$$
$$\sigma_y = \sigma_r \cdot \mathrm{sen}^2(\theta) + \sigma_t \cdot \cos^2(\theta)$$
$$\tau_{xy} = (\sigma_r - \sigma_t) \cdot \mathrm{sen}(\theta) \cdot \cos(\theta)$$

The second example corresponds to the support loaded with a traction of 100MPa show in the following figure. Note that due to the symetry of the problem, only one half of it has been modeled.

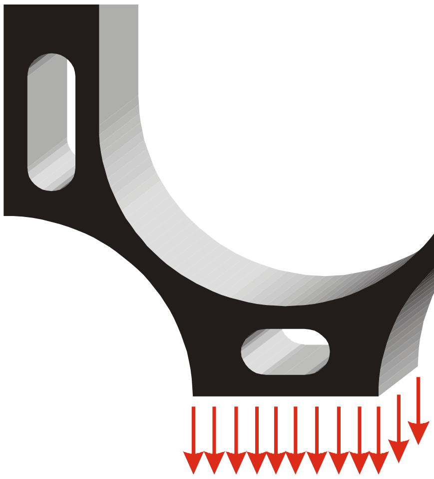


Figure 4: Support

The following meshes have been used.

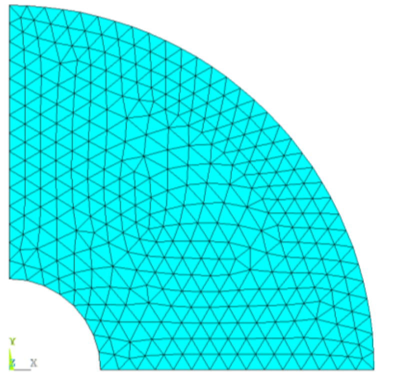
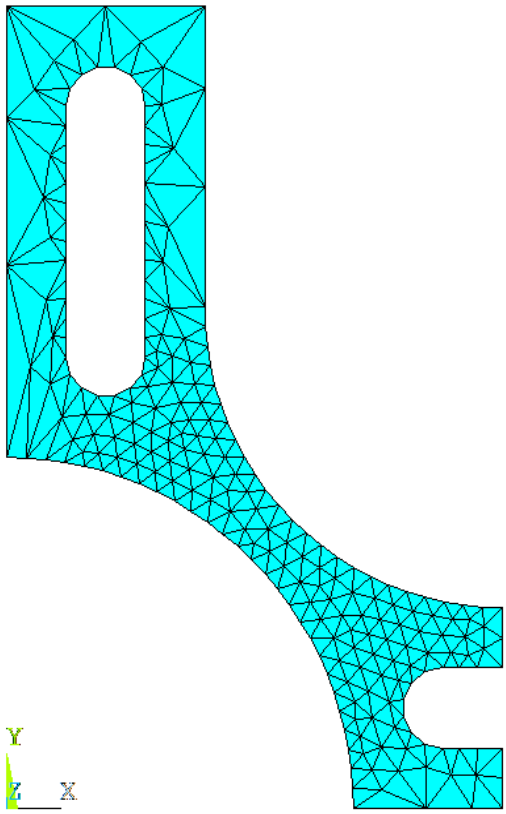
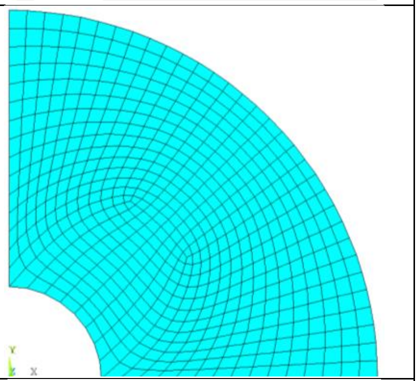
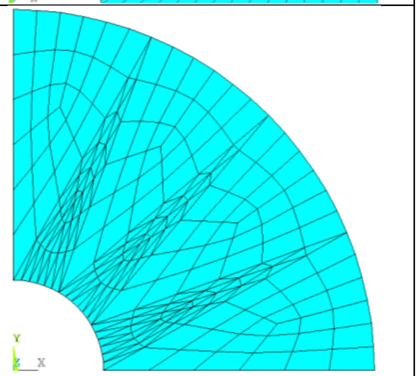
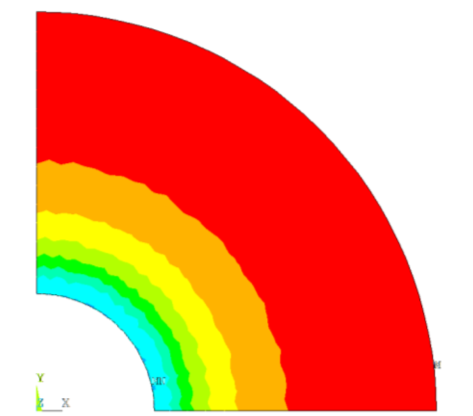
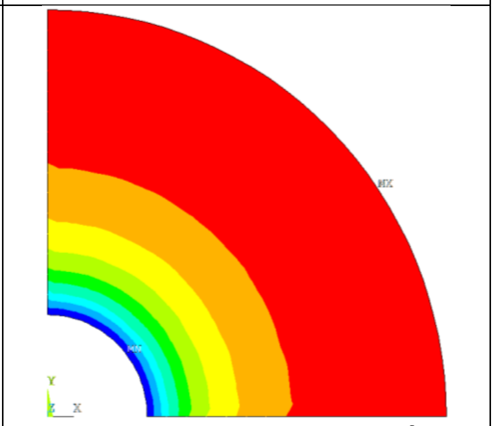
Mesh for cylinder	*Mesh for support*
	
	
	

Figure 5: Meshes used in the numerical analyses

The following sections show the results obtained by Ansys and MLS-C technique for both problems.

5.1. Cylinder under internal pressure.

Linear interpolation. Meshes with Triangular elements

Ansys results	MLS-C results
	
$\sigma_{r\ min}$ = -80.6977 $\sigma_{r\ max}$ = -0.229693	$\sigma_{r\ min}$ = -100 $\sigma_{r\ max}$ = $-0.665 \cdot 10^{-6}$

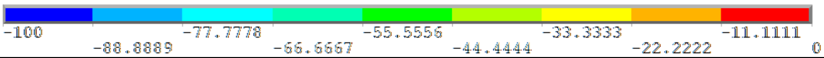
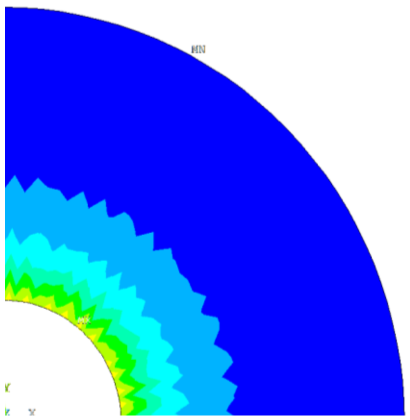
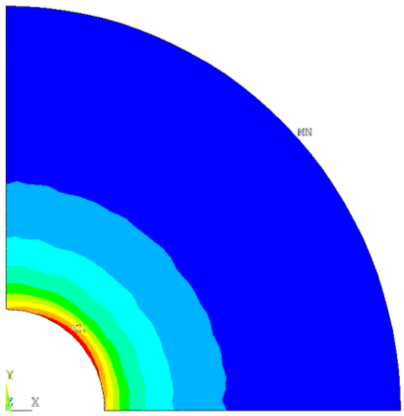


Figure 6: Radial stress σ_r ($\sigma_{r\ min\ exact}$ = -100, $\sigma_{r\ max\ exact}$ = 0)

Ansys results	MLS-C results
$\sigma_{t\ \min}$ = 13.0718 $\sigma_{t\ \max}$ = 119.113	$\sigma_{t\ \min}$ = 12.9006 $\sigma_{t\ \max}$ =154.05





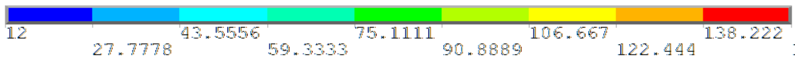
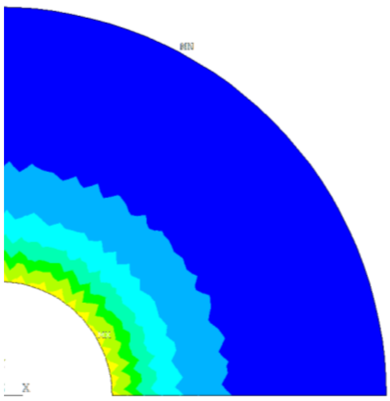
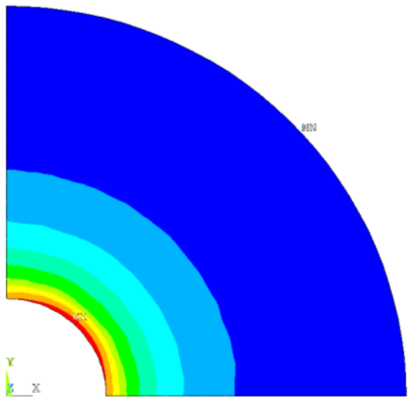


Figure 7: Hoop stress σ_t ($\sigma_{t\ min\ exact}$ =13.33, $\sigma_{t\ max\ exact}$ =113.33)

Ansys results	MLS-C results
$\sigma_{VM\ \min}$ = 13.2032 $\sigma_{VM\ \max}$ =170.334	$\sigma_{VM\ \min}$ = 12.9006 $\sigma_{VM\ \max}$ = 221.667





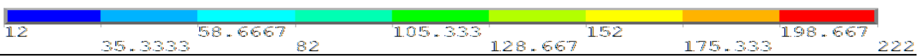


Figure 8: Von Mises stress σ_{vm}

($\sigma_{vm\ min\ exact}$ =11.8509, $\sigma_{vm\ max\ exact}$ =184.771)

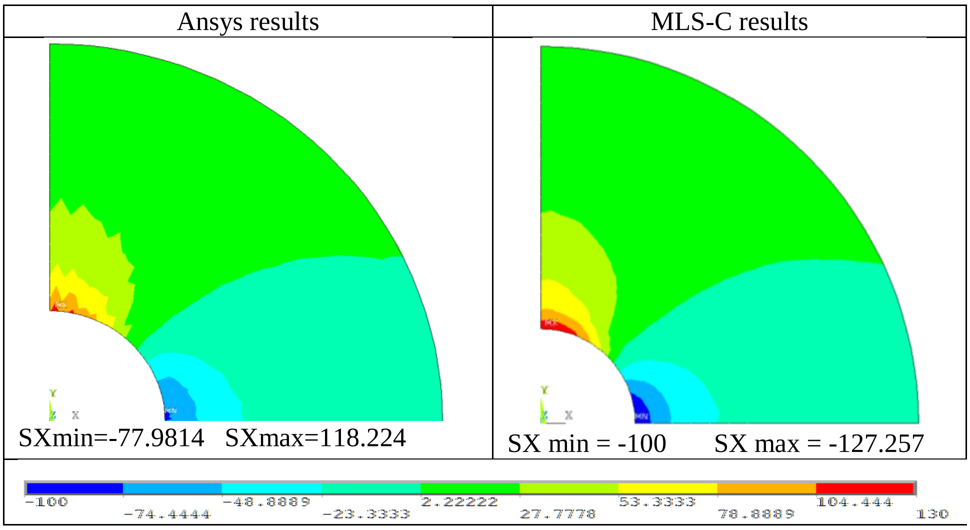


Figure 9: Cartesian stress σ_x ($\sigma_{x\ min\ exact}$=-100, $\sigma_{x\ max\ exact}$=113.33)

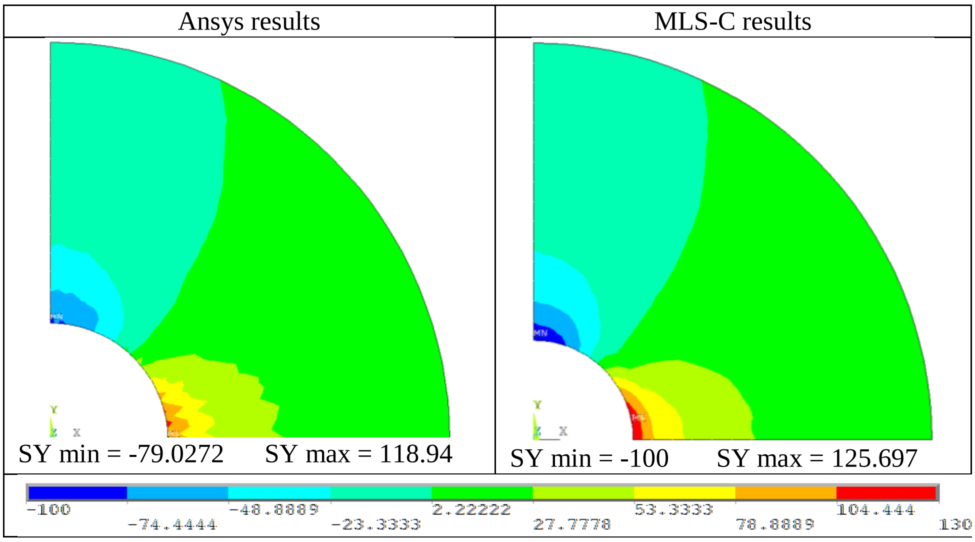


Figure 10: Cartesian stress σ_y ($\sigma_{y\ min\ exact}$=-100, $\sigma_{y\ max\ exact}$=113.33)

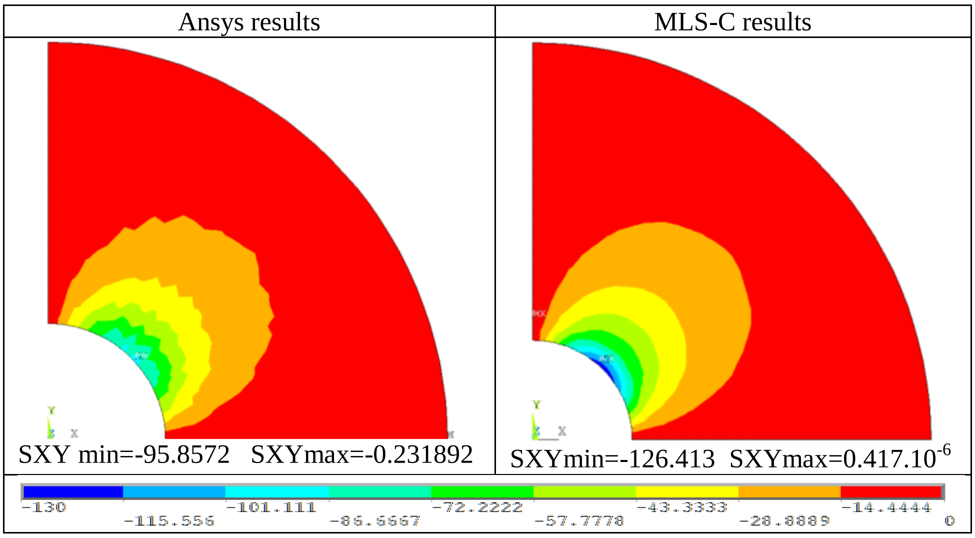


Figure 11: Cartesian shear stress τ_{xy}

Linear interpolation. Meshes with Quadrilateral elements

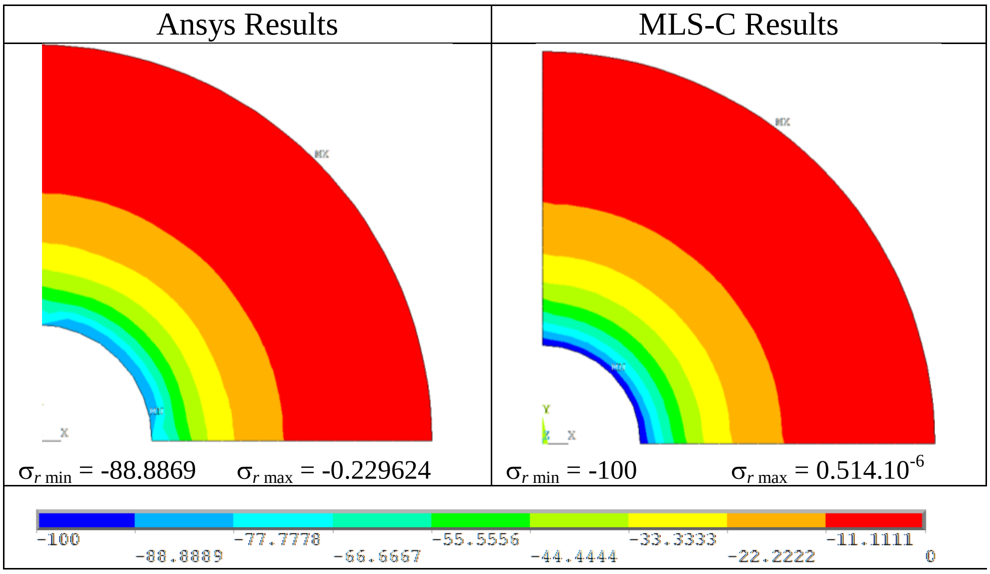


Figure 12: Radial stress σ_r

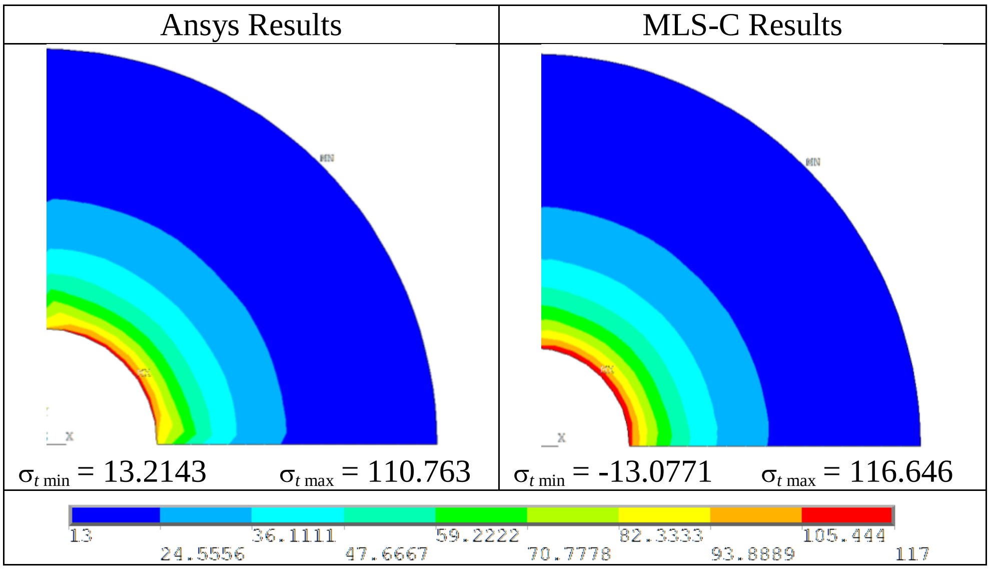


Figure 13: Hoop stress σ_t

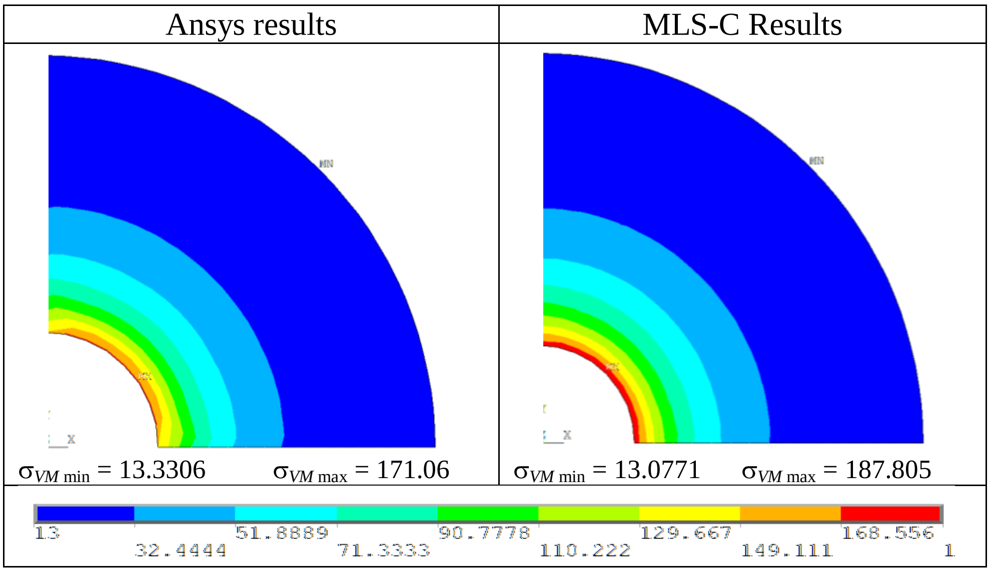


Figure 14: Von Mises stress σ_{VM}

Linear interpolation. Meshes with Quadrilateral and Triangular elements

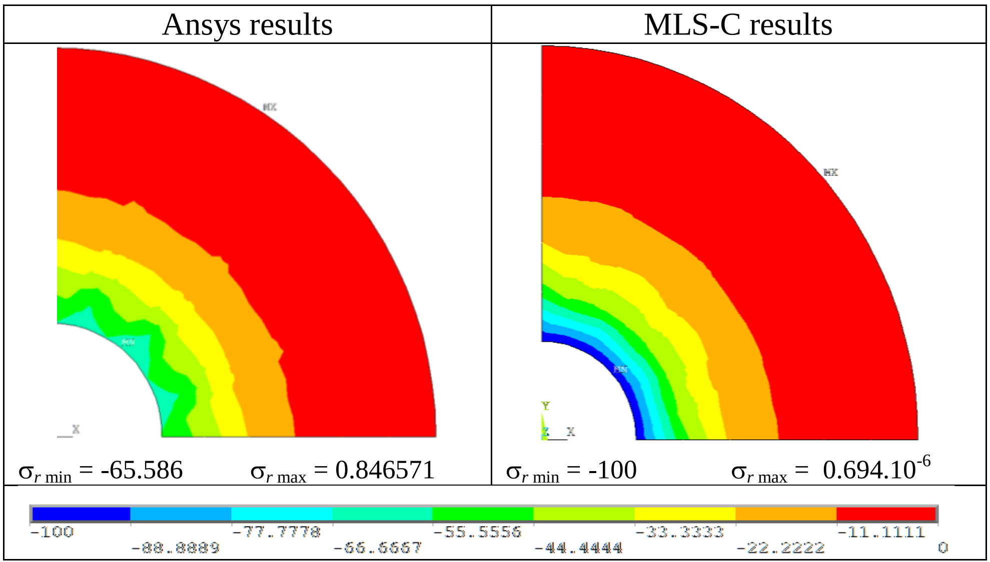


Figure 15: Radial stress σ_r

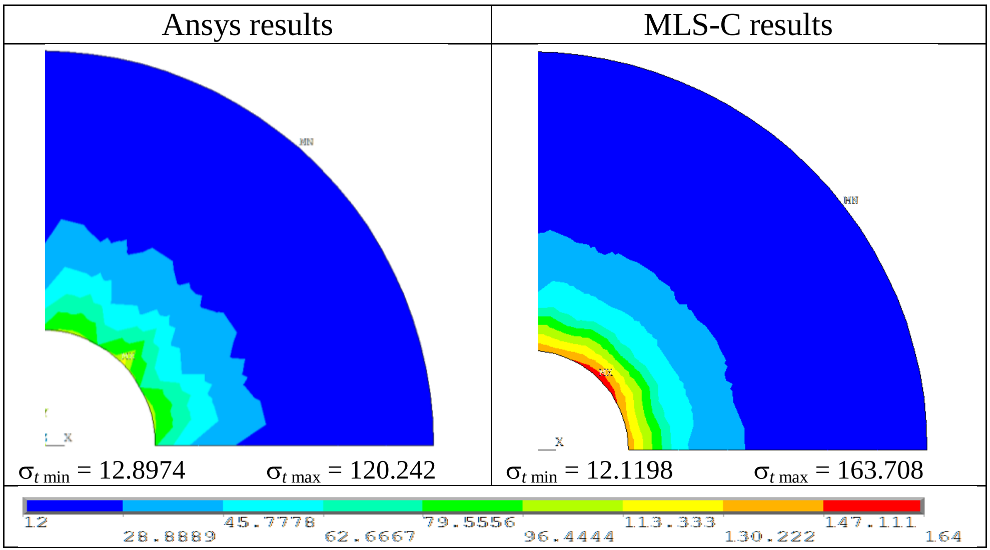


Figure 16: Hoop stress σ_t

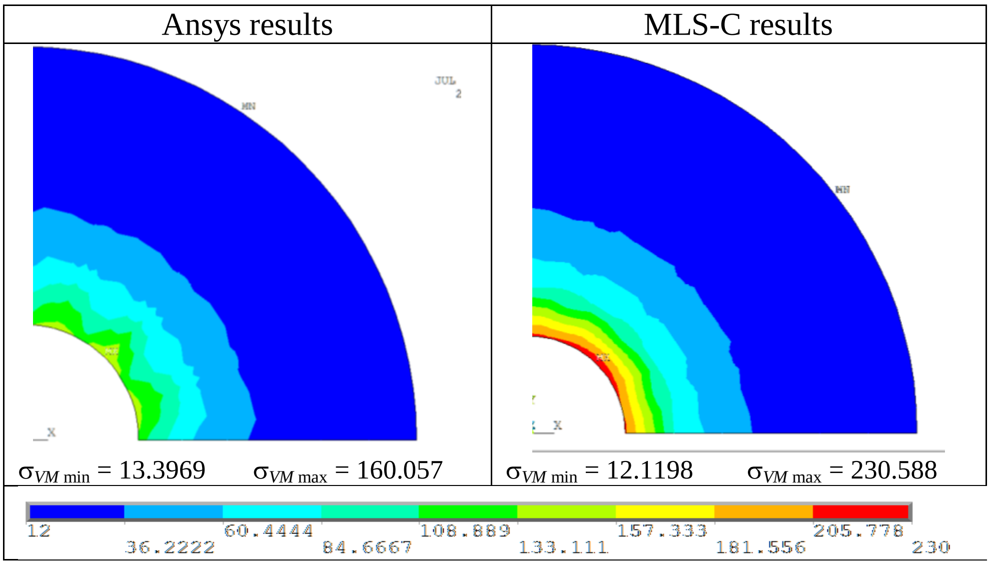
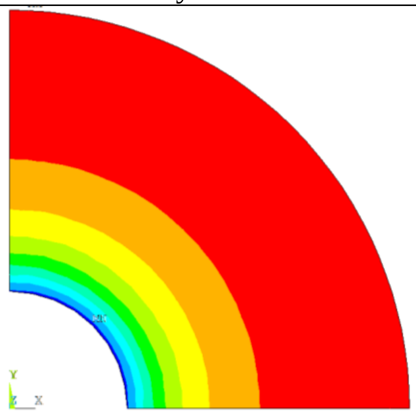
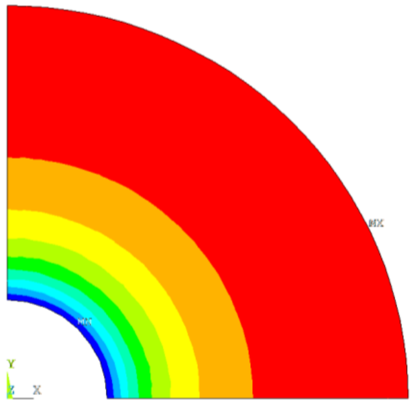


Figure 17: Von Mises stress σ_{VM}

Quadratic interpolation. Meshes with Triangular elements

Ansys results	MLS-C results
	
$\sigma_{r\,min}$ = -94.85 $\sigma_{r\,max}$ = 0.105811	$\sigma_{r\,min}$ = -100 $\sigma_{r\,max}$ = $0.854.10^{-6}$

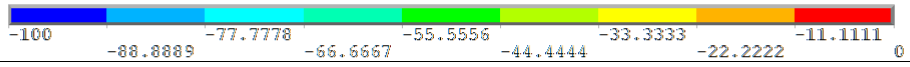
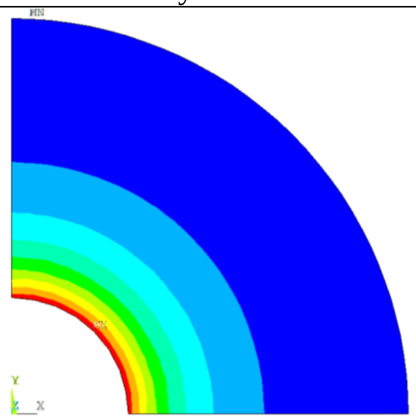
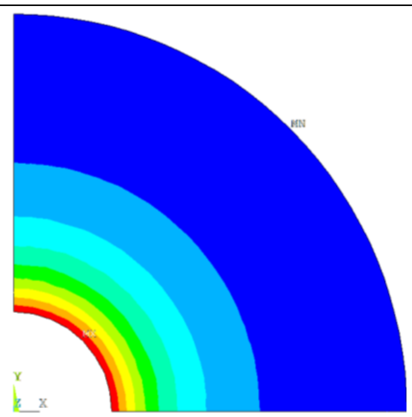


Figure 18: Radial stress σ_r

Ansys results	MLS-C results
	
$\sigma_{t\,min}$ = 13.4384 $\sigma_{t\,max}$ = 107.279	$\sigma_{t\,min}$ = 13.3825 $\sigma_{t\,max}$ = 109.736

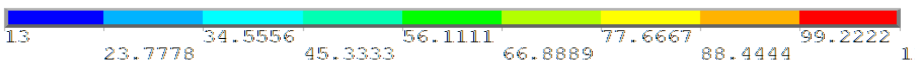
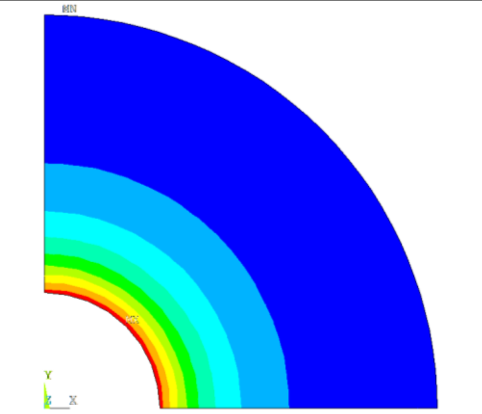
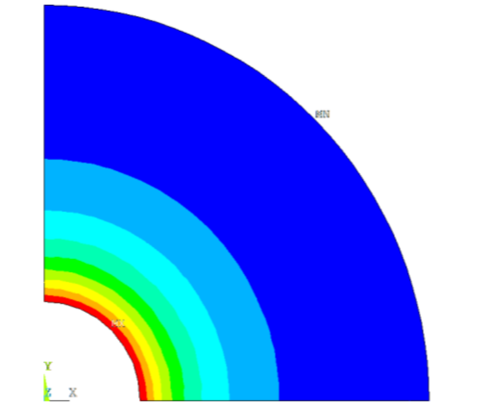


Figure 19: Hoop stress σ_t

Ansys results	MLS-C results
	
$\sigma_{VM\,min}$ = 13.4916 $\sigma_{VM\,max}$ = 174.312	$\sigma_{VM\,min}$ = 13.3825 $\sigma_{VM\,max}$ = 181.702

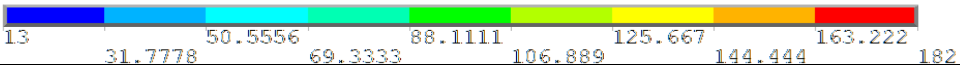


Figure 20: Von Mises stress σ_{VM}

Quadratic interpolation. Meshes with Quadrilateral elements

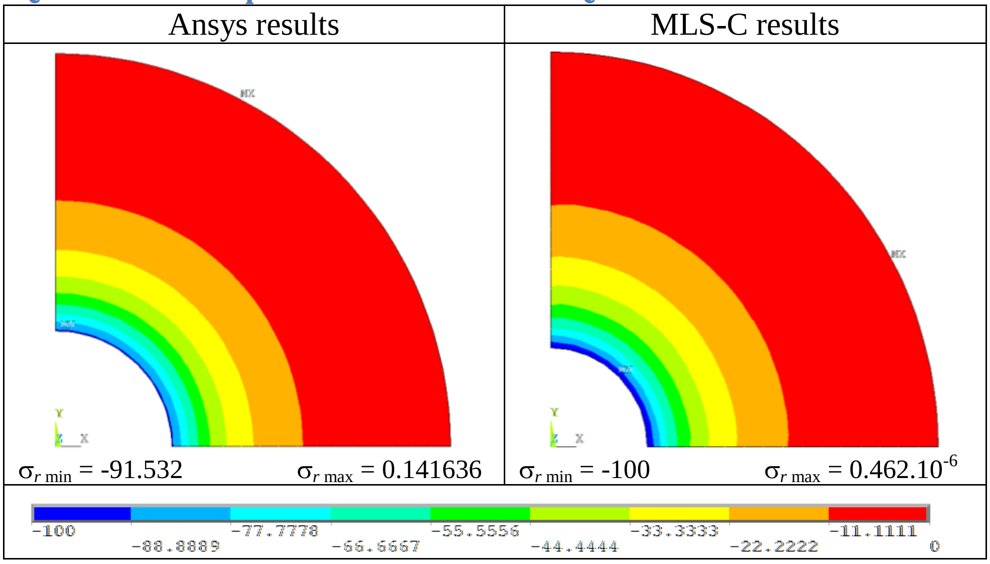


Figure 21: Radial stress σ_r

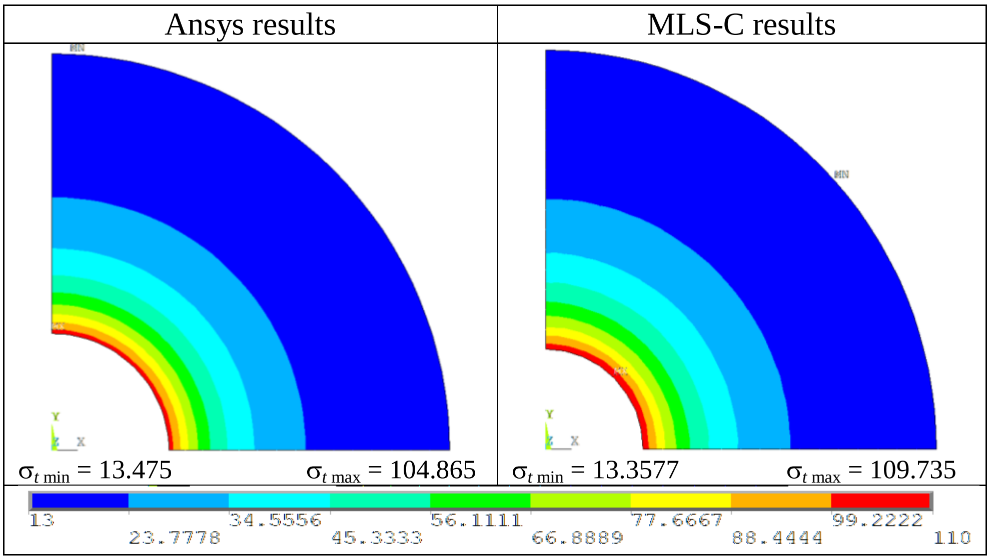
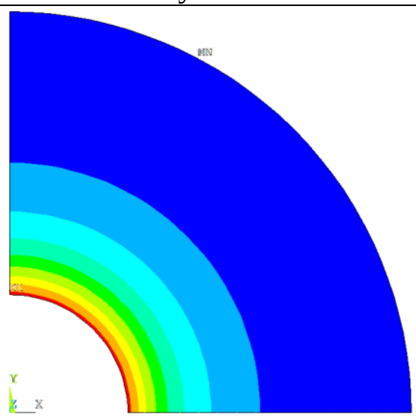
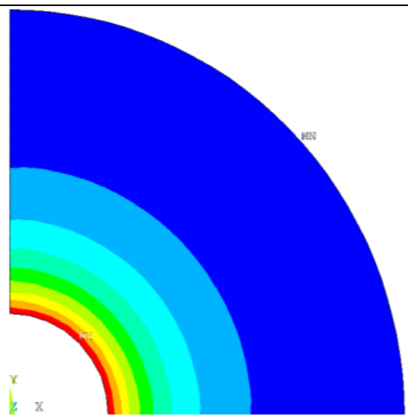


Figure 22: Hoop stress σ_t

Ansys results	MLS-C results
	
$\sigma_{VM\ \min}$ = 13.5463 $\sigma_{VM\ \max}$ = 170.252	$\sigma_{VM\ \min}$ = 13.3577 $\sigma_{VM\ \max}$= 181.701

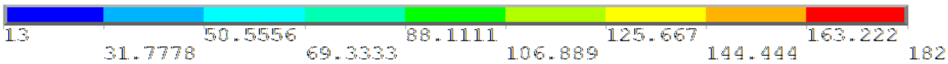
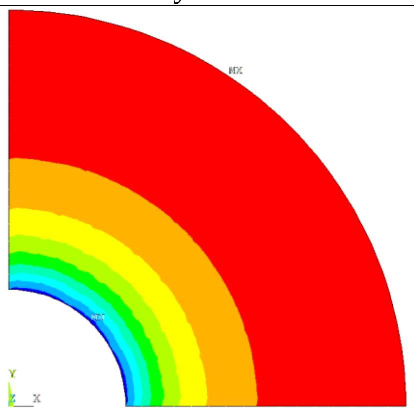
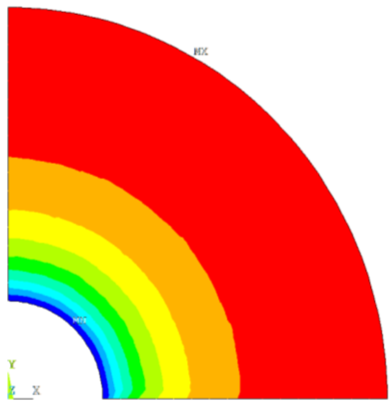


Figure 23: Von Mises stress σ_{VM}

Quadratic interpolation. Meshes with Quadrilateral and Triangular elements

Ansys results	MLS-C results
	
$\sigma_{r\ min}$ = -95.5859 $\sigma_{r\ max}$ = 0.09498	$\sigma_{r\ min}$ = -100 $\sigma_{r\ max}$ = $0.167.10^{-6}$

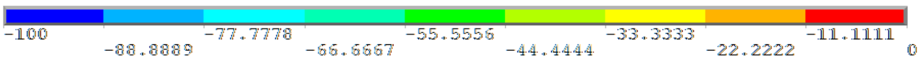
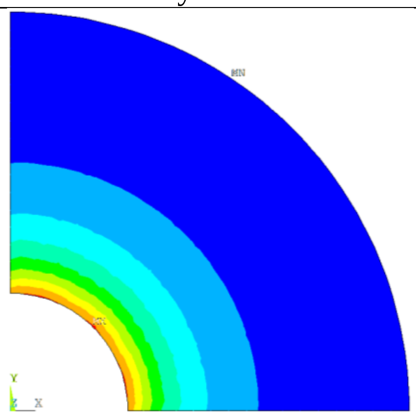
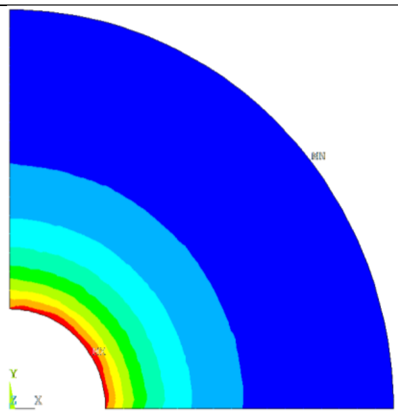


Figure 24: Radial stress σ_r

Ansys results	MLS-C results
	
$\sigma_{t\ min}$ = 13.4287 $\sigma_{t\ max}$ = 109.576	$\sigma_{t\ min}$ = 13.3738 $\sigma_{t\ max}$ = 117.508

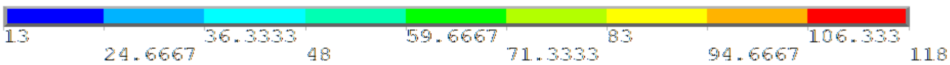
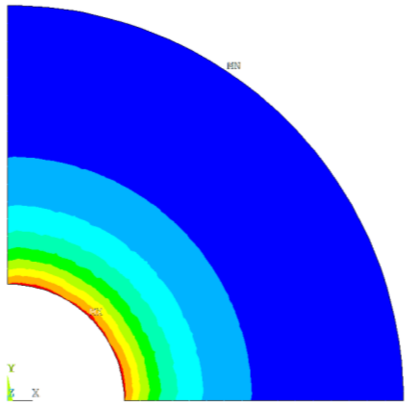
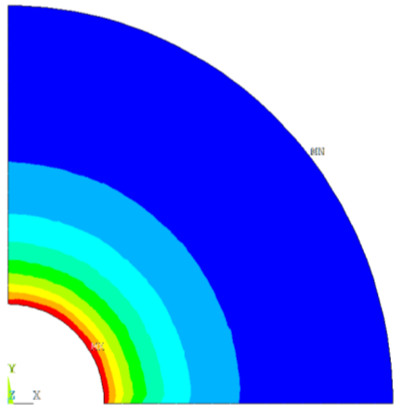


Figure 25: Hoop stress σ_t

Ansys results	MLS-C results
	
$\sigma_{VM\ \min}$ = 13.4764 $\sigma_{VM\ \max}$ = 177.813	$\sigma_{VM\ \min}$ = 13.3738 $\sigma_{VM\ \max}$ = 188.571

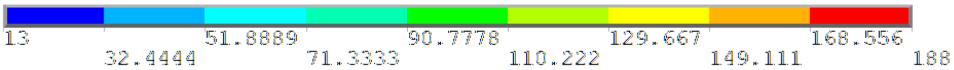
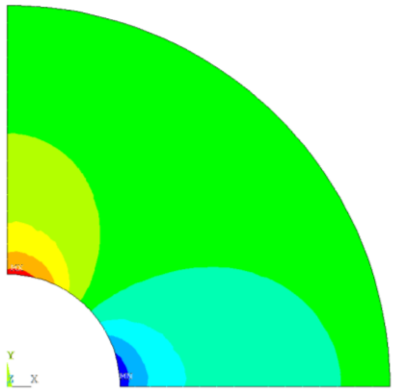
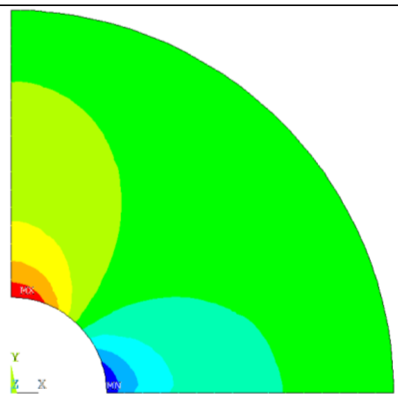


Figure 26: Von Mises stress σ_{VM}

Ansys results	MLS-C results
	
SX min = -92.0642 SX max = 105.535	SX min = -100 SX max = 110.333

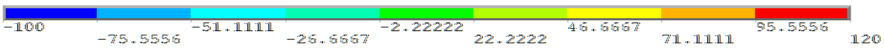


Figure 27: Cartesian stress σ_x

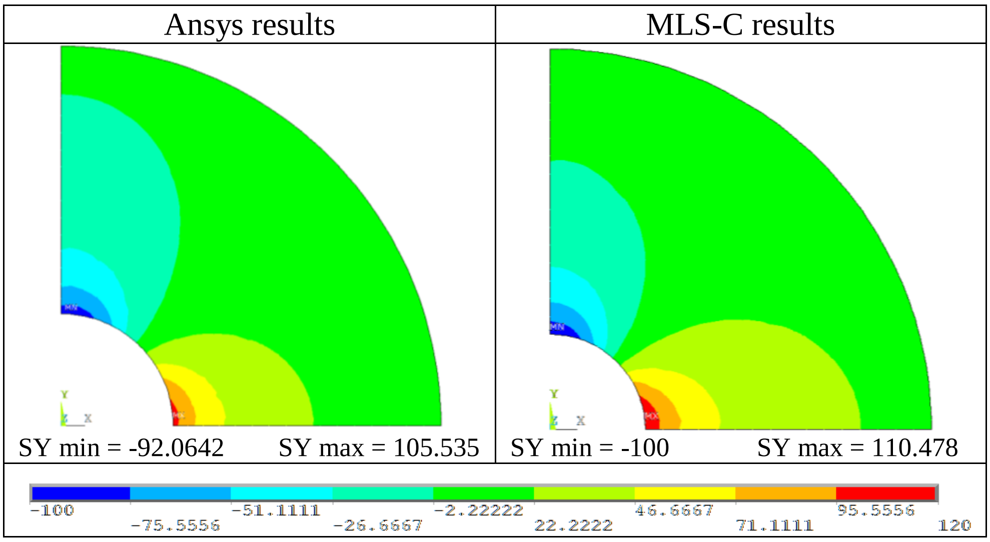


Figure 28: Cartesian stress σ_y

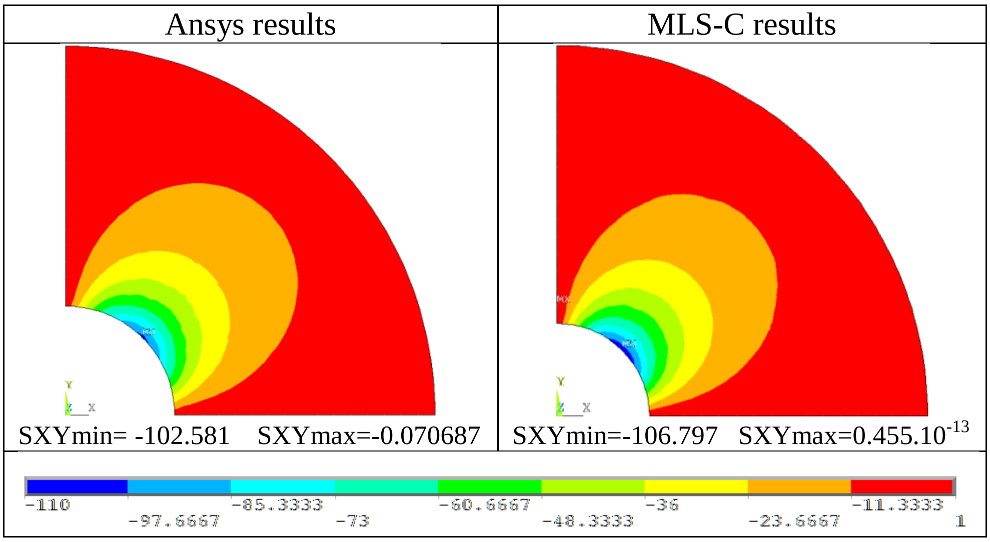


Figure 30: Cartesian Shear stress σ_{xy}

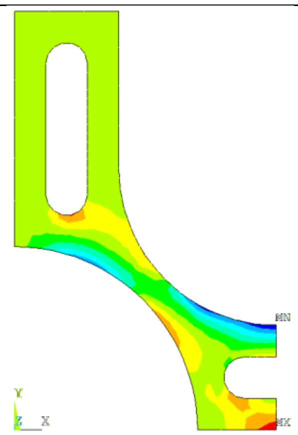
Results discussion

The results clearly show that the results provided by the MLS-C technique are more accurate than the results directly provided by Ansys. The distribution of the stress fields is smoother with the MLS-C technique. The minimum and maximum values of stresses are more accurately evaluated with the MLS-C technique.

5.2. Support

For the support we have:

Linear interpolation.

Ansys results	MLS-C results
	
SX min = -544.897 SX max = 418.916	SX min = -664.369 SX max = 479.771

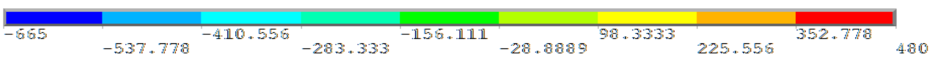
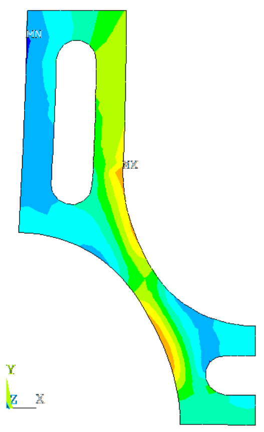
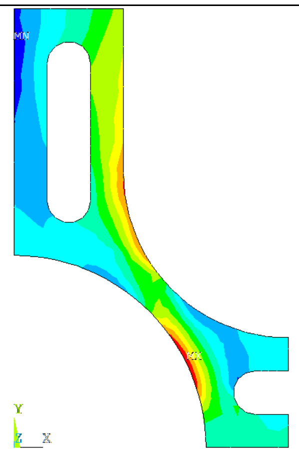


Figure 31: Cartesian stress σ_x

Ansys results	MLS-C results
	
SY min = -232.872 SY max = 680.89	SY min = -329.98 SY max = 801.247

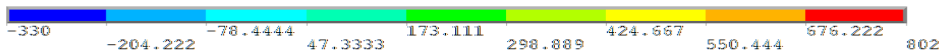
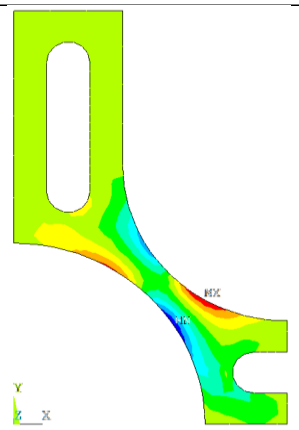


Figure 32: Cartesian stress σ_y

Ansys results	MLS-C results
	
SXYmin=-401.586 SXYmax=244.764	SXY min = -487.014 SXY max = 337.209

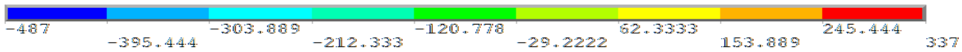
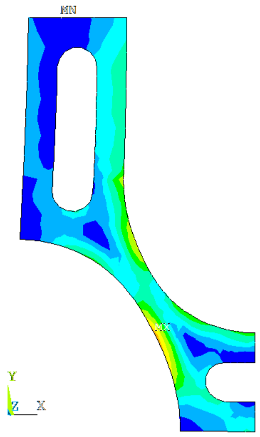
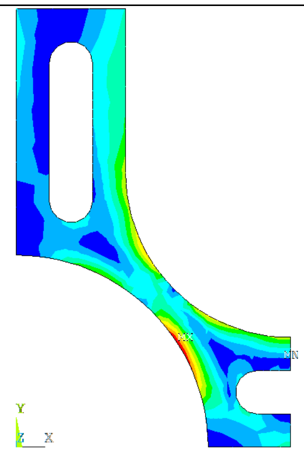


Figure 33: Cartesian shear stress σ_{xy}

Ansys results	MLS-C results
	
$\sigma_{VM\text{min}}$=6.48891 $\sigma_{VM\text{max}}$=894.627	$\sigma_{VM\text{min}}$ = 9.15775 $\sigma_{VM\text{max}}$ = 1074.62

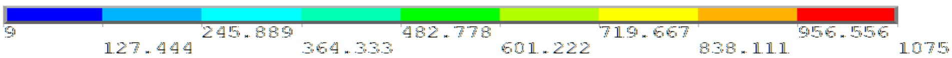
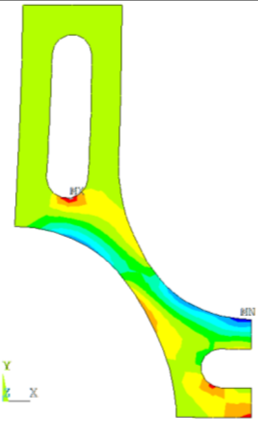
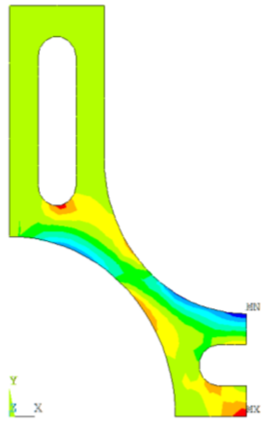


Figure 34: Von Mises stress σ_{VM}

Quadratic interpolation.

Ansys results	MLS-C results
	
SX min = -656.02 SX max = 573.358	SX min = -698.064 SX max = 514.222

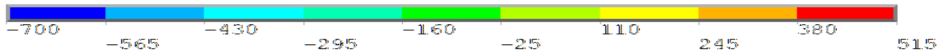
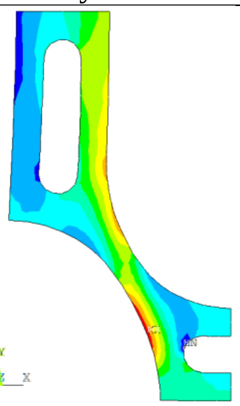
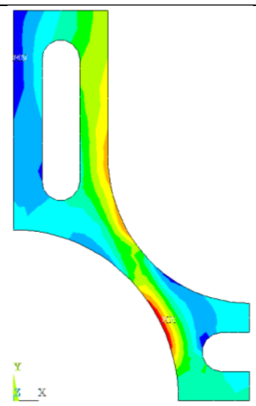


Figure 35: Cartesian stress σ_x

Ansys results	MLS-C results
	
SYmin = -389.251 SYmax = 778.739	SYmin = -324.128 SYmax = 825.448

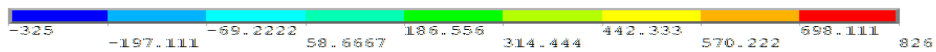
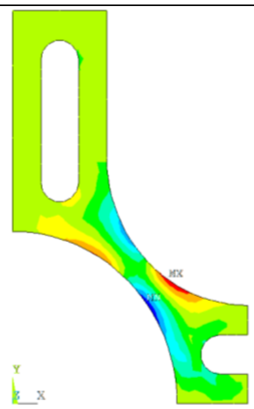


Figure 36: Cartesian stress σ_y

Ansys results	MLS-C results
	
SXYmin =-476.142 SXYmax =328.919	SXYmin =-509.332 SXYmax =365.664

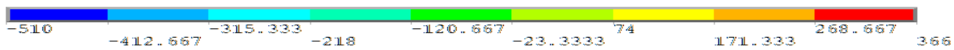
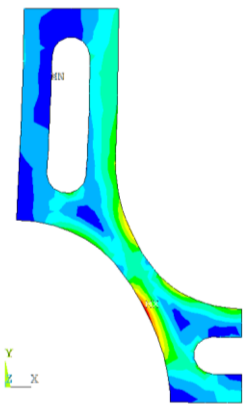
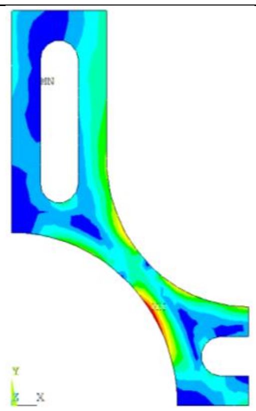


Figure 36: Cartesian shear stress σ_{xy}

Ansys results	MLS-C results
	
σ_{VMmin} = 9.65304 σ_{VMmax} = 1057.59	$\sigma_{VM\min}$ = 5.78102 $\sigma_{VM\max}$ = 1113.49

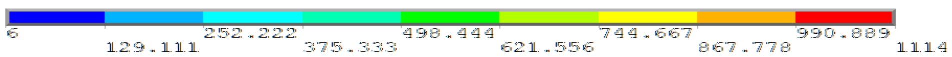


Figure 37: Von Mises stress σ_{VM}

We can clearly see from the figures that the MLS-C results are smoother than those provided by Ansys. In any case it is difficult to see which

results are more accurate. To chech the accuracy we have plotted the stresses along some of the sides of the support (left and right vertical sides and lower and upper arc) as we know that the tractions along these free surfaces are zero. In addition the tractions on the lower segment must be equal to the applied traction of 100 MPa.
The following figures compare, as an example of the performance of the MLS-C library, the stresses along the left vertical side and along the upper arc, with the results obtained as the direct output of Ansys.

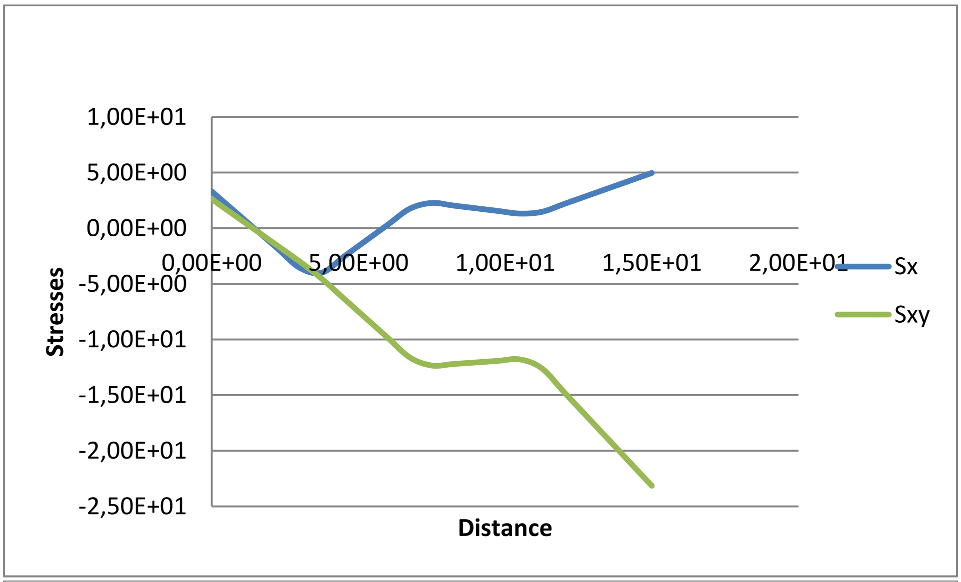
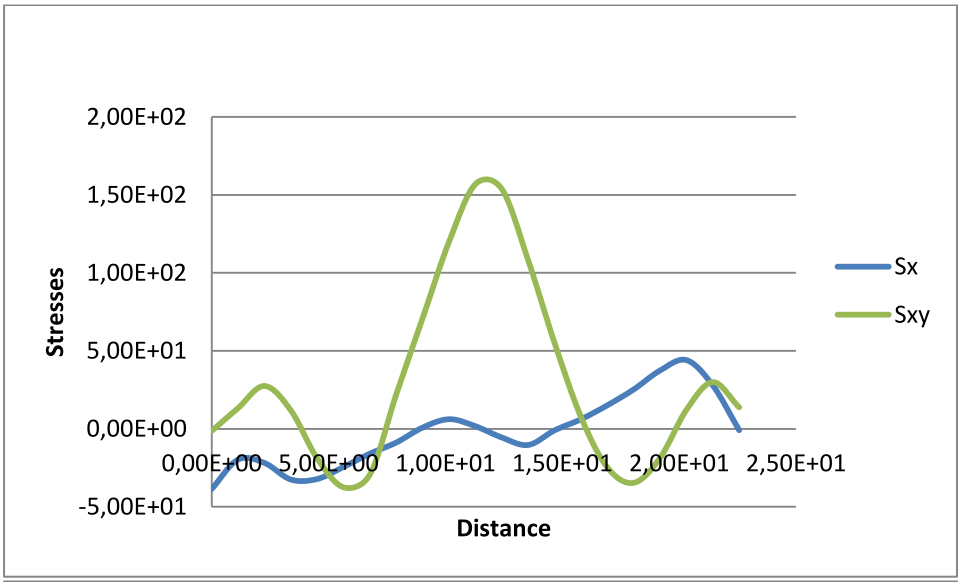


Figure 38: Comparing Stresses at vertical right side of the support for linear interpolation



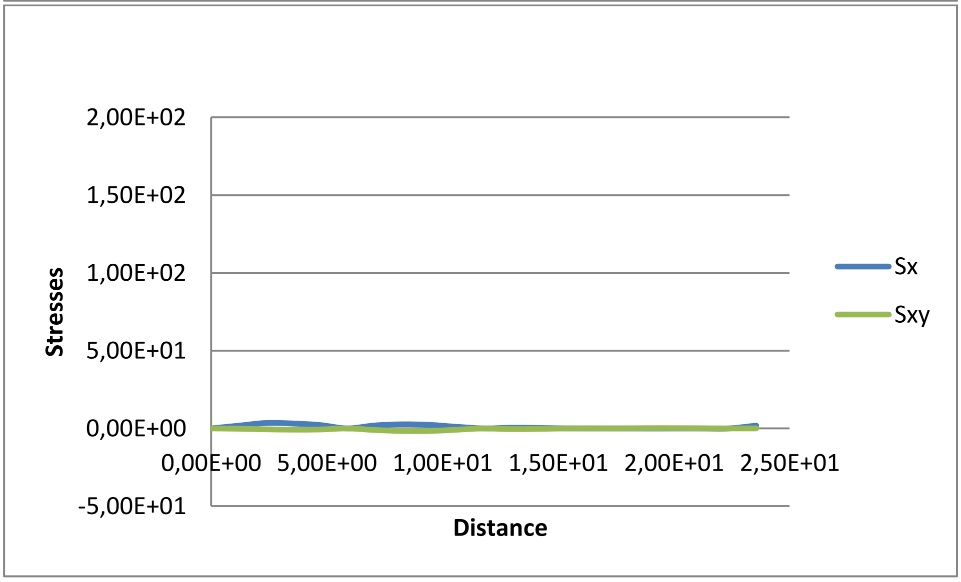
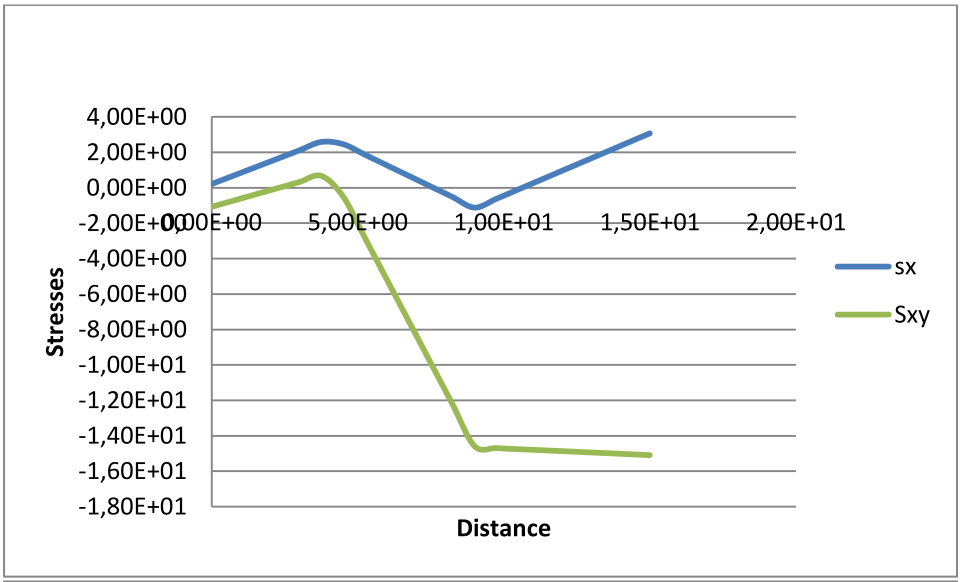


Figure 39: Comparing Stresses at the upper arc of the support for linear interpolation



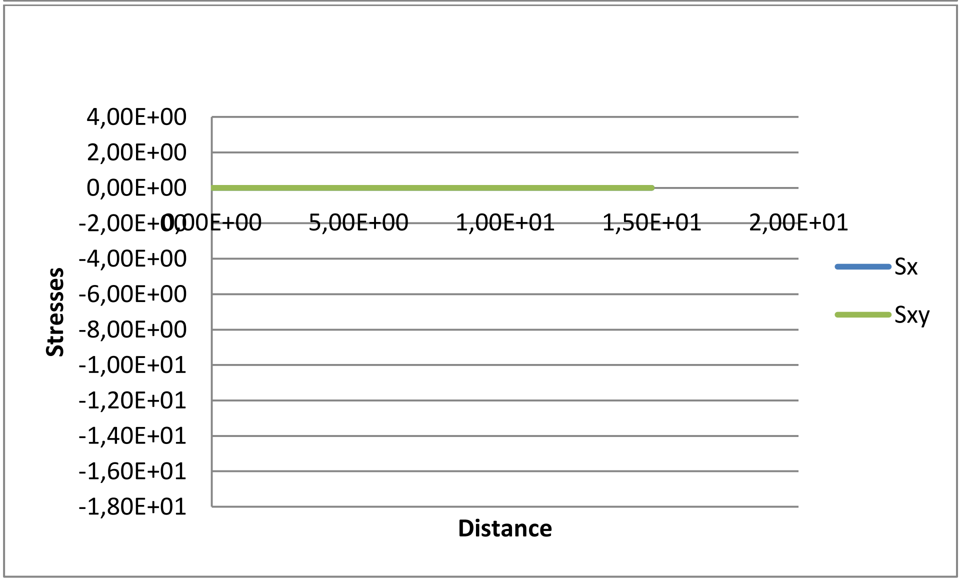


Figure 40: Comparing Stresses at the vertical right of the support for quadratic interpolation

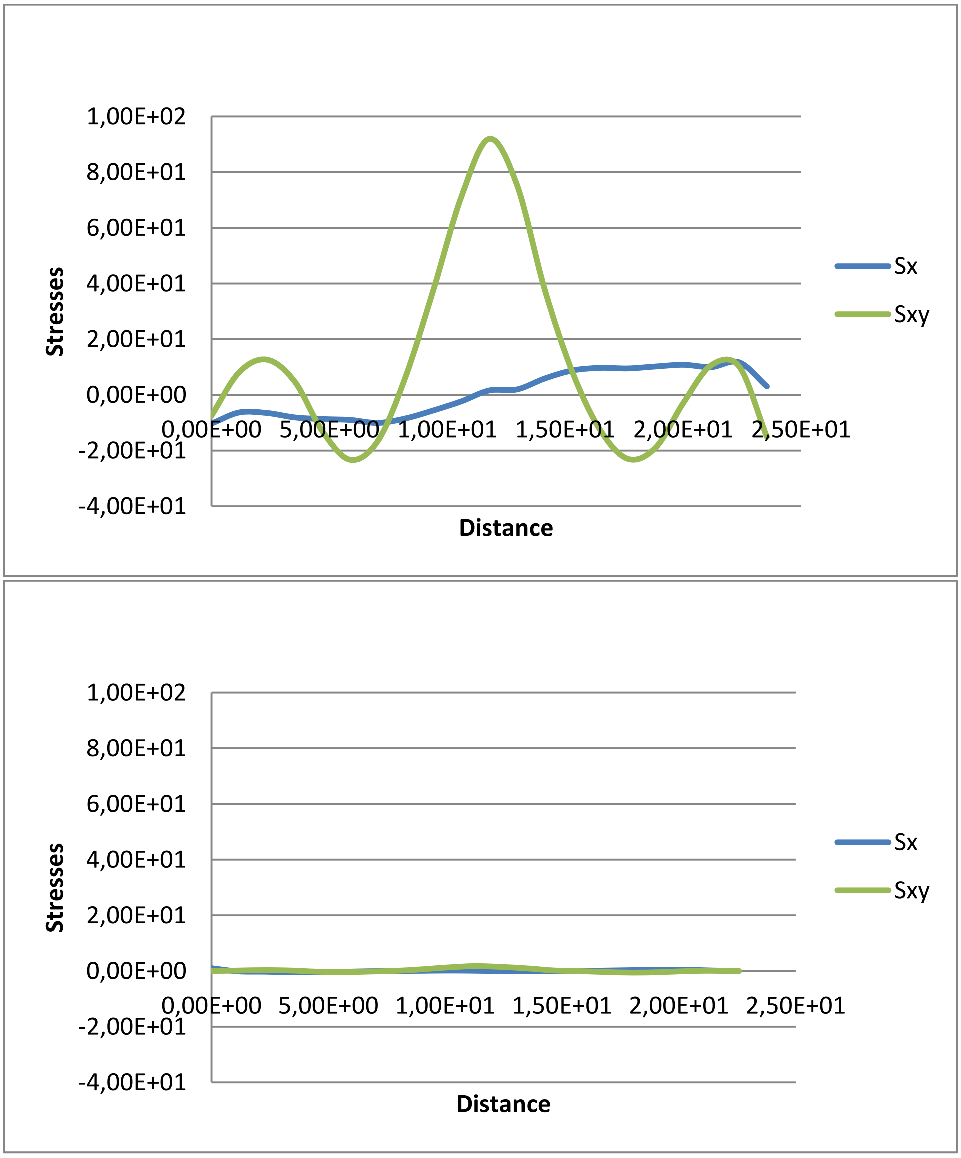


Figure 40: Comparing Stresses at the upper arc of the support for quadratic interpolation

We notice the difference between each two graphs:

For the bottom line of the support we notice that with:

- FEA and MLS-C the stresse Sxy are null and the Sy is the applied

For the inner and outter arc:

- The radial and tangential have to be nulls with the MLS-C technique, the FAE analysis give the opposite.

For the left and right side of the support:

- The Sxx and Sxy stresses are nulls when we use the recovery technique. With Ansys we obtain different results.

The recovered stress field resulted from the MLS technique is continuous and nearly equilibrated, because of these two things:

a) The nearest point approach which introduces the satisfaction of the boundary equations.
b) The Lagrange Multipliers technique to satisfy the internal equilibrium equation.

6. Conclusion

It is interesting to do a recapitulation of what was the main factors in the process.

- The reconstructed field by MLS technique allows for a better solution than that directly obtained from FEM stresses.
- The present library, implemented in MATLAB, can be used for linear and quadratic 2D triangular and quadrilateral elements.
- An interface with Ansys has also been created.
- The library has been modified in order to be published under the GNU GPL for a free use.

It is also interesting to know what are the possible upgrades that need to be studied further in future work

- Develop a library for the SPR recovery technique.
- The use of the library extended to other commercial program beside Ansys as ABAQUS and VISUAL NASTRAN.
- The GNU GPL publishing for a free use.

7. References

[1] O.C. Zienkiewicz, J.Z. Zhu, A simple error estimator and adaptive procedure for practical engineering analysis, Int.J. Numer. Methods Eng. 24 (1987) 337—357.

[2] Ródenas JJ, Tur M, Fuenmayor FJ, Vercher A. Improvement of the superconvergent patch recovery technique by the use of constraint equations: the SPR-C technique. International Journal for Numerical Methods in Engineering 2007;70(6):705{727, doi:10.1002/nme.1903.

[3] Ródenas JJ, González-Estrada OA., Fuenmayor FJ, Chinesta F. Enhanced error estimator based on a nearly equilibrated moving least squares recovery technique for FEM and XFEM. Computational Mechanics. 52(2), 321-344 (2013).

[4] Matlab® 8.1.0.604 (R2013a), The MathWorks Inc., Natick, MA, 2013

[5] ANSYS® Academic Research, Release 14.5, Help System, ANSYS, Inc.

[6] https://www.gnu.org/

[7] Bain, Gallego, Martínez Ribas y Rius, Aspectos legales y de explotación del software libre, parte 1 (2004)

[8] Vidal, Informe sobre licencias libres (2008).

[9] Gómez Padilla, La validez jurídica de la licencia GPL versión 3 en el marco normativo de los derechos de autor en Colombia (2011).

[10] Stallman, Software libre para una sociedad libre. Madrid: Traficantes de Sueños (2004).

[11] http://en.wikipedia.org/wiki/Affero_General_Public_License

[12] https://www.gnu.org (Why you shouldn't use the Lesser GPL for your next library)

8. Annex

In this section we will post all function we have already mentioned in the latter section involved in the recovery process. First the Ansys macro used to invoque the MLS-C library is described. Afterwards the rest of the subroutines are shown. In this case we have used the output of the Matlab Publishing tools to show this files in a more convinient way than the plain text files used when writing the code.

This annex also includes the text of the GNU LGPL.

8.1. GNU LGPL Terms

Version 3, 29 June 2007

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

0. Additional Definitions.

As used herein, “this License” refers to version 3 of the GNU Lesser General Public License, and the “GNU GPL” refers to version 3 of the GNU General Public License.

“The Library” refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An “Application” is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A “Combined Work” is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the “Linked Version”.

The “Minimal Corresponding Source” for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The “Corresponding Application Code” for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

- a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or

- b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

- a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.
- b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

- a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.
- b) Accompany the Combined Work with a copy of the GNU GPL and this license document.
- c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.

- d) Do one of the following:
  - 0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.
  - 1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.
- e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

- a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.
- b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.

8.2.Commercial software adaptation. Ansys

The input variables used in the MLS function implemented in MATLAB are obtained by generating a simple example of application in Ansys in .txt format.

These files are cleaned by removing all headers and blanks spaces and only contain needed data.

A .mac (macro) files are used to obtain the results (input variables) and to visualize the recovered stress field.

The first one is called **`MLSRecovery_1.mac`**

```
! ----------------------------------------------------------
! Define format for output data.
! Define Page layout: defile a huge number of lines so that  all data
fits
! within one page,5000 characters per line so that the stresses of
each
! point fit within one line,-1 removes headers in batch mode [/BATCH],
! Diverted [/OUTPUT], or interactive GUI [/MENU] output.
! Remove the header lines.
! ----------------------------------------------------------
/FORMAT,10,E,20,12
/PAGE,10000000,5000,-1,5000
/HEADER,OFF,OFF,OFF,OFF,OFF,OFF

! ---------------------------------------------------------------------
! Create output in .txt files with the FE results for the MLS
subroutine.
! PRESOL contains the stresses at nodes of each element, whose values
have been substituted by the stress values at Gauss points. The
smoothing subroutine will have to take this information and locate it
back at its original position: the Gauss points.
! NLIST  (x,y) nodal coordinates
! ELIST  Element's topology (nodes included in each element)
! ----------------------------------------------------------
! Create PRESOL
/OUTPUT,.\PRESOL.txt
PRESOL,S,COMP
/OUTPUT
/OUTPUT,.\NLIST.txt
NLIST,,,,COORD
/OUTPUT
/OUTPUT,.\ELIST.txt
ELIST
/OUTPUT

! ----------------------------------------------------------
! From Ansys, open Matlab and run the QuadraticElms code.
! ----------------------------------------------------------
/SYS,matlab -noFigureWindows -r "try; run('QuadraticElems.m'); catch;
end;"
```

8.3. QuadraticElems.m

This Matlab subroutine calls the MLS function to evaluate a recovered stress field from a 2D stress field evaluated using the finite element method for quadratic elements. A similar subrutine called LinearElems.m is used to deal with linear elements.

```
%************************************************************************
% This file is part of the MLS-C© Library; you can redistribute it and/or
% modify it under the terms of the GNU Lesser General Public License as
% published by the Free Software Foundation; either version 3 of the
% License,or any later version.
% This program is distributed in the hope that it will be useful but
% WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General
% Public License for more details.
% You should have received a copy of the GNU Lesser General Public License
% along with GNU; see the file COPYING.  If not, write to the Free Software
% Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA
% or see <http://www.gnu.org/licenses/>.
%************************************************************************
% Created by:
%     E Nadal, M Ouallal, JJ Rodenas, F Chinesta and FJ Fuenmayor
%
% Release 1.0
% Date: 18/07/2014
%************************************************************************
```

This Matlab(c) subroutine calls the Moving Least Squares function to evaluate a recovered stress field from a 2D stress field evaluated using the finite element method for quadratic elements.

```
% Remove all blanck lines and headers from the output files (Element
% solution, Topology, Node Coordinates) generated by Ansys to be used as
% input argument when calling the MLS subroutine.

AnsysRelease = 14; %Valid releases are 12 and 14
```

Read NLIST.txt   Nodal information

```
f = fopen('NLIST.txt', 'rt');
if AnsysRelease ==14
    c = textscan(f,'%s','Delimiter','%s','headerlines',4,...
        'CollectOutput',true); %Read file
    lines = c{1};              %Store nodal coords
elseif AnsysRelease == 12
    c = textscan(f,'%s','Delimiter','%s','headerlines',3,...
        'CollectOutput',true); %Read file
    lines = c{1};              %Store nodal coords
    lines(1:21:end) = [];      %Remove blank lines
else
    err ('Ansys release not supported');
end
fclose(f);

coordinatesXY = zeros(numel(lines),4);

for i = 1:numel(lines)
    coordinatesXY(i,:) = coordinatesXY(i,:) + str2num(lines{i});

end

coordinatesXY(:,[1,4]) = [];
Xcoord = coordinatesXY(:,1);
Ycoord = coordinatesXY(:,2);
```

Read ELIST.txt   Element information

```
f = fopen('ELIST.txt','rt');
if AnsysRelease ==14
    c = textscan(f,'%s','Delimiter','%s','headerlines',4,...
        'CollectOutput',true);
    lines = c{1};
elseif AnsysRelease == 12
    c = textscan(f,'%s','Delimiter','%s','headerlines',2,...
        'CollectOutput',true);
    lines = c{1};
    lines(1:21:end) = [];
else
```

```
    err ('Ansys release not supported');
end
fclose(f);

topology = zeros(numel(lines),14);

for i = 1:numel(lines)
    topology(i,:) = topology(i,:) + str2num(lines{i});

end

topology(:,1:6) = [];

%
```

Read PRESOL.txt Element information

```
f = fopen('PRESOL.txt','rt');
c = textscan(f,'%s','Delimiter','%s','headerlines',2,...
    'CollectOutput',true);

lines= c{1};
lines(1:6:end)=[];
lines(1:5:end)=[];
GaussStresses = zeros(numel(lines),7);
NumGP         = 4;
NumElem       = size(topology,1);

for i = 1:numel(lines)
    GaussStresses(i,:) = GaussStresses(i,:) + str2num(lines{i});

end

GaussStresses(:,[4,6,7]) = [];

Temporary_Quad = mat2cell(GaussStresses,NumGP*ones(NumElem,1),NumGP);

for i = 1:NumElem
Temp = Temporary_Quad{i,1};
    for k = 2:NumGP
```

```
        if Temp(k,1)==Temp(k-1,1)
            Temp(k-1,:)=[];
        end
    end
Temporary_Quad{i,1}=Temp;
end

GaussStresses    = cell2mat(Temporary_Quad);
NodeNumQuad = GaussStresses(:,1);

fclose(f);
```

Obtaining of GP coordinates in global system

Ansys does not dispose of commands that calculate the coordinate of the integration points in the global coordinate system, to do so, this part of the code was implemented to get the coordinates of these points in the global system for quadrilateral, triangle and mixed mesh.

```
Dimensions     = 2;

% Number of side per element
NSidesElemTri   = 3;
NSidesElemQuad = 4;

% Number of GP per element
NumPtsTri    = 6;
NumPtsQuad   = 9;

Grade          = 2;
CaldN          = 0;

% Invoke pgauss function to obtain the GP coordinates in the local system
% reference.
CoordinatesTri   = pgauss(Dimensions, NSidesElemTri, NumPtsTri);
CoordinatesQuad = pgauss(Dimensions,NSidesElemQuad,NumPtsQuad);

% Ansys provide information of gauss points in vertex nodes so it is
% suitable to keep only the gauss point coordinates close to the vertex
% nodes.
CoordinatesTri([4,5,6],:)      = [];
CoordinatesQuad([2,4,5,6,8],:) = [];
```

```
% Shape function for triangular and quadrilateral elements.
NTri = shape_f_2d(CoordinatesTri,Grade,NSidesElemTri,CaldN);
NQuad = shape_f_2d(CoordinatesQuad,Grade,NSidesElemQuad,CaldN);

NumElem       = size(topology,1);
NnodeElem     = size(topology,2);

% GP for quadrilateral elements
if size(GaussStresses,1)==NumElem*NumPtsQuad
   XQuad = zeros(NumElem,NnodeElem);
   YQuad = zeros(NumElem,NnodeElem);

   for i=1:NumElem
       for j=1:NnodeElem
           XQuad(i,j) = Xcoord(topology(i,j));
           YQuad(i,j) = Ycoord(topology(i,j));
       end
   end

   GPXQuad            = XQuad*NQuad';
   GPYQuad            = YQuad*NQuad';
   TempX              = mat2cell(GPXQuad,1*ones(length(topology),1),...
                        NumPtsQuad);
   TempY              = mat2cell(GPYQuad,1*ones(length(topology),1),...
                        NumPtsQuad);
   TempX              = cellfun(@transpose,TempX,'un',0);
   TempY              = cellfun(@transpose,TempY,'un',0);
   GaussPtCoordXQuad = cell2mat(TempX);
   GaussPtCoordYQuad = cell2mat(TempY);
   GaussPtCoordXY     = [GaussPtCoordXQuad GaussPtCoordYQuad];

% GP for quadrilateral and triangular elements
elseif (size(GaussStresses,1)< NumElem*NumPtsQuad) && ...
       (size(GaussStresses,1)> NumElem*NumPtsTri)
    X = zeros(NumElem,NnodeElem);
    Y = zeros(NumElem,NnodeElem);

    for i=1:NumElem
        for j=1:NnodeElem
        X(i,j) = Xcoord(topology(i,j));
        Y(i,j) = Ycoord(topology(i,j));
        end
```

```
end
Temporary1 = mat2cell(X,1*ones(NumElem,1),NnodeElem);
Temporary2 = mat2cell(Y,1*ones(NumElem,1),NnodeElem);
T1          = zeros(NumElem,NnodeElem);
T2          = zeros(NumElem,NnodeElem);
X_tri       = zeros(NumElem,NnodeElem);
Y_tri       = zeros(NumElem,NnodeElem);

for k = 1:NumElem
    Temp1 = Temporary1{k};
    Temp2 = Temporary2{k};
    for m = 2:NnodeElem
        Tg = unique(topology(k,:)','rows');
        if size(Tg',2)==6
           X_tri(k,:)       = X(topology(k,:));
           Y_tri(k,:)       = Y(topology(k,:));
           T1               = X_tri(k,:);
           X_tri(k,:)       = X(k,:);
           Temporary1{k,1} = zeros(1,NnodeElem);
           Temporary2{k,1} = zeros(1,NnodeElem);
           T2               = Y_tri(k,:);
           Y_tri(k,:)       = Y(k,:);
        end
    end
end

remCol1      = unique(X_tri','rows','stable');
XtriCoord    = remCol1';
remCol2      = unique(Y_tri','rows','stable');
YtriCoord    = remCol2';
GPX_tri_Mix  = XtriCoord*NTri';
GPY_tri_Mix  = YtriCoord*NTri';
XquadCoord   = cell2mat(Temporary1);
YquadCoord   = cell2mat(Temporary2);
GPXquadr_Mix = XquadCoord*NQuad';
GPYquadr_Mix = YquadCoord*NQuad';
GPXtri_Mix   = [GPX_tri_Mix zeros(NumElem,1)];
GPYtri_Mix   = [GPY_tri_Mix zeros(NumElem,1)];

for k = 1:NumElem
    for m = 1:NnodeElem
        if GPXquadr_Mix(k,:)==0
            GPXquadr_Mix(k,:)=GPXtri_Mix(k,:);
```

```
            end
            if GPYquadr_Mix(k,:)==0
                GPYquadr_Mix(k,:)=GPYtri_Mix(k,:);
            end
        end
    end

    TempX = mat2cell(GPXquadr_Mix,1*ones(length(topology),1),NumPtsQuad);
    TempY = mat2cell(GPYquadr_Mix,1*ones(length(topology),1),NumPtsQuad);
    TempX = cellfun(@transpose,TempX,'un',0);
    TempY = cellfun(@transpose,TempY,'un',0);

    GaussPtCoordX_Mix = cell2mat(TempX);
    GaussPtCoordY_Mix = cell2mat(TempY);
    GaussPtCoordX_Mix(all(GaussPtCoordX_Mix==0,2),:) = [];
    GaussPtCoordY_Mix(all(GaussPtCoordY_Mix==0,2),:) = [];
    GaussPtCoordXY    = [GaussPtCoordX_Mix GaussPtCoordY_Mix];

% GP for triangular elements
elseif size(GaussStresses,1)==NumElem*NumPtsTri
    Temp         = topology';
    Temp2        = unique(Temp,'stable','rows');
    topologyTemp = Temp2';
    NumElemTri   = size(topologyTemp,1);
    NnodeElemTri = size(topologyTemp,2);
    XTri         = zeros(NumElemTri,NnodeElemTri);
    YTri         = zeros(NumElemTri,NnodeElemTri);

    for h=1:NumElemTri
        for j=1:NnodeElemTri
            XTri(h,j) = Xcoord(topologyTemp(h,j));
            YTri(h,j) = Ycoord(topologyTemp(h,j));
        end
    end

    GPXTri = XTri*NTri';
    GPYTri = YTri*NTri';
    TempX  = mat2cell(GPXTri,1*ones(length(topologyTemp),1),NumPtsTri);
    TempY  = mat2cell(GPYTri,1*ones(length(topologyTemp),1),NumPtsTri);
    TempX  = cellfun(@transpose,TempX,'un',0);
    TempY  = cellfun(@transpose,TempY,'un',0);
    GaussPtCoordX  = cell2mat(TempX);
    GaussPtCoordY  = cell2mat(TempY);
```

```
    GaussPtCoordXY = [GaussPtCoordX GaussPtCoordY];

end

GaussStresses(:,1) = [];
% Matrix containing stresses and node coordinates.
FinalData = [GaussPtCoordXY GaussStresses];
```

Invoking MLS function and obtaining recovered stresses

This function load the data for the MLS smoothing using quadratic elements

```
% Define the path where the data is available

FieldFEGP    = FinalData(:,3:5);
XYZGP        = FinalData(:,1:2);

NumGP       = size(XYZGP,1);
NumNodes    = size(coordinatesXY,1);
XYZout      = [ XYZGP ; coordinatesXY];

% Invoke the MLS function for quadratic element to obtain the recovered
% stress field at integration point and nodes.
[FieldStGP_Quad]      = MLS(FieldFEGP,XYZGP,XYZout,Mesh);
FieldStGP_Quad_GP     = FieldStGP_Quad(1:NumGP,:);
FieldStGP_Quad_Node = FieldStGP_Quad(1+NumGP:end,:);
```

Prepare data to create to invoke the CreateMacroToGetMLSdata.m

```
Quadratic_SigX  = zeros(size(topology,1),size(topology,2));
Quadratic_SigY  = zeros(size(topology,1),size(topology,2));
Quadratic_SigXY = zeros(size(topology,1),size(topology,2));

for i = 1:size(topology,1)
    for j = 1:size(topology,2)
        Quadratic_SigX(i,j)  = FieldStGP_Quad_Node(topology(i,j),1);
        Quadratic_SigY(i,j)  = FieldStGP_Quad_Node(topology(i,j),2);
```

```
        Quadratic_SigXY(i,j) = FieldStGP_Quad_Node(topology(i,j),3);
    end
end

Quadratic_SigZ  = zeros(size(topology,1),size(topology,2));
Quadratic_SigXZ = zeros(size(topology,1),size(topology,2));
Quadratic_SigYZ = zeros(size(topology,1),size(topology,2));

% Prepare the input argument for the CreateMacroToGetMLSdata function
OutputXYZ      = [coordinatesXY zeros(length(coordinatesXY),1)];
OutputTopology = unique(topology(:,1:4));
ShowUX         = zeros(length(OutputTopology),1);
ShowVY         = zeros(length(OutputTopology),1);
ShowSX         = Quadratic_SigX(:,1:4);
ShowSY         = Quadratic_SigY(:,1:4);
ShowTXY        = Quadratic_SigXY(:,1:4);
ShowSZ         = Quadratic_SigZ(:,1:4);
ShowTXZ        = Quadratic_SigXZ(:,1:4);
ShowTYZ        = Quadratic_SigYZ(:,1:4);

CreateMacroToGetMLSdata(ShowUX,ShowVY,OutputXYZ,topology(:,1:4),ShowSX,Sho
wSY,ShowSZ,ShowTXY,ShowTXZ,ShowTYZ)
```

8.4. MLS.m function

This is the main subroutine of the MLS-C library

```
%*************************************************************************
% This file is part of the MLS-C© Library; you can redistribute it and/or
% modify it under the terms of the GNU Lesser General Public License as
% published by the Free Software Foundation; either version 3 of the
% License,or any later version.
% This program is distributed in the hope that it will be useful but
% WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General
% Public License for more details.
% You should have received a copy of the GNU Lesser General Public License
% along with GNU; see the file COPYING.  If not, write to the Free Software
% Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA
% or see <http://www.gnu.org/licenses/>.
%*************************************************************************
% Created by:
%    E Nadal, M Ouallal, JJ Rodenas, F Chinesta and FJ Fuenmayor
%
% Release 1.0
% Date: 18/07/2014
%*************************************************************************
function [FieldStGP]= MLS( FieldFEGP,XYZGP,XYZout )
```

This Matlab(c) subroutine uses a Moving Least Squares approach to evaluate a recovered stress field from a 2D stress field evaluated using the finite element method. The mathematical basis used to create this subroutine can be found in
Ródenas JJ, González-Estrada OA., Fuenmayor FJ, Chinesta F. *Enhanced error estimator based on a nearly equilibrated moving least squares recovery technique for FEM and XFEM*. Computational Mechanics. 52(2), 321-344 (2013).

A basic description of this subroutine can be found in the file MLStechnique.pdf.

INPUT AND OUTPUT

```
%---------------------------------------------------------------------
% INPUT VARIABLES
%   FieldFEGP   = FE stresses at sampling (integration) points
%   XYZGP       = Coordinates of the sampling (integration) points
%   XYZout      = Coordinates of the points where the recovered stress
%                 field has to be evaluated
% OUTPUT VARIABLES
%   FieldStGP = Reecovered stresses at XYZout
%
% MAIN INTERNAL VARIABLES
```

RESHAPE INPUT DATA

Input data is reshaped if needed because the code needs the data stored in columns

```
if size(FieldFEGP,2)> size(FieldFEGP,1)
    FieldFEGP = FieldFEGP';
end
if size(XYZGP,2)>size(XYZGP,1)
    XYZGP     = XYZGP';
end
if size(XYZout,2)>size(XYZout,1)
    XYZout    = XYZout';
end
```

DEFINITION OF SUBROUTINE PARAMETERS

```
% Load geometry data stored in text files GeoPoints.txt and Boundaries.txt
% -------------------------------------------------------------------------
load('GeoPoints.txt');
load('Boundaries.txt');

% Internal codes
% --------------

BoundImpTract       =40;                 % Boundary node with known
                                         % Normal and Tang stresses
Symmetry            =41;                 % Symmetry BC
IntEquil            = 1;                 % Internal equilibrium
FIE                 = 1;                 % Full internal equilibrium
ContourEquilWprime  = 1;                 % Contour equilibrium
TolBundary          = 1e-10;             % Tolerance for bound. conditions
                                         % Used to check if a point is over
                                         % a node on the boundary

% General configuration
% ---------------------
Noc                 = size(FieldFEGP,2); % Number of field components
DrawSupport         = 0;                 % Flag 0 plot / 1 do not plot
                                         % support of each point
SubSizes            = [3,6,10,15,21,28]; % Size of equation system for
each
                                         % polynomial degree of recovered
                                         % stress field
MaxFieldDegree      = 3;                 % Degree of the recovered field
NumGP               = size(XYZGP,1);     % Total number of points in XYZGP
Numout              = size(XYZout,1);    % Total number of points in
XYZout
NumInflPoints       = 151;               % Number of influence points in
                                         % the support

disp(['The number of influence points considered in the support is: '...
    num2str(NumInflPoints) '.'])

MinDegree = 100;                 % Keep polynomial expansion degree used
                                 % for each GP.
CondNum    = zeros(NumGP,1);     % Initilize the number of GP verctor.
```

ERROR ESTIMATION EQUATIONS

The ZZ error estimator will be evaluated as follows:

$$||\mathbf{e}||^2 = \int_\Omega (\sigma^* - \sigma^h)^T \mathbf{D}^{-1} (\sigma^* - \sigma^h) d\Omega$$

The recovered stress field components are obtained from a polynomial expansion, using the following expression:

$$\sigma_i^*(\mathbf{x}) = \mathbf{p}(\mathbf{x})\mathbf{a}_i(\mathbf{x}) \quad i = xx, yy, xy$$

For each stress components in the 2D case we have:

$$\sigma^*(\mathbf{x}) = \begin{bmatrix} \sigma^*_{xx}(x) \\ \sigma^*_{yy}(x) \\ \sigma^*_{xy}(x) \end{bmatrix} = \mathbf{P}(\mathbf{x})\mathbf{A}(\mathbf{x}) = \begin{bmatrix} p(x) & 0 & 0 \\ 0 & p(x) & 0 \\ 0 & 0 & p(x) \end{bmatrix} \begin{bmatrix} a_{xx}(x) \\ a_{yy}(x) \\ a_{xy}(x) \end{bmatrix}$$

MAXIMUM INFLUENCE RADIUS ASSUMING UNIFORM SAMPLING POINTS DISTRIBUTION

Now, the MLS technique considers a point χ in a support denominated Ω_x corresponding to a point $\mathbf{x}$ defined by a radius R_{Ω_x}. So the MLS approximation is given by:

$$\sigma_i^*(\mathbf{x}, \chi) = \mathbf{p}(\chi)\mathbf{a}_i(\mathbf{x}) \quad \forall \chi \in \Omega_x, \quad i = xx, yy, xy$$

```
XMin         = min(XYZGP(:,1));
XMax         = max(XYZGP(:,1));
YMin         = min(XYZGP(:,2));
YMax         = max(XYZGP(:,2));
RadProb      = sqrt((XMax-XMin)^2 + (YMax-YMin)^2)/2;
Area         = pi() * RadProb^2;
PointDensity = NumGP / Area;
MaxRadInf    = sqrt(4*NumInflPoints/(PointDensity * pi()));
disp(['The influence radius is: ' num2str(MaxRadInf) ...
    ' and the problem radius is: ' num2str(RadProb)]);
```

MSL LOOP AND DEFINITION OF EQUATIONS TO OBTAIN THE RECOVERED FIELD.

In order to obtain the recovered stress field $\sigma_i^*(\mathbf{x}) = p(\mathbf{x})a_i(\mathbf{x}) \quad i = xx, yy, xy$ we have to solve the linear system of equations $\mathbf{M}(\mathbf{x})\mathbf{A}(\mathbf{x}) = \mathbf{G}(\mathbf{x})$ so we can get the vector of coefficient $\mathbf{A}(\mathbf{x})$.

```
% Initilize output matrix
% -----------------------

FieldStGP = zeros(Numout,Noc);
Auxds     = zeros(Numout,1);

for iGP = 1:Numout
```

Initialize variables in case Lagrange for boundary

```
    Madd = [];                  % Main part of the system matrix
    Cadd = [];                  % Constraint equations
```

Evaluation of points within the support of the assembly point

```
    % Find influence GP.
    % -------------------
    XYPt = XYZout(iGP,:);      % Coordinates of the assembly GP

    if NumGP <= NumInflPoints
        InflPoints      = 1:NumGP;
    else
        dummyXGP        = (XYZGP(:,1)-XYPt(1));
        dummyYGP        = (XYZGP(:,2)-XYPt(2));
        dummyRadXYZGP   = sqrt(dummyXGP.^2 + dummyYGP.^2);
        SuitablePoints = find(dummyRadXYZGP <= MaxRadInf);
        [~,I]           = sort(dummyRadXYZGP(SuitablePoints),'ascend');
```

```
        if length(I) > NumInflPoints % Find number of influence points
            InflPoints = SuitablePoints(I(1:NumInflPoints));
        else
            InflPoints = SuitablePoints(I);
        end
    end


    NSP        = length(InflPoints); % Number of sampling points
    ds         = max(sqrt((XYZGP(InflPoints,1)-XYPt(1)).^2 +...
                  (XYZGP(InflPoints,2)-XYPt(2)).^2));% Influence radius
    Auxds(iGP) = ds;
```

There are n sampling points of coordinates χ within the support using the FE analysis where the stresses are already available. The normalized distance function is given to calculate the weighting function:

$$s = \frac{\|\mathbf{x} - \chi\|}{R_{\Omega_{\mathbf{x}}}}$$

```
    % Local coordinates of the influence points
    % -----------------------------------------

    X = (XYZGP(InflPoints,1) - XYPt(1))/ds;
    Y = (XYZGP(InflPoints,2) - XYPt(2))/ds;
    S = sqrt(X.^2 + Y.^2);


    % Check if the AssGP support in in contact with any boundary
    % ----------------------------------------------------------

    [BoundPts,AngleBoundPts,tBoundPts,BoundNum,NodeType,NBP]...
        = ClosestPointsOnBoundrs(XYPt(1),XYPt(2),ds,GeoPoints,Boundaries);

    % Local coordinates of the influence points on the boundaries.
    % ------------------------------------------------------------
    XBP = ( BoundPts(:,1)-XYPt(1) )/ds;
    YBP = ( BoundPts(:,2)-XYPt(2) )/ds;
```

```
% Distance to the boundary points.
% --------------------------------
SBP = ( XBP.^2 + YBP.^2 ).^0.5;

% Draw support and influence points
% ---------------------------------
if DrawSupport ==1
    figure;
    hold on;
    PlotGeo
    hold on;
    rectangle('Position',[XYPt(1)-ds,XYPt(2)-ds,2*ds,2*ds],...
              'Curvature',[1,1]);
    plot(XYPt(1),XYPt(2),'bo');
    plot(XYZGP(InflPoints,1),XYZGP(InflPoints,2),'r.');
    if NBP~=0
        plot(BoundPts(:,1),BoundPts(:,2),'kO');
    end
    hold off
    axis equal
    pause;
end
```

The weighting function has been taken as the fourth-order spline, commonly used in the MLS related literature:

$$W(\mathbf{x}-\chi) = 1 - 6s^2 + 8s^3 - 3s^4$$

```
% Evaluate the Weighting Function
% -------------------------------
S2     =  S.*S;
W      = (1 - 6*S2 + 8*S2.*S - 3*S2.*S2);
W(W<0) =  0; % needed due to round-off errors


% Create PatchMatrices and impose constraint equations.
% -----------------------------------------------------
if IntEquil
    dummy    = (-12+24*S-12*S.^2);
    Diameter = ds;
    dWdx     = dummy.*X/Diameter^2;
    dWdy     = dummy.*Y/Diameter^2;
end
```

```
    WpFactor = 1;
```

The boundary equilibrium must be satisfied at each point along the contour and in order to obtain a continuous recovered stress field the *nearest point* approach is used. Being *NBP* the number of points on the boundary where the constraints are imposed (see MLStechnique.pdf) In that case, the weighting function is defined as:

$$W'(\mathbf{x}-x_j) = \frac{W(\mathbf{x}-x_j)}{s} = \frac{1}{s} - 6s + 8s^2 - 3s^3$$

```
    % Evaluate the Weighting Function in Boundaries
    % ---------------------------------------------
    if ContourEquilWprime && NBP > 0
        WBP = abs(1./SBP - 6*SBP + 8*SBP.^2 - 3*SBP.^3)*WpFactor;
        if IntEquil
            dummy  = (-1./SBP.^3 - 6./SBP + 16 - 9*SBP)*WpFactor;
            dWBPdx = dummy.*XBP/Diameter^2;
            dWBPdy = dummy.*YBP/Diameter^2;
        end
    end

    % Evaluate Degree of interpolation polynomial in patch.
    % -----------------------------------------------------

    % Remember that SubSizes = [3,6,10,15,21,28];
    Degree = MaxFieldDegree;
    while NSP<SubSizes(Degree)
        Degree = Degree-1;
    end

    if Degree < 1
        disp...
      ('Not enough sampling points. Please reconsider the
discretization');
    end

    SubMSize        = SubSizes(Degree);
    MinDegree(iGP) = Degree;
```

Prepare data for M and G matrices for internal and boundary equ.

Before getting through the imposition of the boundary and internal equilibrium equations, we prepare the equation system for **M** and **G**

```
% Prepare data for M and G matrices
% ---------------------------------
[pTerms]   = polininterp2d(X,Y,Degree);
RowOfOnes = ones(1,SubMSize);
TotalSize = SubMSize*Noc;
M          = zeros(TotalSize,TotalSize);

    % Internal equilibrium case.
    % **************************

if IntEquil;
    dMdx   = M;
    dMdy   = M;
end

        % Create original system of equations
        % ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

W1          =  W * RowOfOnes;
pTermsTr    =  pTerms';
pTermsTrW =  pTermsTr.*W1';
Wptp        =  pTermsTr*(pTerms.*W1);
G           = [pTermsTrW*FieldFEGP(InflPoints,1);...
               pTermsTrW*FieldFEGP(InflPoints,2);...
               pTermsTrW*FieldFEGP(InflPoints,3)];

        % Loop over field components
        % ~~~~~~~~~~~~~~~~~~~~~~~~~~

for iNoc=1:Noc
    Index   = SubMSize*(iNoc-1)+1;
    M(Index:Index+SubMSize-1,Index:Index+SubMSize-1)...
           = Wptp;
end
```

```
    if IntEquil
        dWdx1         =  dWdx * RowOfOnes;
        dWdy1         =  dWdy * RowOfOnes;
        pTermsTrdWdx =  pTermsTr.*dWdx1';
        pTermsTrdWdy =  pTermsTr.*dWdy1';
        dWdxptp       =  pTermsTr*(pTerms.*dWdx1);
        dWdyptp       =  pTermsTr*(pTerms.*dWdy1);
        dGdx          = [pTermsTrdWdx*FieldFEGP(InflPoints,1);...
                         pTermsTrdWdx*FieldFEGP(InflPoints,2);...
                         pTermsTrdWdx*FieldFEGP(InflPoints,3)];
        dGdy          = [pTermsTrdWdy*FieldFEGP(InflPoints,1);...
                         pTermsTrdWdy*FieldFEGP(InflPoints,2);...
                         pTermsTrdWdy*FieldFEGP(InflPoints,3)];

        for iNoc=1:Noc
            Index = SubMSize*(iNoc-1)+1;
            dMdx(Index:Index+SubMSize-1,Index:Index+SubMSize-1)...
                  = dWdxptp;
            dMdy(Index:Index+SubMSize-1,Index:Index+SubMSize-1)...
                  = dWdyptp;
        end
    end
```

Boundary and internal equilibrium enforcement.

Hereinafter the part of the code showing the process to add terms to **M** and **G** in both; boundary and internal equilibrium. We have along the support $\Omega_{\mathbf{x}}\chi$:

$$\mathbf{M}(\mathbf{x}) = \int_{\Omega_{\mathbf{x}}} W(\mathbf{x}-\chi)\mathbf{P}^T(\chi)\mathbf{P}(\chi)d\chi$$

$$\mathbf{G}(\mathbf{x}) = \int_{\Omega_{\mathbf{x}}} W(\mathbf{x}-\chi)\mathbf{P}^T(\chi)\sigma^h(\chi)d\chi$$

Assuming *n* sampling points of coordinates χ_l we would numerically have:

$$\mathbf{M} = \sum_{l=1}^{n} W(\mathbf{x}-\chi_l)\mathbf{P}^T(\chi_l)\mathbf{P}(\chi_l)|J(\chi_l)|H_l$$

$$\mathbf{G} = \sum_{l=1}^{n} W(\mathbf{x}-\chi_l)\mathbf{P}^T(\chi_l)\sigma^h(\chi_l)|J(\chi_l)|H_l$$

```
    % Adding constraints
    % ------------------

        % Add terms related to boundary conditions.
        % *****************************************

    if ContourEquilWprime && NBP > 0
        for i=1:NBP
```

Along the boundaries, the stress vector $\sigma^*(x,\chi)$ in the local coordinate system $\widetilde{xy}$ is expressed :

$$\widetilde{\sigma}^*(\mathbf{x},\chi) = r(\alpha)\sigma^*(\mathbf{x},\chi)$$

where **r** is the stress rotation matrix:

$$r = \begin{bmatrix} r_{\widetilde{xx}} \\ r_{\widetilde{yy}} \\ r_{\widetilde{xy}} \end{bmatrix} = \begin{bmatrix} \cos^2\alpha & \sin^2\alpha & \sin(2\alpha) \\ sin^2\alpha & \cos^2\alpha & -\sin(2\alpha) \\ -\sin(2\alpha)/2 & \sin(2\alpha)/2 & \cos(2\alpha) \end{bmatrix}$$

```
            Ang          = AngleBoundPts(i);
            pBP          = polininterp2d...
                           (XBP(i),YBP(i),Degree,1);
            LocaDistance = sqrt(XBP(i)^2 + YBP(i)^2);

            if LocaDistance > TolBundary
                Lagrange = 0; % If far, normal way
            else
                Lagrange = 1; % If close, Lagrange multipliers
            end
            switch NodeType(i)
                case BoundImpTract

                   % Evaluate stress at boundary point(Normal & Tangential)
                   % ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                     [~,SIG]=press(BoundNum(i),tBoundPts(i),...
                         BoundPts(i,1),BoundPts(i,2),AngleBoundPts(i));

                     % Prepare the Rotation matrices
                     % ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                     [~,RStress] = RotationMatrices(Ang,2);
```

```
            PTotalNode  = zeros(Noc,TotalSize);
            for iNoc=1:Noc
                Index = SubMSize*(iNoc-1)+1;
                PTotalNode(iNoc,Index:Index+SubMSize-1)...
                      = pBP;
            end


            % Sxx and Sxy stress at contour
            % ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
            RPxx          = RStress(1,:)*PTotalNode;
            RPxy          = RStress(3,:)*PTotalNode;
            PtRtRP        = RPxx'*RPxx   + RPxy'*RPxy  ;
            PtRtSigt      = RPxx'*SIG(1) + RPxy'*SIG(2);

        case Symmetry
            % Evaluate stresses at symmetric side
            % ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
            [~,RStress]   = RotationMatrices(Ang,2);
            PTotalNode    = zeros(Noc,TotalSize);
            for iNoc=1:Noc
                Index     = SubMSize*(iNoc-1)+1;
                PTotalNode(iNoc,Index:Index+SubMSize-1)...
                          = pBP;
            end
            % Sxy stress at contour = 0
            RPxy          = RStress*PTotalNode;
            RPxy          = RPxy(3,:);
            PtRtRP        = RPxy'*RPxy;

        otherwise   % Change this for other types of nodes
            PtRtRP        = sparse(size(M,1),size(M,1));
            PtRtSigt      = sparse(size(G,1),1);

    end
```

For *NBP* points in the boundary, the **M** and **G** are in this case:

$$\mathbf{M} = \sum_{l=1}^{n} W(\mathbf{x} - \chi_l)\mathbf{P}^T(\chi_l)\mathbf{P}(\chi_l)|J(\chi_l)|H_l + \sum_{j=1}^{nbc} W'(\mathbf{x} - \chi_j)\mathbf{P}^T(\chi_j) r_{\tilde{i}}^T r_{\tilde{i}} \mathbf{P}(\chi_j)$$

$$\mathbf{G} = \sum_{l=1}^{n} W(\mathbf{x} - \chi_l)\mathbf{P}^T(\chi_l)\sigma^h(\chi_l)|J(\chi_l)|H_l + \sum_{j=1}^{nbc} W'(\mathbf{x} - \chi_j)\mathbf{P}^T(\chi_j) r_{\tilde{i}}^T \sigma'^{ez}_{\tilde{i}}(\chi_j)$$

```
        % Adding equations to M an G matrices
        % ************************************
             % Add equations to M an G
             % ~~~~~~~~~~~~~~~~~~~~~~~

            if ~Lagrange
                M = M + WBP(i)*PtRtRP;
                if NodeType(i)~=Symmetry % if NodeType==Symmetry => Sig=Tau=0
                    G = G + WBP(i)*PtRtSigt;
                end
                if IntEquil
                    dMdx = dMdx + dWBPdx(i)*PtRtRP;
                    dMdy = dMdy + dWBPdy(i)*PtRtRP;
                    if NodeType~=Symmetry % if NodeType==Symmetry =>Sig=Tau=0
                        dGdx = dGdx + dWBPdx(i)*PtRtSigt;
                        dGdy = dGdy + dWBPdy(i)*PtRtSigt;
                    end
                end
            else
                if  NodeType(i)~=Symmetry
                    Madd = [Madd ; RPxx ; RPxy];
                    Cadd = [Cadd ; SIG(1) ; SIG(2)];
                else
                    Madd = [Madd ; RPxy];
                    Cadd = [Cadd ; 0];
                end
                break
            end
        end
    end
```

The internal equilibrium equation is satisfied using the Lagrange multipliers technique. (see MLStechnique.pdf)
The constrain equations can be evaluated from the derivatives of matrices **M** and **G**:

$$\frac{\partial \mathbf{M}}{\partial \mathbf{x}} = \sum_{l=1}^{n} \frac{\partial W(\mathbf{x}-\chi_l)}{\partial \mathbf{x}} \mathbf{P}^T(\chi_l)\mathbf{P}(\chi_l)|J(\chi_l)|H_l + \sum_{j=1}^{nbc} \frac{\partial W'(\mathbf{x}-\chi_j)}{\partial \mathbf{x}} \mathbf{P}^T(\chi_j) r_{\tilde{i}}^T r_{\tilde{i}} \mathbf{P}(\chi_j)$$

$$\frac{\partial \mathbf{G}}{\partial \mathbf{x}} = \sum_{l=1}^{n} \frac{\partial W(\mathbf{x}-\chi_l)}{\partial \mathbf{x}} \mathbf{P}^T(\chi_l)\sigma^{h}(\chi_l)|J(\chi_l)|H_l + \sum_{j=1}^{nbc} \frac{\partial W'(\mathbf{x}-\chi_j)}{\partial \mathbf{x}} \mathbf{P}^T(\chi_j) r_{\tilde{i}}^T \sigma_{\tilde{i}}^{ez}(\chi_j)$$

```
        % Add internal equilibrium constraint equations which
        % satisfy contour equilibrium.
        % ***************************************************
    if IntEquil

        % MAIN EQUATIONS (related to x-direction derivatives)

            % pTerms and its partial derivatives
            % ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        [p0,dp0dx,dp0dy]=polininterp2d(0,0,Degree,1);

        Minv  = inv(M);
        P0    = zeros(Noc,TotalSize);
        dP0dx = P0;
        dP0dy = P0;
        for iNoc=1:Noc
            Index                             = SubMSize*(iNoc-1)+1;
            P0   (iNoc,Index:Index+SubMSize-1) = p0;
            dP0dx(iNoc,Index:Index+SubMSize-1) = dp0dx;
            dP0dy(iNoc,Index:Index+SubMSize-1) = dp0dy;
        end
        T1x      =  dP0dx - FIE*P0*Minv*dMdx;
        T2x      =  FIE*P0*Minv*dGdx;
        T1y      =  dP0dy - FIE*P0*Minv*dMdy;
        T2y      =  FIE*P0*Minv*dGdy;
        MConstr  = [T1x(1,:)+T1y(3,:);...
                    T1y(2,:)+T1x(3,:)];

            % Evaluate body forces
            % ~~~~~~~~~~~~~~~~~~~~
        [b]      =  VolLoads(XYPt,[],[]);
```

```
        GConstr = [-T2x(1,:)-T2y(3,:)-b(1);...
                   -T2y(2,:)-T2x(3,:)-b(2)];
        M       = [ M       , MConstr';...
                    MConstr , zeros(size(MConstr,1),size(MConstr,1))];
        G       = [ G; GConstr ];
    end
```

Obtaining the vector of coeficient A.

The use of Lagrange Multipliers leads to the following system of equations:

$$\begin{bmatrix} M & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} A \\ \lambda \end{bmatrix} = \begin{bmatrix} G \\ D \end{bmatrix}$$

The resolution of that system is processed by blocks. (further details could be found in the MLStechnique.pdf.

```
    % Solving The System of Equation
    % ------------------------------
    CondNum(iGP) = condest(M);

        % If boundary conditions with Lagrange Multipliers
        % *************************************************
    if (ContourEquilWprime && NBP > 0)  && Lagrange
        if size(Madd,2) < size(M,2)
            Madd(size(Madd,1),size(M,2)) = 0;
        end
        M = [M Madd';Madd sparse(size(Madd,1),size(Madd,1))];
        G = [G; Cadd];
    end

    Coefi             = M\G;
    FieldStGP(iGP,:) = Coefi(1:SubMSize:((Noc-1)*SubMSize+1))';
end
disp(['MinDegree = ' num2str(min(MinDegree))]);

disp('Smoothing technique finished. All data saved');
```

8.5.Press.m

This subroutine is used to define the pressure to be applied on the boundaries. This subroutine must be adapted by the user in each of the problems analyzed.

```
%***********************************************************************
% This file is part of the MLS-C© Library; you can redistribute it and/or
% modify it under the terms of the GNU Lesser General Public License as
% published by the Free Software Foundation; either version 3 of the
% License,or any later version.
% This program is distributed in the hope that it will be useful but
% WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General
% Public License for more details.
% You should have received a copy of the GNU Lesser General Public License
% along with GNU; see the file COPYING.  If not, write to the Free Software
% Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA
% or see <http://www.gnu.org/licenses/>.
%***********************************************************************
% Created by:
%    E Nadal, M Ouallal, JJ Rodenas, F Chinesta and FJ Fuenmayor
%
% Release 1.0
% Date: 18/07/2014
%***********************************************************************
function [txty,tntt]=press(Curve,t,x,y,Angle)


% INPUT DATA:
%------------------
% Curve - Id. Curve (internal ID)
% t     - relative position of the point within the Curve
% x & y - x and y global coordinates of the point
% Angle - Angle of the normal to the surface at points
%
% OUTPUT DATA:
%------------------
% txty  - will be a 2x1 vector where:
%           txty(1) = tx
%           txty(2) = ty
%
```

```
%**************************************************************

GlobalCS=1;   %the tractions will be specified as {tx,ty}
LocalCS=0;    %the tractions will be specified as {tn,tt}
              %note: use t>0 for tractions and t<0 for compressions


switch Curve
    %@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
    % The user can make changes to define the pressure on boundary,
    % from here...

    % Different examples are given to define the pressure for each curve.

    case 1
        CS=LocalCS;
        tn=-100;
        tt=   0;

%    case 2
%         CS=GlobalCS;
%         tx=-100;
%         ty=   0;

%     case 3
%         CS=LocalCS;
%         tn=-100*t;
%         tt= 0*t;

%     case 3
%         CS=LocalCS;
%         tn=-100*y;
%         tt= 0*t;


        % to here.
    %@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
    otherwise
        CS=GlobalCS;
        tx=0;
        ty=0;
end
% Change the coordinate system to global
```

```
if CS==LocalCS
    tntt=[tn;tt];
    txty =[cos(-Angle),sin(-Angle);-sin(-Angle),cos(-Angle)]*tntt;
elseif CS==GlobalCS
    [txty]=[tx;ty];
    tntt =[cos(Angle),sin(Angle);-sin(Angle),cos(Angle)]*txty;
end
```

8.6.VolLoads.m

This subroutine is used to define the voulume loada pressure to be applied. This subroutine must be adapted by the user in each of the problems analyzed.

```
%**********************************************************************
% This file is part of the MLS-C© Library; you can redistribute it and/or
% modify it under the terms of the GNU Lesser General Public License as
% published by the Free Software Foundation; either version 3 of the
% License,or any later version.
% This program is distributed in the hope that it will be useful but
% WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General
% Public License for more details.
% You should have received a copy of the GNU Lesser General Public License
% along with GNU; see the file COPYING.  If not, write to the Free Software
% Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA
% or see <http://www.gnu.org/licenses/>.
%**********************************************************************
% Created by:
%    E Nadal, M Ouallal, JJ Rodenas, F Chinesta and FJ Fuenmayor
%
% Release 1.0
% Date: 18/07/2014
%**********************************************************************
function [b]=VolLoads(XY,E,nu)
%=============================================
% Returns Loads per unit volume applied at XY
%=============================================

if size(XY,1)~=2
    XY = XY';
end

NumOfPoints = size(XY,2);
b           = [];
x           = XY(1,:);
y           = XY(2,:);
b           = [0;0]*ones(1,NumOfPoints);
```

8.7.CreateMacroToGetMLSdata.m

This subroutine is used to create the file MLSresults.mac that will be used to load the results in the Ansys database.

```
%**********************************************************************
% This file is part of the MLS-C© Library; you can redistribute it and/or
% modify it under the terms of the GNU Lesser General Public License as
% published by the Free Software Foundation; either version 3 of the
% License,or any later version.
% This program is distributed in the hope that it will be useful but
% WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General
% Public License for more details.
% You should have received a copy of the GNU Lesser General Public License
% along with GNU; see the file COPYING.  If not, write to the Free Software
% Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA
% or see <http://www.gnu.org/licenses/>.
%**********************************************************************
% Created by:
%    E Nadal, M Ouallal, JJ Rodenas, F Chinesta and FJ Fuenmayor
%
% Release 1.0
% Date: 18/07/2014
%**********************************************************************

function
CreateMacroToGetMLSdata(ShowUX,ShowVY,OutputXYZ,OutputTopology,ShowSX,Show
SY,ShowSZ,ShowTXY,ShowTXZ,ShowTYZ)
% THIS FUNCTION IS USED TO CREATE THE ANSYS MACRO CONTAINING THE RECOVERED
% STRESS FIELD OBTAINED BY THE MLSCX TECHNIQUE.
% INPUT VARIABLES
%=====================================================================

%   ShowUX,ShowVY,ShowWZ      NODE DISPLACEMENTS IN U, V and W DIRECTION.
%   OutputXYZ                 NODE COORDINATES.
%   OutputTopology            MESH TOPOLOGY.
%   ShowSX,ShowSY,ShowSZ,...
%   ShowTXY,ShowTXZ,ShowTYZ   STRESS COMPONENTS.


    fid = fopen('MLSresults.mac','wt');     % 'wt' means "write text"
```

```
    if (fid < 0)
        error('could not open file "MLSresults.mac"');
    end

    formatSpec  ='/PREP7   \n';
    fprintf(fid, formatSpec);
    formatSpec  ='ET,1,PLANE182 \n';
    fprintf(fid, formatSpec);

    % List of nodes coordinates
    formatSpec  ='N , %4u , %4.16f, %4.16f, %4.16f  \n';
    for i=1:size(OutputXYZ,1)
        fprintf(fid, formatSpec,i,OutputXYZ(i,1),OutputXYZ(i,2)...
                ,OutputXYZ(i,3));
    end

    % List for nodes per element
    formatSpec  ='EN , %4u , %4u , %4u , %4u , %4u   \n';
    for i=1:size(OutputTopology,1)

fprintf(fid,formatSpec,i,OutputTopology(i,1),OutputTopology(i,2)...
                ,OutputTopology(i,3),OutputTopology(i,4));
    end

    % Data introduced to postprocess
    formatSpec  ='MP,EX,1,1   \n';
    fprintf(fid, formatSpec);

    formatSpec  ='MP,NUXY,1,0.3   \n';
    fprintf(fid, formatSpec);

    formatSpec  ='D,ALL,ALL   \n';
    fprintf(fid, formatSpec);

    formatSpec  ='SAVE  \n';
    fprintf(fid, formatSpec);

    formatSpec  ='FINI   \n';
    fprintf(fid, formatSpec);

    formatSpec  ='/SOLUTION   \n';
    fprintf(fid, formatSpec);
```

```
formatSpec  ='SOLVE   \n';
fprintf(fid, formatSpec);

formatSpec  ='FINISH   \n';
fprintf(fid, formatSpec);

formatSpec  ='/POST1   \n';
fprintf(fid, formatSpec);


% List of dispacements
formatSpec  ='DNSOL , %4u, U, X, %4u   \n';
for i=1:numel(ShowUX)
    fprintf(fid, formatSpec,i,ShowUX(i));
end
formatSpec  ='DNSOL , %4u, U, Y, %4u   \n';
for i=1:numel(ShowVY)
    fprintf(fid, formatSpec,i,ShowVY(i));
end

% List of stresses
formatSpec   ='DESOL , %4u, %4u, S, X, %4u   \n';
for i=1:size(ShowSX,1)
    for j=1:size(ShowSX,2)
        fprintf(fid, formatSpec,i,OutputTopology(i,j),ShowSX(i,j));
    end
end
formatSpec   ='DESOL , %4u, %4u, S, Y, %4u   \n';
for i=1:size(ShowSY,1)
    for j=1:size(ShowSX,2)
        fprintf(fid, formatSpec,i,OutputTopology(i,j),ShowSY(i,j));
    end
end
formatSpec   ='DESOL , %4u, %4u , S, Z, %4u \n';
for i=1:size(ShowSZ,1)
    for j=1:size(ShowSZ,2)
        fprintf(fid, formatSpec,i,OutputTopology(i,j),ShowSZ(i,j));
    end
end
formatSpec   ='DESOL , %4u, %4u , S, XY, %4u   \n';
for i=1:size(ShowTXY,1)
    for j=1:size(ShowTXY,2)
        fprintf(fid, formatSpec,i,OutputTopology(i,j),ShowTXY(i,j));
```

```
        end
    end
    formatSpec   ='DESOL , %4u, %4u , S, XZ, %4u   \n';
    for i=1:size(ShowTXZ,1)
        for j=1:size(ShowTXZ,2)
            fprintf(fid, formatSpec,i,OutputTopology(i,j),ShowTXZ(i,j));
        end
    end
    formatSpec   ='DESOL , %4u, %4u , S, YZ, %4u  \n';
    for i=1:size(ShowTYZ,1)
        for j=1:size(ShowTYZ,2)
            fprintf(fid, formatSpec,i,OutputTopology(i,j),ShowTYZ(i,j));
        end
    end
    formatSpec   ='/GRAPHICS,FULL    \n';
    fprintf(fid, formatSpec);

    fclose(fid);
end
```

8.8.Auxiliary functions

This is a set of auxiliary functions required by the previous subroutines.

Pgauss.m

This subroutines is used to obtain the local coordinates of the Gauss integration points

```
%***********************************************************************
% This file is part of the MLS-C© Library; you can redistribute it and/or
% modify it under the terms of the GNU Lesser General Public License as
% published by the Free Software Foundation; either version 3 of the
% License,or any later version.
% This program is distributed in the hope that it will be useful but
% WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General
% Public License for more details.
% You should have received a copy of the GNU Lesser General Public License
% along with GNU; see the file COPYING.  If not, write to the Free Software
% Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA
% or see <http://www.gnu.org/licenses/>.
%***********************************************************************
% Created by:
%     E Nadal, M Ouallal, JJ Rodenas, F Chinesta and FJ Fuenmayor
%
% Release 1.0
% Date: 18/07/2014
%***********************************************************************
function [Coordinates,Weights]=...
                        pgauss(Dimensions,NumSideElem,NumPoints,PointClass)
% CALCULATION OF GAUSS POINTS COORDIANTES AND WEIGHTS FOR
% VERTEX TRIANGLES (0,0),(1,0),(0,1)
% AND VERTEX QUADRILATERAL (-1,-1),(-1,1), (1,1) (1,-1)
%
%
% INPUT
    % Dimensions   1 Countour integral / 2 Integral area
```

```
    % NumSideElem  Number of side per element
    % NumPoints    Number of total integration points per element
    % PointClass   Set of integration points which being taken.
    %              Used to diferentiate quadrature rules with
    %              equal number of integration points
%

% OUTPUT
    % Coordinates  Integration points coordinates in the local system.
    % Weights      Weigth of each inegration point.
%==========================================================================
=
if exist('PointClass')==0
    PointClass=1;
end

%CONTOUR INTEGRAL
%================
if Dimensions==1
   switch NumPoints

   % Polynomial exact integration of degree P=1
   %-------------------------------------------
   case 1
      Coordinates = 0;
      Weights     = 2;

   % Polynomial exact integration of degree P=3
   %-------------------------------------------
   case 2
      Coordinates = [...
                    -0.577350269189626;...
                     0.577350269189626];
      Weights     = [1,1]';

   % Polynomial exact integration of degree P=5
   %-------------------------------------------
   case 3
      Coordinates  =[...
                    -0.774596669241483;...
                     0;...
                     0.774596669241483];
      Weights      =[...
```

```
                       0.555555555555555;...
                       0.888888888888888;...
                       0.555555555555555];



     % Polynomial exact integration of degree P=7
     %--------------------------------------------
     case 4
        Coordinates    =[...
                      -0.861136311594953;...
                      -0.339981043584856;...
                       0.339981043584856;...
                       0.861136311594953];
        Weights        =[...
                       0.347854845137454;...
                       0.652145154862546;...
                       0.652145154862546;...
                       0.347854845137454];

     % Polynomial exact integration of degree P=9
     %--------------------------------------------
     case 5
        Coordinates     =[...
                       -0.906179845938664;...
                       -0.538469310105683;...
                        0;...
                        0.538469310105683;...
                        0.906179845938664];
        Weights         =[...
                        0.236926885056189;...
                        0.478628670499366;...
                        0.568888888888889;...
                        0.478628670499366;...
                        0.236926885056189];

     % Polynomial exact integration of degree P=11
     %---------------------------------------------
     case 6
        Coordinates     =[...
                       -0.932469514203152;...
                       -0.661209386466265;...
                       -0.238619186083197;...
```

```
                       +0.238619186083197;...
                       +0.661209386466265;...
                       +0.932469514203152];
        Weights       =[...
                        0.171324492379170;...
                        0.360761573048139;...
                        0.467913934572691;...
                        0.467913934572691;...
                        0.360761573048139;...
                        0.171324492379170;];
     otherwise
        disp (['Not implemented for a given dimension '...
              'NumPoints = ',num2str(NumPoints)]);
        error('Stopping');
     end

  %======================================================================
  % AREA INTEGRAL
  % =============
  else  %Dimensions = 2 :
     switch NumSideElem
     %-------------------------------------------------------------------
     case 3 %Area Integral for triangles
        switch NumPoints
        case 3
           if PointClass==1 %3 points inside the element
              Coordinates=[...
                    1/6,1/6;...
                    2/3,1/6;...
                    1/6,2/3];
              Weights=[...
                    1/6;...
                    1/6;...
                    1/6];
           else              % 3 points in midside.
              Coordinates=[...
                    0.5,0;...
                    0.5,0.5;...
                    0,0.5];
              Weights=[...
                    1/6;...
                    1/6;...
                    1/6];
```

```
        end
    case 4
        Coordinates=[...
              1/3,1/3;...
              1/5,1/5;...
              3/5,1/5;...
              1/5,3/5];
        Weights=[...
              -27/96;...
              25/96;...
              25/96;...
              25/96];
    case 6
        if PointClass==1 %6 points inside the elem.
            Coordinates=[...
                  0.091576213509771, 0.091576213509771;...
                  0.816847572980459, 0.091576213509771;...
                  0.091576213509771, 0.816847572980459;...
                  0.445948490915965, 0.445948490915965;...
                  0.108103018168070, 0.445948490915965;...
                  0.445948490915965, 0.108103018168070];
            Weights=[...
                  0.109951743655322/2.0;...
                  0.109951743655322/2.0;...
                  0.109951743655322/2.0;...
                  0.223381589678011/2.0;...
                  0.223381589678011/2.0;...
                  0.223381589678011/2.0];
        else                %6 points in midside.
            Coordinates=[...
                  1/6,1/6;...
                  2/3,1/3;...
                  1/6,2/3;...
                  0.5,0;...
                  0.5,0.5;...
                  0,0.5];
            Weights=[...
                  1/6;...
                  1/6;...
                  1/6;...
                  1/6;...
                  1/6;...
                  1/6];...
```

```
            end
        otherwise
            disp (['Not implemented for triangular area '...
                  'NumPoints = ',num2str(NumPoints)]);
            error('Stopping');
        end

    %------------------------------------------------------------------
    case 4 % Area integral for quadrilaterals.
        switch NumPoints
        case 1
            Coordinates=[0,0];
            Weights=4;

        case 4
            Coordinates=[...
                  -0.57735026918963, -0.57735026918963;...
                  +0.57735026918963, -0.57735026918963;...
                  -0.57735026918963, +0.57735026918963;...
                  +0.57735026918963  +0.57735026918963];
            Weights=[...
                  1;...
                  1;...
                  1;...
                  1];

        case 9
            Coordinates=[...
                  -0.77459666924148, -0.77459666924148;...
                  +0                , -0.77459666924148;...
                  +0.77459666924148, -0.77459666924148;...
                  -0.77459666924148, +0                ;...
                  +0                , +0                ;...
                  +0.77459666924148, +0                ;...
                  -0.77459666924148, +0.77459666924148;...
                  +0                , +0.77459666924148;...
                  +0.77459666924148, +0.77459666924148];
            Weights=[...
                  0.30864197530864;...
                  0.49382716049383;...
                  0.30864197530864;...
                  0.49382716049383;...
                  0.79012345679012;...
```

```
            0.49382716049383;...
            0.30864197530864;...
            0.49382716049383;...
            0.30864197530864];

    case 16
        Coordinates=[...
            -0.86113631159495, -0.86113631159495;...
            -0.33998104358486, -0.86113631159495;...
            +0.33998104358486, -0.86113631159495;...
            +0.86113631159495, -0.86113631159495;...
            -0.86113631159495, -0.33998104358486;...
            -0.33998104358486, -0.33998104358486;...
            +0.33998104358486, -0.33998104358486;...
            +0.86113631159495, -0.33998104358486;...
            -0.86113631159495, +0.33998104358486;...
            -0.33998104358486, +0.33998104358486;...
            +0.33998104358486, +0.33998104358486;...
            +0.86113631159495, +0.33998104358486;...
            -0.86113631159495, +0.86113631159495;...
            -0.33998104358486, +0.86113631159495;...
            +0.33998104358486, +0.86113631159495;...
            +0.86113631159495, +0.86113631159495];
        Weights=[...
            0.12100299328560;...
            0.22685185185185;...
            0.22685185185185;...
            0.12100299328560;...
            0.22685185185185;...
            0.42529330301069;...
            0.42529330301069;...
            0.22685185185185;...
            0.22685185185185;...
            0.42529330301069;...
            0.42529330301069;...
            0.22685185185185;...
            0.12100299328560;...
            0.22685185185185;...
            0.22685185185185;...
            0.12100299328560];


    case 25
```

```
Coordinates=[...
      -0.90617984593866,  -0.90617984593866;...
      -0.53846931010568,  -0.90617984593866;...
      +0                ,  -0.90617984593866;...
      +0.53846931010568,  -0.90617984593866;...
      +0.90617984593866,  -0.90617984593866;...
      -0.90617984593866,  -0.53846931010568;...
      -0.53846931010568,  -0.53846931010568;...
      +0                ,  -0.53846931010568;...
      +0.53846931010568,  -0.53846931010568;...
      +0.90617984593866,  -0.53846931010568;...
      -0.90617984593866,  +0                ;...
      -0.53846931010568,  +0                ;...
      +0                ,  +0                ;...
      +0.53846931010568,  +0                ;...
      +0.90617984593866,  +0                ;...
      -0.90617984593866,  +0.53846931010568;...
      -0.53846931010568,  +0.53846931010568;...
      +0                ,  +0.53846931010568;...
      +0.53846931010568,  +0.53846931010568;...
      +0.90617984593866,  +0.53846931010568;...
      -0.90617984593866,  +0.90617984593866;...
      -0.53846931010568,  +0.90617984593866;...
      +0                ,  +0.90617984593866;...
      +0.53846931010568,  +0.90617984593866;...
      +0.90617984593866,  +0.90617984593866];
Weights=[...
      0.05613434886243;...
      0.11340000000000;...
      0.13478507238752;...
      0.11340000000000;...
      0.05613434886243;...
      0.11340000000000;...
      0.22908540422399;...
      0.27228653255075;...
      0.22908540422399;...
      0.11340000000000;...
      0.13478507238752;...
      0.27228653255075;...
      0.32363456790123;...
      0.27228653255075;...
      0.13478507238752;...
      0.11340000000000;...
```

```
                0.22908540422399;...
                0.27228653255075;...
                0.22908540422399;...
                0.11340000000000;...
                0.05613434886243;...
                0.11340000000000;...
                0.13478507238752;...
                0.11340000000000;...
                0.05613434886243];

       otherwise
          disp (['Not implemented for quadrilateral area '...
                'NumPoints = ',num2str(NumPoints)]);
          error('Stopping');
       end
    %-------------------------------------------------------------------
    otherwise
       disp ([' A ',num2str(NumSideElem),...
             ' side element was specified which will not be used']);
       error('Stopping');

    end
end
```

polininterp2d.m

This subroutine creates a 2D polynomial expansion for a point of coordinates (*x*,*y*).

```
%***************************************************************************
% This file is part of the MLS-C© Library; you can redistribute it and/or
% modify it under the terms of the GNU Lesser General Public License as
% published by the Free Software Foundation; either version 3 of the
% License,or any later version.
% This program is distributed in the hope that it will be useful but
% WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General
% Public License for more details.
% You should have received a copy of the GNU Lesser General Public License
% along with GNU; see the file COPYING.  If not, write to the Free Software
% Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA
% or see <http://www.gnu.org/licenses/>.
%***************************************************************************
% Created by:
%     E Nadal, M Ouallal, JJ Rodenas, F Chinesta and FJ Fuenmayor
%
% Release 1.0
% Date: 18/07/2014
%***************************************************************************

function [P,dPdx,dPdy]=polininterp2d(X,Y,Degree,DerivFlag)
%INPUT
    % X and Y are global coordinates
    % Degree : the polinomial degree
    % DeriveFlag: Flag used to indicate if derivatives of the polynomial
    %             respect to the global coordinates system must be
    %             evaluated.
%
% OUTPUT
%
    % P : the polynomial
    % dPdx : Polynomial first derivative with respect to x component
    % dPdy : Polynomial first derivative with respect to y component
```

```
% Evaluate the matrix of vectors used as interpolation polinomial
if size(X,2)>size(X,1)
    X = X';
    Y = Y';
end
if nargin<4                 % if ~exist('DerivFlag','var')
    DerivFlag = 0;
end

C1 = ones (length(X),1);  %Column of Ones
C0 = zeros(length(X),1);  %Column of Zeros


% Calculate the polynomial for each degree and its derivatives.
switch Degree
    case 1 % first polynomial degree
        P = [...
             C1 , ...
             X  ,Y];
        if DerivFlag
            dPdx = [...
                    C0     , ...
                    C1     , C0];
            dPdy = [...
                    C0     , ...
                    C0     , C1];
        end
    case 2 % second polynomial degree
        P = [...
             C1   , ...
             X    , Y    ,...
             X.*X, X.*Y,  Y.*Y];
        if DerivFlag
            dPdx = [...
                    C0     , ...
                    C1     , C0       , ...
                    2*X    , Y        , C0];
            dPdy = [...
                    C0     , ...
                    C0     , C1       , ...
                    C0     , X        , 2*Y];
        end
    case 3 % third polynomial degree
```

```
        X2 = X.*X;  XY=X.*Y; Y2=Y.*Y;
        P  = [...
            C1      , ...
            X       , Y     ,...
            X2      , XY    , Y2     ,...
            X2.*X   , X2.*Y , X.*Y2  , Y.*Y2];
        if DerivFlag
            dPdx = [...
                    C0      , ...
                    C1      , C0       , ...
                    2*X     , Y        , C0         , ...
                    3*X2    , 2*X.*Y   , Y2         , C0];
            dPdy = [...
                    C0      , ...
                    C0      , C1       , ...
                    C0      , X        , 2*Y        , ...
                    C0      , X2       , 2*X.*Y     , 3*Y2];
        end
    case 4 % fourth polynomial degree
        X2 = X.*X;  XY=X.*Y;   Y2=Y.*Y;
        X3 = X2.*X; X2Y=X2.*Y; XY2=X.*Y2; Y3=Y.*Y2;
        P  = [...
              C1      , ...
              X       , Y     ,...
              X2      , XY    , Y2     ,...
              X3      , X2Y   , XY2    , Y3  ,...
              X3.*X   , X3.*Y , X2.*Y2 , X.*Y3   , Y3.*Y];
        if DerivFlag
            dPdx = [...
                    C0      , ...
                    C1      , C0       , ...
                    2*X     , Y        , C0         , ...
                    3*X2    , 2*X.*Y   , Y2         , C0       , ...
                    4*X3    , 3*X2.*Y  , 2*X.*Y2    , Y3       , C0    ];
            dPdy = [...
                    C0      , ...
                    C0      , C1       , ...
                    C0      , X        , 2*Y        , ...
                    C0      , X2       , 2*X.*Y     , 3*Y2     , ...
                    C0      , X3       , 2*X2.*Y    , 3*X.*Y2  , 4*Y3];
         end
    case 5 % fifth polynomial degree
        X2   = X.*X;
```

```
        XY   = X.*Y;
        Y2   = Y.*Y;
        X3   = X2.*X;
        X2Y  = X2.*Y;
        XY2  = X.*Y2;
        Y3   = Y.*Y2;
        X4   = X2.*X2;
        X3Y  = X3.*Y;
        X2Y2 = X2.*Y2;
        XY3  = X.*Y3;
        Y4   = Y3.*Y;
        P    = [...
                C1     , ...
                X      , Y    ,...
                X2     , XY   , Y2     ,...
                X3     , X2Y  , XY2    , Y3     ,...
                X4     , X3Y  , X2Y2   , XY3    , Y4     ,...
                X4.*X  , X4.*Y , X3.*Y2 , X2.*Y3 , X.*Y4  , Y4.*Y];
        if DerivFlag
            dPdx = [...
                    C0     , ...
                    C1     , C0       , ...
                    2*X    , Y        , C0       , ...
                    3*X2   , 2*X.*Y   , Y2       , C0       , ...
                    4*X3   , 3*X2.*Y  , 2*X.*Y2  , Y3       , C0    , ...
                    5*X4   , 4*X3.*Y  , 3*X2.*Y2 , 2*X.*Y3  , Y4    , C0 ];
            dPdy = [...
                    C0     , ...
                    C0     , C1       , ...
                    C0     , X        , 2*Y      , ...
                    C0     , X2       , 2*X.*Y   , 3*Y2     , ...
                    C0     , X3       , 2*X2.*Y  , 3*X.*Y2  ,4*Y3   , ...
                    C0     , X4       , 2*X3.*Y  , 3*X2.*Y2 ,4*X.*Y3 ,5*Y4];
         end
    otherwise % error displays no results for Degree > 5
        err(['Degree' num2str(Degree)...
            'not implemented in PatchMatricesUStar']);
end
```

Shape_f_2d.m

Obtains the shape functions for 2 dimensions.

```
%***************************************************************************
% This file is part of the MLS-C© Library; you can redistribute it and/or
% modify it under the terms of the GNU Lesser General Public License as
% published by the Free Software Foundation; either version 3 of the
% License,or any later version.
% This program is distributed in the hope that it will be useful but
% WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General
% Public License for more details.
% You should have received a copy of the GNU Lesser General Public License
% along with GNU; see the file COPYING.  If not, write to the Free Software
% Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA
% or see <http://www.gnu.org/licenses/>.
%***************************************************************************
% Created by:
%    E Nadal, M Ouallal, JJ Rodenas, F Chinesta and FJ Fuenmayor
%
% Release 1.0
% Date: 18/07/2014
%***************************************************************************
function [N,dNpsi,dNeta]=shape_f_2d(Coor,Degree,NumSideElem,CaldN)
% Shape Function and its derivatives in a point for triangles and
% quadrilaterals in local coordinates X=psi y Y=eta
%
% IMPORTANT: Node enumeration was made that way:
%             vertex node are first, then midside nodes.
%
% INPUT
    % Coor        = coordinates (X = psi e Y = eta)
    % Degree      = Element degree
    % NumSideElem = Number of side per elements (1, 3 & 4)
    % CaldN       = Derivatives of N
    %               0    ==> no derivative
    %               else ==> derivative takes place

% OUTPUT
    % N           = Shape function matrix. Each row corresponds to the
```

```
    %               shape function in a coordinate points (x,y)
    % dNpsi       = Shape function matrix derivatives with respect to psi
    % dNeta       = Shape function matrix derivatives with respect to eta

if nargin==3
    CaldN = 0;
end


X = Coor(:,1)';
Y = Coor(:,2)';

switch NumSideElem
```

TRIANGLES

```
case
 switch Degree
   case 1         % Linear elements
      N      = [...
               1-X-Y;...
               X;...
               Y];
      if CaldN==0
         dN = 0;
      else
         dN = [...
               -1+0.*X,-1+0.*X;...
               +1+0.*X, 0+0.*X;...
                0+0.*X, 1+0.*X];

      end

   case 2         % Quadratics elements
      N     =[...
            (1-X-Y).*(1-2*X-2*Y);...
            -X.*(1-2*X);...
            -Y.*(1-2*Y);...
             4*X.*(1-X-Y);...
             4*X.*Y;...
             4*Y.*(1-X-Y)];
      if CaldN==0
```

```
            dN = 0;
        else
            dN = [...
                  -3+4*X+4*Y , -3+4*X+4*Y;...
                  -1+4*X     ,  0.*X        ;...
                  +0.*X      , -1+4*Y    ;...
                  +4-8*X-4*Y , -4*X      ;...
                  +4*Y       ,  4*X      ;...
                  -4*Y       , +4-4*X-8*Y];
        end

         otherwise  % Superior degree triangles
        disp(['Polynomial degree ' num2str(Degree)...
              'for triangles not implemented']);
        error('Stop');
    end
```

QUADRILATERALS

```
case 4
    switch Degree
         case 1      % Linears
        N = [...
            (1-Y).*(1-X)/4;...
            (1-Y).*(1+X)/4;...
            (1+Y).*(1+X)/4;...
            (1+Y).*(1-X)/4];
        if CaldN==0
            dN = 0;
        else
            dN = [...
                 (-1+Y)/4, (-1+X)/4;...
                 (+1-Y)/4, (-1-X)/4;...
                 (+1+Y)/4, (+1+X)/4;...
                 (-1-Y)/4, (+1-X)/4];
        end

    case 2           % Quadratics
        N = [...
             -(1-Y).*(1-X).*(1+X+Y)/4 ;...
             -(1-Y).*(1+X).*(1-X+Y)/4 ;...
             -(1+Y).*(1+X).*(1-X-Y)/4 ;...
```

```
           -(1+Y).*(1-X).*(1+X-Y)/4 ;...
           +(1-X.^2).*(1-Y)/2       ;...
           +(1-Y.^2).*(1+X)/2       ;...
           +(1-X.^2).*(1+Y)/2       ;...
           +(1-Y.^2).*(1-X)/2       ];
      if CaldN==0
         dN = 0;
      else
         dN = [...
               -(-1+Y).*(Y+2*X)/4  , -(-1+X).*(X+2*Y)/4  ;...
               -(-1+Y).*(-Y+2*X)/4 , -(1+X).*(X-2*Y)/4   ;...
               +(Y+1).*(Y+2*X)/4   , +(1+X).*(X+2*Y)/4   ;...
               -(Y+1).*(Y-2*X)/4   , -(-1+X).*(-X+2*Y)/4 ;...
               -X.*(1-Y)           , +(-1+X.^2)/2        ;...
               +(1-Y.^2)/2         , -Y.*(1+X)           ;...
               -X.*(Y+1)           , +(1-X.^2)/2         ;...
               +(-1+Y.^2)/2        , -Y.*(1-X)           ];
      end

       otherwise  % Superior degree quadrilaterals
      disp([ num2str(Degree) 'Polynomial degree not programmed for
quadrilaterals']);
      error('Stopping');
   end
otherwise
   disp ([num2str(NumSideElem),...
         ' Side per element showed up. it will not be used ']);
   error('Stopping');
end
N  = N';
dN = dN';
if CaldN ~= 0
    dNpsi  = dN(1:size(Coor,1),:);
    dNeta  = dN(size(Coor,1)+1:end,:);
else
    dNpsi  = [];
    dNeta  = [];
end

return;
```

RotationMatrices.m

This subroutines is used to obtain the rotation matrix and for the system transformation.

```
%**************************************************************************
% This file is part of the MLS-C© Library; you can redistribute it and/or
% modify it under the terms of the GNU Lesser General Public License as
% published by the Free Software Foundation; either version 3 of the
% License,or any later version.
% This program is distributed in the hope that it will be useful but
% WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General
% Public License for more details.
% You should have received a copy of the GNU Lesser General Public License
% along with GNU; see the file COPYING.  If not, write to the Free Software
% Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA
% or see <http://www.gnu.org/licenses/>.
%**************************************************************************
% Created by:
%    E Nadal, M Ouallal, JJ Rodenas, F Chinesta and FJ Fuenmayor
%
% Release 1.0
% Date: 18/07/2014
%**************************************************************************
function [RCoord,RStress]=RotationMatrices(Alpha,Dims)
% Evaluates rotation matrices for Coordinates and Stresses (or Stains)
% so that if we calculate
%     Results = RMatrix*Data
% Then Results are expressed in a refference system rotated Alpha rads
% with respect to the coordinates system used to express Data
% Note: Alpha could be a vector. This could be used for rotations in 3-D
%
% INPUT
% Alpha     = The angle of rotation
% Dims      = Dimensions
%
% OUTPUT
% RCoord  = The coordinate rotation matrix
% RStress = The stress rotation matrix
```

```
if nargin<2
    Dims = 2;
end

switch Dims
    case 2 % 2-D
        CA  = cos(Alpha);
        SA  = sin(Alpha);
        S2A = sin(2*Alpha);

% Rotates points in the xy-Cartesian plane counter-clockwise
% through the angle Alpha about the origin of the Cartesian coordinate
system.
        RCoord  = [...
                   CA , SA;...
                  -SA, CA];

        RStress = [...
                   CA^2      , SA^2     ,  S2A         ;...
                   SA^2      , CA^2     , -S2A         ;...
                  -0.5*S2A , 0.5*S2A ,  cos(2*Alpha)];
        return
% Only available for 2-D
    otherwise
        disp (['Cannot create matrices to rotate in ',num2str(Dims),'-
D']);
        error('stop');

end
```

ClosestPointsOnBoundrs.m

Subroutine used to obtain the closest point to boundary within the support of the point.

```
%**********************************************************************
% This file is part of the MLS-C© Library; you can redistribute it and/or
% modify it under the terms of the GNU Lesser General Public License as
% published by the Free Software Foundation; either version 3 of the
% License,or any later version.
% This program is distributed in the hope that it will be useful but
% WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General
% Public License for more details.
% You should have received a copy of the GNU Lesser General Public License
% along with GNU; see the file COPYING.  If not, write to the Free Software
% Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA
% or see <http://www.gnu.org/licenses/>.
%**********************************************************************
% Created by:
%    E Nadal, M Ouallal, JJ Rodenas, F Chinesta and FJ Fuenmayor
%
% Release 1.0
% Date: 18/07/2014
%**********************************************************************

function [BoundPts,AngleBoundPts,tBoundPts,BoundNum,NodeType,NBP]=...
    ClosestPointsOnBoundrs(x,y,ds,GeoPoints,Boundaries)
% This function finds the closest points to (x,y) on each of the
% boundaries. Points are within the support of (x,y), i.e. at a distance
% dist2xy, dist2xy<=ds

BoundPts      = [];
AngleBoundPts = [];
BoundNum      = [];
NodeType      = [];
tBoundPts     = [];
dist2xy       = [];

xy = [x y];
```

```
for i=1:size(Boundaries,1)
    v1=GeoPoints(Boundaries(i,1),:);
    v2=GeoPoints(Boundaries(i,2),:);

    %find nearest point only if curve is of
    %type==40 (Newman boundary) or type=41 (symmetry boundary)
    if Boundaries(i,5)==40 || Boundaries(i,5)==41

        if Boundaries(i,3)==0 %Curve is of type LINE SEGMENT

            %Translate segment limits and point
            a  = xy-v1;
            b  = [0,0]; %v1-v1;
            c  = v2-v1;
            % Rotate coordinates of point and 2nd vertex of segment
            [theta,~]=cart2pol(c(1),c(2));
            M   =[cos(-theta),-sin(-theta);sin(-theta),cos(-theta)];
            xyPoint =M*a';
            xyV2    =M*c';

            %Check point possition with repect to segment
            if xyPoint(1)<=0 %point is on the left of the segment
                xyNearest= v1;
                t=0;
            elseif xyPoint(1)>=xyV2(1) %point is on the right of segment
                xyNearest= v2;
                t=1;
            else %point is above or below the segment
                xyNearest=[xyPoint(1) 0];
                t=(xyNearest(1)-0)/(xyV2(1)-0);
                M   =[cos(theta),-sin(theta);sin(theta),cos(theta)];
                xyNearest= (M*xyNearest'+v1')';
            end


            AngleBoundPts = [AngleBoundPts; theta-pi()/2];


        elseif Boundaries(i,3)>0 %Curve is of type ARC OF CIRCUMFERENCE
            %Translate segment limits and point
            vc = GeoPoints(Boundaries(i,3),:);
            %Translate coordinates system to center of circumf.
            PointCart  = xy-vc;
```

```
CenterCart = [0,0]; %vc-vc;
Point1Cart = v1-vc;
Point2Cart = v2-vc;
%Evaluate angle of each point
PointAng      =cart2pol(PointCart(1), PointCart(2) );
[Point1Ang,R] =cart2pol(Point1Cart(1),Point1Cart(2));
Point2Ang     =cart2pol(Point2Cart(1),Point2Cart(2));

ClockWise=Boundaries(i,4);
if ClockWise>=0; ClockWise=1;
else ClockWise=0;
end

%The general case (does not require any adjustment) is:
%       ((Point1Ang<Point2Ang) && ~ClockWise) ||...
%       ((Point2Ang<Point1Ang) &&  ClockWise)

if ((Point1Ang<Point2Ang) && ClockWise)
    Point2Ang=Point2Ang-2*pi();
elseif ((Point2Ang<Point1Ang) && ~ClockWise)
    Point2Ang=Point2Ang+2*pi();
end

minAng=min(Point1Ang,Point2Ang);
maxAng=max(Point1Ang,Point2Ang);
Points2Check=[PointAng-2*pi,PointAng,PointAng+2*pi];

%Check points possition with repect to segment
NearestAng=[]; AngDif=[];

for j=1:3
    P=Points2Check(j);

    if P <= minAng %point on the left of segment
        NearestAng(j) = minAng;
        AngDif(j)     = minAng-P;
    elseif P >= maxAng %point on the right of segment
        NearestAng(j) = maxAng;
        AngDif(j)     = P-maxAng;
    else %point is above or below the segment
        NearestAng(j) = P;
        AngDif(j)     = 0;
    end
```

```
            end
            [~,IndexOfMinAngleDif] =min(AngDif);
            NearestAng=NearestAng(IndexOfMinAngleDif);
            t=(NearestAng-minAng)/(maxAng-minAng);

            [xyNearest(1),xyNearest(2)]= pol2cart(NearestAng,R);
            xyNearest=xyNearest+vc;

            if Clockwise
                NearestAng=NearestAng-pi;
            end
            AngleBoundPts = ...
                [AngleBoundPts; NearestAng];

        end
    end

    dist2xy            = [dist2xy,norm([x y] - xyNearest)];
    BoundPts  = [BoundPts;  xyNearest      ];
    tBoundPts = [tBoundPts; t              ];
    BoundNum  = [BoundNum;  i              ];
    NodeType  = [NodeType;  Boundaries(i,5)];

end
```

Select only the points within the support ds

```
index         = find(dist2xy<=ds);
BoundPts      = BoundPts(index,:);
AngleBoundPts = AngleBoundPts(index);
tBoundPts     = tBoundPts(index);
BoundNum      = BoundNum(index);
NodeType      = NodeType(index);
NBP           = length(index);
```

Activate the following code if you want to plot the nearest points

```
%to a given point in each of the curves used to define the boundary
%{
```

```
PlotGeo
% plot(BoundPts(:,1),BoundPts(:,2),'r*');
SizeX=max(GeoPoints(:,1))-min(GeoPoints(:,1));
ArrowSize=SizeX*.1;
for i=1:size(BoundPts,1)
    text(BoundPts(i,1),BoundPts(i,2),num2str([BoundNum(i) tBoundPts(i)]));
    plot(x,y,'+r')
    plot(...
        [BoundPts(i,1),BoundPts(i,1)+ArrowSize*cos(AngleBoundPts(i))],...
        [BoundPts(i,2),BoundPts(i,2)+ArrowSize*sin(AngleBoundPts(i))],...
        'r-');
end
%}
```

Plot_arc.m

Plots an arc of circumference on the screen.

```
%**********************************************************************
% This file is part of the MLS-C© Library; you can redistribute it and/or
% modify it under the terms of the GNU Lesser General Public License as
% published by the Free Software Foundation; either version 3 of the
% License,or any later version.
% This program is distributed in the hope that it will be useful but
% WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General
% Public License for more details.
% You should have received a copy of the GNU Lesser General Public License
% along with GNU; see the file COPYING.  If not, write to the Free Software
% Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA
% or see <http://www.gnu.org/licenses/>.
%**********************************************************************
% Created by:
%    E Nadal, M Ouallal, JJ Rodenas, F Chinesta and FJ Fuenmayor
%
% Release 1.0
% Date: 18/07/2014
%**********************************************************************
```

```
function plot_arc(XY1,XY2,XYC,ClockWise)
```

This function is used to plot arcs of circumference given the starting and end points, the center of the circumference and the sense to go from the starting to the end point
XY1 = coordinates of starting point
XY2 = coordinates of ending point
XYC = coordinates of the center of the arc

```
nsegments = 50;

if ClockWise>=0; ClockWise=1;
else ClockWise=0;
end

Cart1=XY1-XYC;
Cart2=XY2-XYC;
%
[Th1,Rad1]=cart2pol(Cart1(1),Cart1(2));
[Th2,Rad2]=cart2pol(Cart2(1),Cart2(2));
if abs(Rad1-Rad2)>100*eps
    error (['Rad1 ~= Rad2']);
end

%The general case (does not require any adjustment) is:
% ((Th1<Th2) && ~ClockWise) || ((Th2<Th1) &&  ClockWise)

%Special cases
if ((Th1<Th2) && ClockWise)
    Th2=Th2-2*pi();
elseif ((Th2<Th1) && ~ClockWise)
    Th2=Th2+2*pi();
end

th    = Th1:(Th2-Th1)/nsegments:Th2;
xunit = Rad1 * cos(th) + XYC(1);
yunit = Rad1 * sin(th) + XYC(2);
plot(xunit, yunit);
end
```

PlotGeo.m

```
This script is implemented to plot the geometry we work on
%*********************************************************************
% This file is part of the MLS-C© Library; you can redistribute it and/or
% modify it under the terms of the GNU Lesser General Public License as
% published by the Free Software Foundation; either version 3 of the
% License,or any later version.
% This program is distributed in the hope that it will be useful but
% WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General
% Public License for more details.
% You should have received a copy of the GNU Lesser General Public License
% along with GNU; see the file COPYING.  If not, write to the Free Software
% Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA
% or see <http://www.gnu.org/licenses/>.
%*********************************************************************
% Created by:
%    E Nadal, M Ouallal, JJ Rodenas, F Chinesta and FJ Fuenmayor
%
% Release 1.0
% Date: 18/07/2014
%*********************************************************************
figure
hold on
axis equal

for i=1:size(Boundaries,1)
    if Boundaries(i,3)==0
        Points=Boundaries(i,1:2);
        XYPoints=GeoPoints(Points,:);
        line(XYPoints(:,1),XYPoints(:,2));
    elseif Boundaries(i,3)>=0
        Points=Boundaries(i,1:2);
        XYPoints=GeoPoints(Points,:);
        XYCenter=GeoPoints(Boundaries(i,3),:);
        plot_arc(XYPoints(1,:),XYPoints(2,:),...
            XYCenter,Boundaries(i,4));
    end
end
```