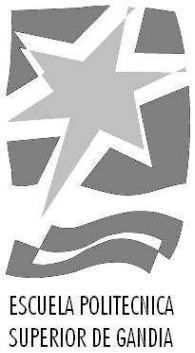
UNIVERSIDAD POLITECNICA DE VALENCIA

ESCUELA POLITECNICA SUPERIOR DE GANDIA

I.T. Telecomunicación (Sonido e Imagen)



“Desarrollo de una aplicación de secuenciado MIDI en MatLab”

TRABAJO FINAL DE CARRERA

Autor/es:
Francisco José Molina García

Director/es:
D. Jaime García Rupérez

GANDIA, 2010

ÍNDICE

1. Introducción

1.1 Introducción al PFC

La tecnología musical, en nuestras últimas décadas, ha crecido de forma espectacular y ha dado grandes cambios respecto a sus inicios. En los comienzos teníamos grandes estudios de grabación, repletos de hardware con un coste bastante elevado y sólo al alcance de muy pocos. No obstante, se han producido grandes cambios en esta evolución, substituyendo ese hardware por programas informáticos (software) de coste más moderado y accesibles a un mayor público. Con estos programas se pueden igualar o incluso superar los resultados que se obtenían y se obtienen mediante los sistemas hardware.

Gracias a esta evolución, una de las aplicaciones más desarrolladas, y utilizadas, son los secuenciadores de audio que son la herramienta principal de composición, programación y control sobre los equipos de instrumentación electrónica musical. Estos secuenciadores permiten programar y reproducir eventos musicales de forma secuencial mediante una interfaz de control físico o lógico conectado a uno o más instrumentos musicales electrónicos.

El interfaz de control más extendido es el estándar MIDI (Musical Instrument Digital Interface). Se trata de un protocolo de comunicación estándar que permite a los computadores, sintetizadores, secuenciadores, controladores y otros dispositivos musicales electrónicos comunicarse y compartir información para la generación de sonidos.

El objetivo del presente proyecto final de carrera es diseñar y desarrollar una aplicación en Matlab que permita el secuenciado musical, de forma similar al secuenciado utilizado en los ficheros de audio MIDI. Este tipo de secuenciado se basa en indicar en cada instante de tiempo qué sonido reproducir (qué instrumento, qué nota, qué duración,…), de forma que la aplicación desarrollada en Matlab se encargará de generar ese sonido mediante fórmulas matemáticas. Esto es lo que se conoce como síntesis de sonido. De esta forma, el usuario de la aplicación creada únicamente tendrá que ir indicando qué sonido reproducir en cada momento para crear de esta forma la composición musical final.

Como lenguaje de programación se ha escogido el MATLAB ya que es uno de los más extendidos dentro del ámbito de la ingeniería y además permite una rápida ampliación y adaptación según las preferencias propias de cada usuario. Al tratarse de aplicaciones gráficas, el usuario no necesita conocer MATLAB para poder hacer uso de ellas. Por otro lado, si lo conociera, podría modificar o ampliar él mismo las funciones que está ejecutando y profundizar en otras.

1.2 Objetivos

A continuación se presentan los objetivos principales a desarrollar en este proyecto:

- Análisis y modelado del sonido de instrumentos musicales. Se ha concretizado el análisis para el caso de una guitarra clásica.
- Síntesis del timbre característico de una guitarra mediante el método de síntesis aditiva (superposición de señales senoidales).
- Estudio de las diversas alternativas para conseguir la síntesis de sonidos de percusión con una buena fidelidad.
- Creación de una interfaz gráfica para facilitar la composición al usuario.

1.3 Estructura del proyecto

En cuanto a la estructura de desarrollo del proyecto se pueden diferenciar tres partes dependiendo del área de trabajo y las tareas a desempeñar.

Una primera parte dedicada al análisis espectral de cada una de las notas para poder sintetizarlas después. Se hará un análisis espectral mediante la Transformada de Fourier de unas notas de guitarra acústica previamente creadas mediante un software de audio (‘FL Studio 9’). El análisis espectral se realizará mediante MatLab.

En la segunda parte se trabajará con aquellos parámetros obtenidos de las notas analizadas en la primera parte. Se sintetizarán mediante algoritmos matemáticos todas aquellas notas que puede reproducir una guitarra acústica intentando que el sonido sintetizado sea lo más fiel al sonido real. También se sintetizará algún instrumento de percusión sencillo para que la composición final creada sea más completa.

En la última parte de este proyecto se creará mediante la aplicación Guide de MatLab una interfaz gráfica para que la composición de las piezas musicales sea más sencilla y amena.

Por lo tanto para realizar la implementación deberemos hacer empleo de distintas herramientas y conocimientos, todos relacionados con el mundo de la ingeniería y de la tecnología musical.

2. Teoría del sonido natural y sintetizado

Para poder hacer la síntesis del sonido, primero se tiene que estudiar qué es, qué características tiene e investigar demás parámetros que lo definen.

Se puede definir el sonido como una sensación auditiva que está producida por la vibración de algún objeto. Estas vibraciones se propagan a través de un medio elástico como el aire o el agua y son captadas por nuestro oído y transformadas en impulsos nerviosos que se mandan a nuestro cerebro.

En el estudio del sonido se deben distinguir los aspectos físicos de los fisiológicos relacionados con la audición. Desde el punto de vista físico el sonido comparte todas las propiedades características del movimiento ondulatorio, por lo que puede ser descrito utilizando los conceptos sobre las ondas. Desde el punto de vista fisiológico solo existe sonido cuando el oído es capaz de percibirlo. Dicho de otra forma, la descripción del sonido es en parte objetiva y en parte subjetiva. Cada persona escucha un mismo sonido de una forma distinta.

2.1 El sonido como movimiento ondulatorio (M.A.S)

Solemos decir que el sonido de una determinada nota musical se representa gráficamente por la función seno. Esta representa un movimiento vibratorio llamado movimiento armónico simple, que es aquel que se obtiene cuando los desplazamientos del cuerpo vibrante son directamente proporcionales a las fuerzas causantes de este desplazamiento.

El movimiento armónico simple es un movimiento periódico que queda descrito en función del tiempo por una función armónica (seno o coseno). Si la descripción de un movimiento requiriese más de una función armónica, en general sería un movimiento armónico, pero no un M.A.S. Esto ocurre con el sonido que, como cualquier clase de movimiento ondulatorio, puede considerarse como la superposición de movimientos armónicos simples.

Combinando la ley de Hooke y la 2ª ley de Newton se obtiene la ecuación que define el M.A.S.

$$\left.\begin{aligned} F_x &= -k\,x \\ F_x &= m\,a_x \end{aligned}\right\} \; m\,a_x = m\frac{d^2x}{dt^2} = -k\,x$$

De donde se llega a su ecuación generalizada:

$$x = A\ sen(2\pi\ f\ t + \varphi') = A\ sen(\omega\, t + \varphi')$$

donde:

- x es la elongación de la partícula.
- A es la amplitud del movimiento (elongación máxima).
- ω es la frecuencia angular ($2\pi\ f$)
- t es el tiempo.
- φ es la fase inicial e indica el estado de oscilación o vibración (o fase) en el instante t = 0 de la partícula que oscila.

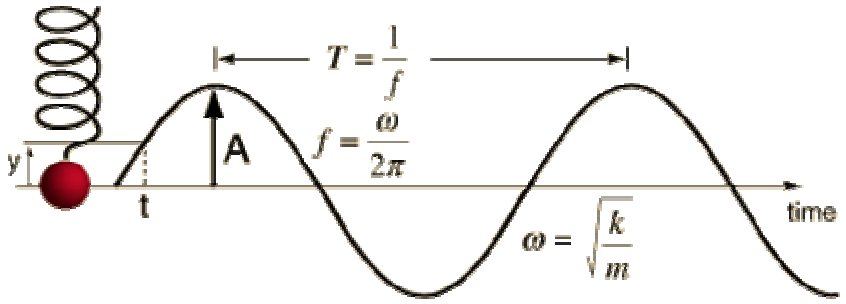


Fig. 2.1.2. Representación del M.A.S comparándolo con muelle

Como se ha comentado anteriormente un sonido cualquiera es una combinación de ondas sonoras que difieren en los cinco parámetros anteriores. La caracterización de un sonido arbitrariamente complejo implica analizar tanto la energía transmitida como la distribución de dicha energía entre las diversas ondas componentes, para ello resulta útil investigar:

- Potencia acústica: El nivel de potencia acústica es la cantidad de energía radiada en forma de ondas por unidad de tiempo por una fuente determinada. La potencia acústica depende de la amplitud.
- Espectro de frecuencias: que permite conocer en qué frecuencias se transmite la mayor parte de la energía.

2.2 Cualidades y parámetros del sonido natural

El oído es capaz de distinguir unos sonidos de otros porque es sensible a las diferencias que puedan existir entre ellos en lo que concierne a alguna de las tres cualidades que caracterizan todo sonido y que son la intensidad o sonoridad, el tono y el timbre. Todas estas cualidades se refieren al sonido fisiológico pero están relacionadas con diferentes propiedades físicas de las ondas sonoras. Esta relación se indica en la tabla siguiente:

Efecto sensorial	**Propiedad física de la onda**
Sonoridad	Intensidad de la onda
Tono	Frecuencia de la onda
Timbre	Forma de la onda

2.2.1 Intensidad

La intensidad del sonido percibido, o propiedad que hace que éste se capte como fuerte o como débil, está relacionada con la intensidad de la onda sonora correspondiente, también llamada intensidad acústica. La intensidad acústica es una magnitud que da idea de la cantidad de energía que está fluyendo por el medio como consecuencia de la propagación de la onda.

Se define como la energía que atraviesa por segundo una superficie unidad dispuesta perpendicularmente a la dirección de propagación. Equivale a una potencia por unidad de superficie y se expresa en W/m^2. La intensidad de una onda sonora es proporcional al cuadrado de su frecuencia y al cuadrado de su amplitud y disminuye con la distancia al foco.

La magnitud de la sensación sonora depende de la intensidad acústica, pero también depende de la sensibilidad del oído. El intervalo de intensidades acústicas que va desde el umbral de audibilidad, o valor mínimo perceptible, hasta el umbral del dolor.

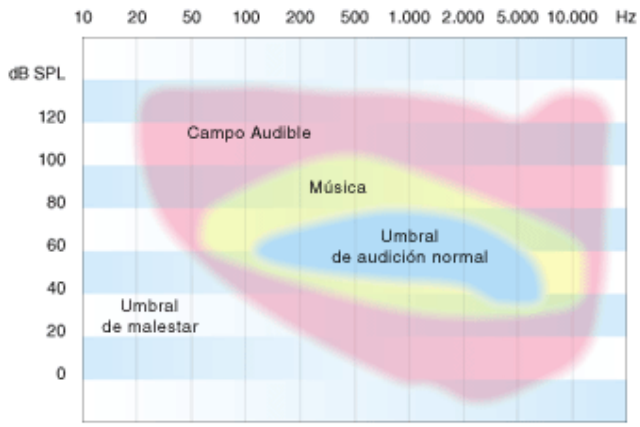


Fig. 2.2.1. 1. Representación de los rangos de isofonía

Para expresar la intensidad fisiológica o sensación sonora de un sonido se emplea una escala cuyas divisiones son potencias de diez y cuya unidad de medida es el decibelio (dB). Por ejemplo, el umbral de la audición está en 0 dB, la intensidad fisiológica de un susurro corresponde a unos 10 dB y el ruido de las olas en la costa a unos 40 dB.

La conversión entre intensidad y decibelios viene dada por la ley de Weber-Fechnes. Esta ley asegura que, "la sensación sonora de intensidad es aproximadamente igual al logaritmo de la energía que produce la excitación". Si escribimos esta ley con una formula la expresión es la siguiente:

$$S = 10\log\left(\frac{I}{I_0}\right)$$

donde $I_0 = 10^{-12}$ W/m^2 y corresponde a un nivel de 0 decibelios por tanto. El umbral del dolor corresponde a una intensidad de 1 W/m^2 o 120 dB.

Al ser una escala logarítmica significa que una intensidad acústica de 10 decibelios corresponde a una energía diez veces mayor que una intensidad de cero decibelios; una intensidad de 20 dB representa una energía 100 veces mayor que la que corresponde a 0 decibelios y así sucesivamente. Por ejemplo, el ruido de las olas en la costa es 1.000 veces más intenso que un susurro, lo que equivale a un aumento de 30 dB.

No obstante hay que aclarar que esta ley es aplicable a la zona central del campo de audibilidad pero no a cualquier frecuencia. Los físicos Fletcher y Munson dedujeron experimentalmente en 1933 las líneas de igual nivel auditivo (curvas de isofonía).

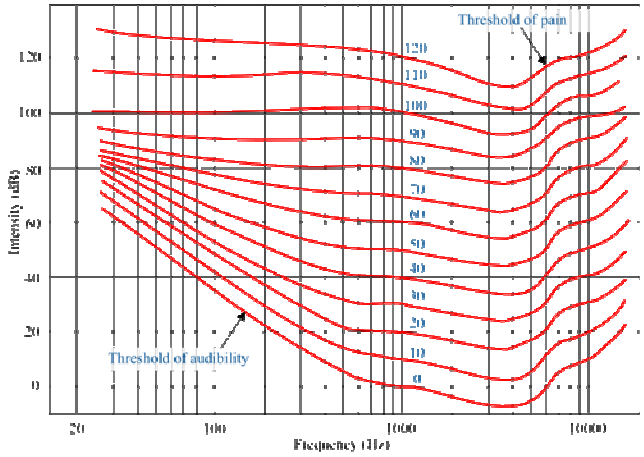


Fig. 2.1.2.2. Curvas isofónicas en dB

2.2.2 Tono

El tono es la cualidad del sonido mediante la cual el oído le asigna un lugar en la escala musical, permitiendo, por tanto, distinguir entre los graves y los agudos. La magnitud física que está asociada al tono es la frecuencia y aunque entre los dos términos exista una muy estrecha relación, no se refieren al mismo fenómeno. El tono es una magnitud subjetiva y se refiere a la altura o gravedad de un sonido. Sin embargo, la frecuencia es una magnitud objetiva y mensurable referida a formas de onda periódicas.

El tono de un sonido aumenta con la frecuencia, pero no en la misma medida. Con la frecuencia lo que medimos es el número de vibraciones. Su unidad de medida es el herzio (Hz) que se refiere a tantas vibraciones por segundo. Así un frecuencia de 1 Herzio es lo mismo que decir que el sonido tiene una vibración por segundo.

Los sonidos percibidos como graves corresponden a frecuencias bajas, mientras que los agudos son debidos a frecuencias altas. Así el sonido más grave de una guitarra corresponde a una frecuencia de 82,4 Hz y el más agudo a 698,5 Hz.

No todas las ondas sonoras pueden ser percibidas por el oído humano, el cual es sensible únicamente a aquellas cuya frecuencia está comprendida entre los 20 y los 20 KHz. En el aire dichos valores extremos corresponden a longitudes de onda que van desde 16 metros hasta 1,6 centímetros respectivamente. En general se trata de ondas de pequeña amplitud.

Junto con la frecuencia, en la percepción sonora del tono intervienen otros factores de carácter psicológico. Así sucede por lo general que al elevar la intensidad se eleva el tono percibido para frecuencias altas y se baja para las frecuencias bajas. Entre frecuencias comprendidas entre 1000 y 3000 Hz el tono es relativamente independiente de la intensidad.

2.2.3 Timbre

El timbre es la cualidad del sonido que permite distinguir sonidos procedentes de diferentes instrumentos, aun cuando posean igual tono e intensidad. Debido a esta misma cualidad es posible reconocer a una persona por su voz, que resulta característica de cada individuo.

Este fenómeno es debido a que un sonido no esta formado sólo de una frecuencia, sino por la suma de otras que son múltiplos de la fundamental. Estas otras frecuencias varían en intensidad y son llamados armónicos. La proporción e intensidad de estos armónicos son diferentes en cada instrumento y es por ello que podemos diferenciar sus sonidos.

Jean Joseph Fourier demostró matemáticamente que:

“toda onda periódica de frecuencia f puede ser descompuesta en una cantidad infinita de ondas sinusoidales de frecuencias ***f, 2f, 3f, 4f … ”***

Las senoidales carecen de armónicos, por lo cual podemos considerarlas puras. Este modo de descomponer una señal es conocido como análisis de Fourier.

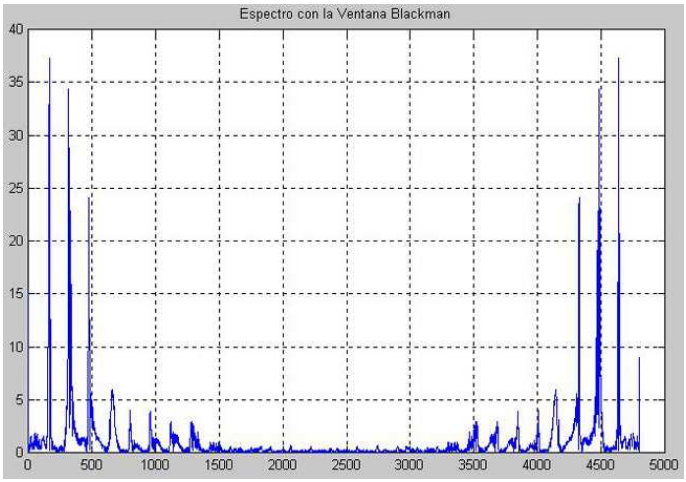


Fig. 2.2.3.1. Espectro frecuencial de una señal de audio

Si a una señal se le van añadiendo armónicos, la forma de onda irá variando pero su frecuencia fundamental permanecerá inalterada. Por lo tanto vemos que el timbre varía en razón de los armónicos mientras que la frecuencia se mantiene. Cabe destacar que la posición en frecuencia de los armónicos no es aleatoria. Los armónicos se sitúan en los múltiplos de ½ de la frecuencia fundamental. Es decir:

$$f_a = i * \frac{f_f}{2}$$

siendo f_a frecuencia del armónico, i el número de armónico y f_f la frecuencia fundamental

Las amplitudes relativas de cada armónico varían en función de la forma de onda, siendo normalmente el de mayor amplitud el que se considera fundamental. Por este motivo el timbre está relacionado con la complejidad de las ondas sonoras que llegan al oído. Sólo los diapasones son capaces de generar sonidos puros, que son debidos a una sola frecuencia y representados por una onda armónica. Los instrumentos musicales, por el contrario, dan lugar a un sonido más rico que resulta de vibraciones complejas. Cada vibración compleja puede considerarse compuesta por una serie de vibraciones armónico simples de una frecuencia y de una amplitud determinadas, cada una de las cuales, si se considerara separadamente, daría lugar a un sonido puro. Esta mezcla de tonos parciales es característica de cada instrumento y define su timbre.

2.2.4 Duración y curva ADSR

Además de la intensidad, tono y timbre también podemos caracterizar un sonido o señal musical dependiendo de su duración. La duración de un sonido es el tiempo que tarda desde que empieza hasta que termina.

A lo largo de toda la duración cualquier sonido podemos dividir este en varias partes teniendo en cuenta que existe una variación de amplitud o intensidad de la señal. Para expresar esta variación se utiliza la función ADSR (Attack Decay Sustain Release). Esta variación de amplitud también se asocia al envolvente acústico, que será diferente para cada instrumento.

En la siguiente imagen podemos distinguir los diferentes parámetros que conforman la función ADSR.

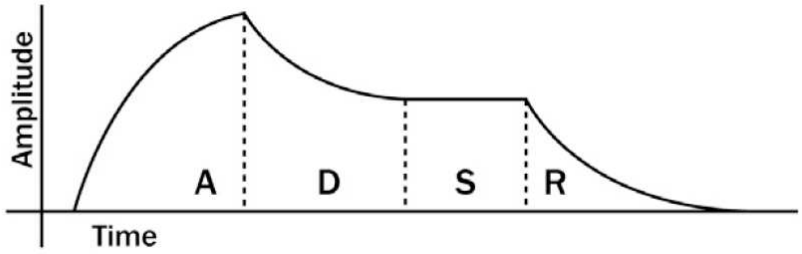


Fig. 2.2.4.1. Envolvente temporal característica. Curva ADSR.

El **tiempo de ataque** (**A** - Attack Time) es el tiempo que tarda en escucharse el sonido después de haber sido tocado el instrumento. Desde que se toca hasta que se alcanza la amplitud máxima.

El **tiempo de decaimiento** (**D** - Decay Time) es el tiempo de atenuación de amplitud después de haber alcanzado el máximo hasta que se estabiliza.

El **sostenimiento** (**S -** Sustain) va asociado al tiempo de duración máximo que suena una nota manteniendo su amplitud una vez ha sido pulsada. Este factor viene determinado en gran parte por la construcción y constitución de cada instrumento. Por ejemplo, en el caso de una guitarra, hay que mencionar que el sostenimiento también depende de las cuerdas. Las graves tienen más mientras que las agudas menos.

Por último la **relajación** (**R** - Release Time) es el tiempo que tarda el sonido en perder toda su amplitud después de pasar por la fase de sostenimiento.

2.3 El sonido en la música

El objetivo principal de este secuenciador de audio es poder componer piezas musicales sencillas. Para poder desarrollarlo es necesario entender qué tipo de orden y lógica tienen los sonidos dentro de la disciplina de la música.

El sonido, en combinación con el silencio, es la materia prima de la música pero para convertirse en materia artística, debe partir de un orden. Del mismo modo que un bote de pintura no puede ser nunca un cuadro, un sonido sin más, no puede ser nunca una obra musical. Así pues, el sonido ha de ser ordenado, y a esta ordenación, se le llama escala musical.

Una escala es una serie de sonidos que van desde la frecuencia más baja a la más alta siguiendo intervalos de frecuencia definidos. La variedad de tonos que nuestro oído es capaz de percibir es muy elevada. Es por ello que es preciso elegir ciertas frecuencias o tonos para disponer de un conjunto de sonidos que permitan la construcción de las melodías, es decir, necesitamos de las notas de la escala para componer y ejecutar la música. La construcción de la escala musical se realiza a partir de la existencia de la octava. Una octava es el intervalo que separa dos sonidos cuyas frecuencias fundamentales tienen una relación de dos a uno. La octava fue descubierta por Pitágoras al observar que una cuerda que vibra en toda su extensión produce un sonido parecido pero más grave que el de la misma cuerda reducida a la mitad de su longitud.

Es necesario disponer de un número limitado de sonidos o notas para poder dividir la octava. Las notas que dividen una octava son:

DO RE MI FA SOL LA SI

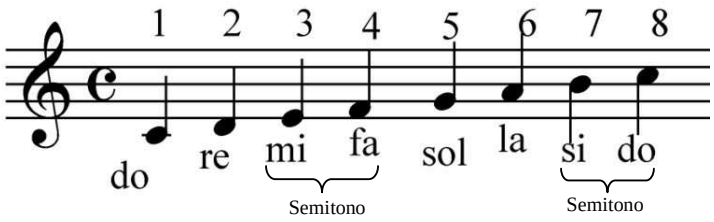
o según la nomenclatura inglesa:

C D E F G A B

El número de notas musicales de una octava ha variado según las diversas culturas y su tipo de escala musical. Las más destacables son:

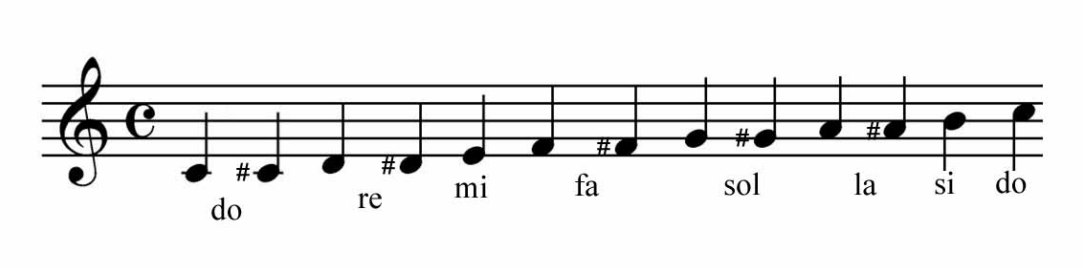
<u>La escala diatónica</u>

Desde la Edad Media las escalas que más se han utilizado son las escalas diatónicas, que se pueden simbolizar con las teclas blancas del piano. Estas escalas tienen dos intervalos diferentes: el semitono (en las teclas blancas, mi-fa y si-do) y tonos completos (entre las otras parejas de notas adyacentes). Tienen siete notas por octava.



La escala cromática

A finales del siglo XIX, y dado el hecho del uso cada vez más frecuente de los sostenidos y los bemoles, la música occidental comenzó a basarse no en la escala diatónica, sino en la cromática. Se compone de 12 notas por octava, separadas por un semitono. El semitono se define como la distancia entre notas contiguas en el piano, incluyendo las negras. El tono equivale a dos semitonos y una octava esta formada por doce semitonos.



Do Do# Re Re# Mi Fa Fa# Sol Sol# La La# Si

La escala temperada

Los problemas de afinación en instrumentos con intervalos fijos (piano, guitarra, etc.), hizo construir una escala en la que el intervalo entre dos notas consecutivas fuese siempre el mismo. Esta es la escala temperada, que consta también de doce notas, como la cromática, pero la relación de la frecuencia de una nota y la anterior es siempre igual:

$$F_{n+1} = F_n * 2^{(1/12)}$$

Esta relación es obtenida sabiendo que al aumentar una octava, una nota queda multiplicada por dos y, además, la octava se divide en doce partes de forma logarítmica, ya que la recepción auditiva del sonido en el cerebro humano sigue pautas logarítmicas. Si se quiere aumentar un semitono se multiplica la nota por dos elevado a 1/12. Si por el contrario se quiere bajarlo se dividirá por dos elevado a 1/12. El numerador del exponente se corresponde con el numero de semitonos de distancia que queremos calcular.

En 1939 se fijó la frecuencia de una nota de referencia, a partir de la cual poder deducir todas las otras. La nota y frecuencia escogidas fueron el LA_4 a 440 Hz.

Este tipo de escala será la utilizada en el presente proyecto para la implementación de los instrumentos guitarra y sintetizador. A partir del LA_4 de referencia se obtendrán todos los semitonos necesarios para cada instrumento, dependiendo del registro que pueda reproducir cada uno de ellos.

Una vez conocidos los parámetros, cualidades y cómo se ordena el sonido en la música y los instrumentos, se debe investigar cuál es la mejor forma de síntesis para poder imitar los instrumentos a implementar.

2.4 Síntesis de sonido

La síntesis de un sonido que se aproxime a lo natural es de interés tanto como para investigadores en acústica como también para compositores convirtiéndose en un elemento importante en la música de vanguardia. Consiste en obtener sonidos a partir de medios no acústicos; variaciones de voltaje en el caso de la síntesis analógica, o por medio de programas de ordenador en el caso de la síntesis digital.

Existen diferentes métodos de síntesis, entre ellos:

- **Síntesis aditiva**. Consiste en la superposición o mezcla de ondas simples para construir ondas complejas, de manera similar a como funciona un órgano de tubos.
- **Síntesis substractiva**. Se sintetiza el sonido, mediante la filtración de una onda compleja. La señal pasa a través de un filtro que modifica su contenido armónico, atenuando o reforzando determinadas áreas del espectro de la señal.
- **Síntesis por modulación**. Comprende los métodos en los que se altera algún parámetro de una onda en razón de otra onda, para producir ondas con espectros complejos. En esta categoría podemos notar dos métodos bastante usuales: **Síntesis por modulación de amplitud** (AM), que consiste en alterar la amplitud de una señal portadora en función de la onda moduladora y **síntesis por modulación de frecuencias** (FM), que consiste básicamente en variar la frecuencia de una onda portadora en función de la forma de otra onda moduladora.
- **Síntesis por tabla de ondas**. También llamada Wavetable sintetiza el sonido reproduciendo una serie de muestras que están guardadas en una memoria.
- **Síntesis por modelos físicos**. La síntesis se hace a partir de la simulación en un ordenador de un objeto físico y sus características.
- **Síntesis granular**. La síntesis granular es una técnica que parte de una visión atomista del sonido, en la que los elementos mínimos son pequeños fragmentos de sonido llamados 'granos'.

2.4.1. Síntesis aditiva

Como se ha explicado en los puntos anteriores los timbres están formados por cantidades variables de armónicos o parciales que cambian a lo largo del tiempo con respecto a un tono o frecuencia fundamental. Los parciales son las ondas que complementan a la onda fundamental para crear un timbre, si las frecuencias de los parciales son múltiplos enteros de la frecuencia fundamental son denominados parciales armónicos, y si son múltiplos reales son denominados no armónicos.

En la síntesis aditiva es muy importante la utilización de diferentes envolventes que se encargan del manejo la amplitud sobre cada parcial y es lo que estructura el comportamiento del sonido en el tiempo.

Para realizar el proceso se hace necesario disponer de un banco de osciladores para que generen las diferentes ondas que complementan la onda fundamental cada una con amplitudes y frecuencias diferentes.

Así pues, se puede implementar cualquier sonido periódico en el dominio del tiempo basándose en el descubrimiento de Jean Joseph Fourier y que viene determinado por:

$$y[n] = \sum_{i=1}^{N} x_i[n] = \sum_{i=1}^{N} a_i \sin(2\pi f_i t + \varphi')$$

donde cada función $x_i[n]$ es una sinusoide de amplitud a_i, frecuencia f_i y fase inicial φ', y t es la duración.

Las siguientes figuras muestran el resultado de la suma de las componentes de una onda cuadrada. Esta forma de onda presenta dos características: la primera es que la amplitud relativa de cada una de los armónicos adicionales decrece con el orden del armónico; y la segunda es que sólo están presentes los armónicos de orden impar. Las figuras ilustran las etapas de la suma de armónicos en una serie de formas de onda en el dominio temporal:

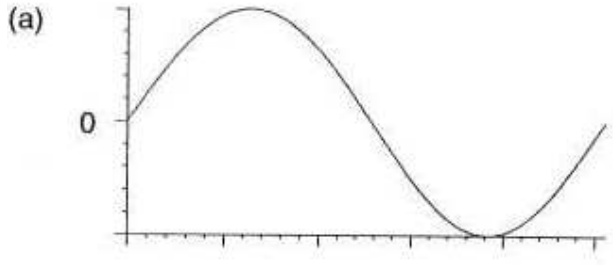


Fig. 2.4.1.1. (a) Frecuencia fundamental

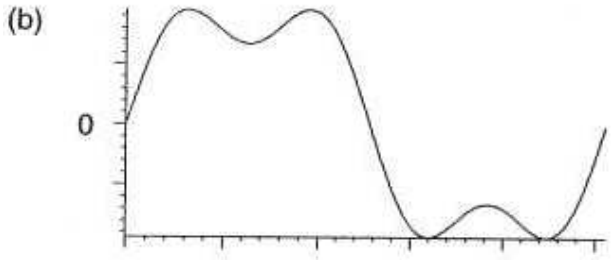


Fig. 2.4.1.2. (b) Primer y tercer armónico

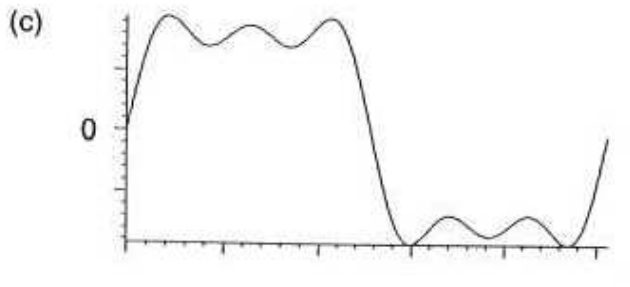


Fig.2.4.1.3. (c) Suma de los armónicos impares hasta el quinto

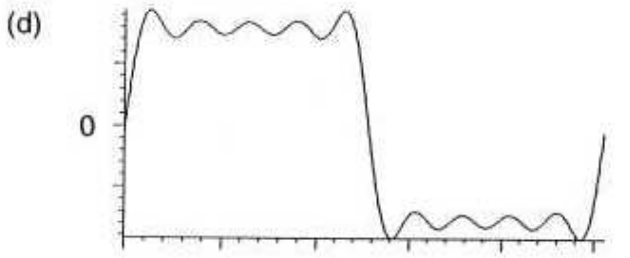


Fig. 2.4.1.4. (d) Suma de los armónicos impares hasta el noveno

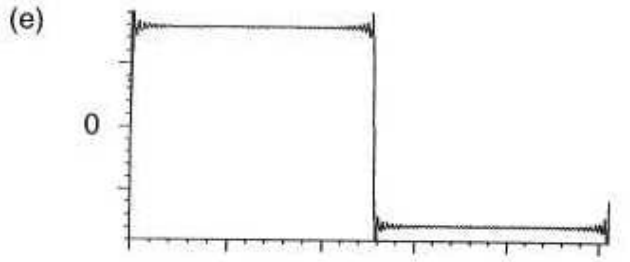


Fig. 2.4.1.5. (e) Forma de onda aproximada a la cuadrada obtenida sumando los armónicos impares hasta el 101

Con este ejemplo se puede demostrar claramente que cuanto mayor sea el número de armónicos o señales senoidales que se superponen a la señal total más se parecerá y más fiel será el sonido con respecto al original.

La síntesis aditiva por si misma no es siempre satisfactoria al oído. Este tipo de síntesis es muy limitada, porque no puede darnos un espectro dinámico, es decir, que varía en el tiempo. La forma de onda, aunque contenga diferentes armónicos, no se comporta como en los sonidos naturales donde la onda cambia con el tiempo. Además, como se ha explicado en apartados anteriores, los sonidos obtienen sus características

distintivas gracias a los períodos de ataque y subida (ADSR), períodos durante los cuales el espectro del sonido es muy diferente y varía rápidamente. El problema que representa en la síntesis aditiva es que las características esenciales son incontrolables.

La solución a este problema es combinar la síntesis aditiva con una multiplicación de la señal para variar su amplitud en función del tiempo. Mediante una señal exponencial decreciente que sea similar a la curva ADSR del instrumento que se quiera implementar se obtiene la envolvente caracteristica a la curva ADSR del intrumento. La señal creada mediante síntesis aditiva se multiplicará por esta exponencial decreciente consiguiendo una señal nueva con las características de ataque, caída, sostenimiento y relajación muy similares a las del sonido original.

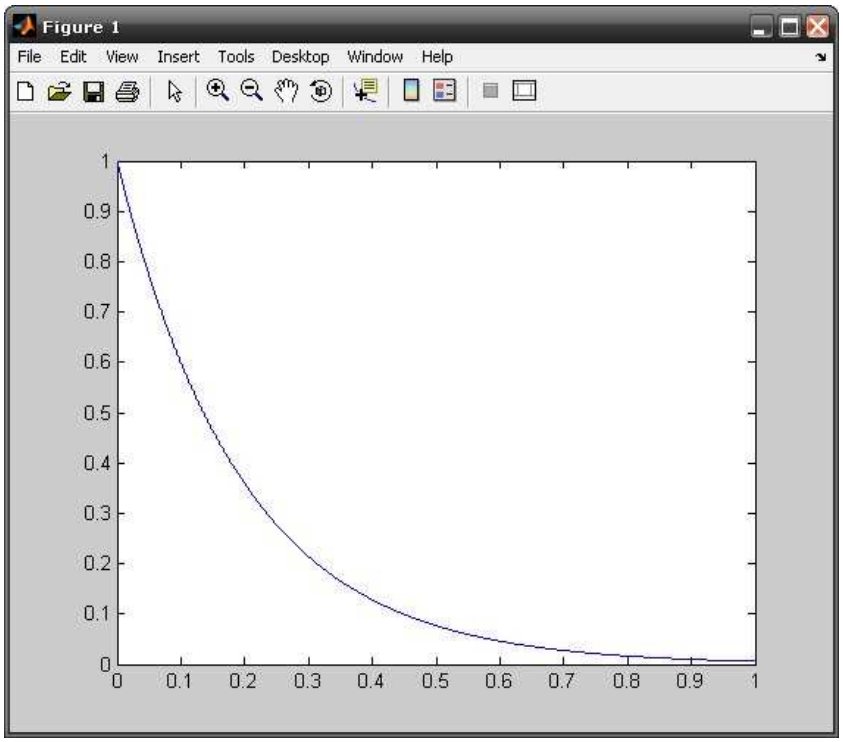


Fig. 2.4.1.6. Exponencial decreciennte aplicada a las notas sintetizadas

En el presente proyecto se utilizará la síntesis aditiva para generar el timbre característico de una guitarra. Esto es debido a que uno de los objetivos de este proyecto es el estudio espectral del timbre que la caracteriza y poder sintetizarla haciendo referencia a los armónicos obtenidos en su análisis.

2.4.2 Caso particular de síntesis. El sintetizador.

Como caso particular de síntesis cabe destacar un instrumento muy comercializado en la acualidad para llevar a cabo esta función llamado sintetizador. Un sintetizador es un instrumento musical electrónico diseñado para producir sonido generado artificialmente, usando las técnicas de síntesis enumeradas anteriormente.

Existen sintetizadores analogicos los cuales crean sonidos mediante manipulación directa de corrientes eléctricas; sintetizadores digitales que crean el sonido mediante la manipulación de una onda FM digital; o los sintetizadores basados en software que combinan cualquier metodo manipulando los valores discretos usando los ordenadores.

Este sonido sintético se distingue de la grabación de sonido natural, donde la energía mecánica de una onda de sonido se transforma en una señal que más tarde se convertirá de nuevo en energía mecánica durante su reproducción.

En los sintetizadores actuales utilizan mayoritariamente los métodos de síntesis por modulación FM y síntesis por tabla de ondas. Por este motivo es interesante hacer una breve explicación de estos dos tipos de síntesis.

2.4.2.1 Síntesis FM

Este fue uno de los primeros sistemas que permitió una riqueza sonora considerable, con un pequeño coste computacional. John Chowning, de la universidad de Stanford, patentó este método en 1973, y lo licenció a Yamaha dos años más tarde. La compañía japonesa tardó siete años en diseñar y fabricar el chip que permitiese ejecutar, en tiempo real, el algoritmo que Chowning había implementado en software.

Al contrario que la sintesis aditiva, que utiliza tantos osciladores como señales parciales contenga, la síntesis FM necesita tan sólo dos osciladores: la señal portadora y la señal moduladora. Parte de la idea de que cuando la moduladora no es una señal de baja frecuencia sino que entra ya en el rango de las frecuencias audibles (a partir de los 20 Hz) se crean un gran número de frecuencias adicionales que generan un sonido con un gran contenido armónico. La figura siguiente muestra la onda obtenida de la modulación en frecuencia de las mismas ondas sinusoidales del ejemplo anterior. En este caso, la onda obtenida no mantiene ya la frecuencia de la onda más grave.

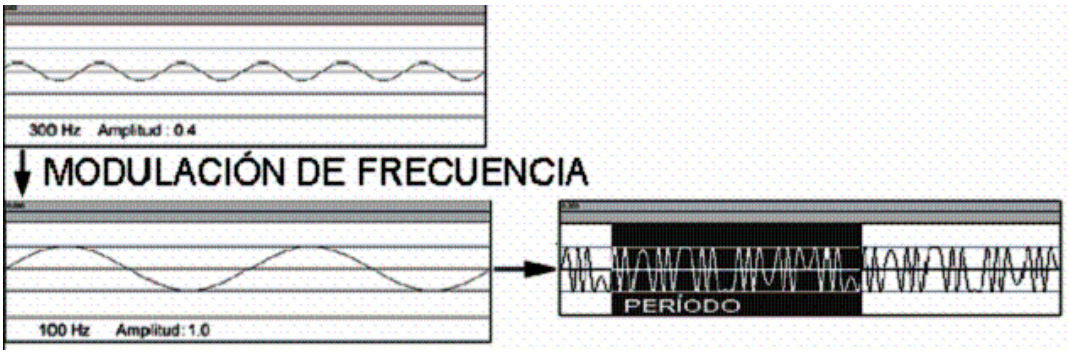


Fig. 2.4.2.1.1. Ejemplo básico de modulación en frecuencia

Este fenómeno no tiene una equivalencia en la naturaleza y, aunque permite generar sonidos de gran riqueza, es difícil programarlo para obtener sonidos imitativos, por lo que hoy en día ha quedado un tanto desbancado.

Se puede expresar la modulación en frecuencia mediante la siguiente ecuación matemática:

$$x(t) = \sin(2\pi f_p t + I \sin(2\pi f_m t)t)$$

siendo:

- f_p : frecuencia de la onda portadora
- f_m : frecuencia de la onda moduladora
- I : indice de modulación

El número de parciales frecuenciales o armónicos, y la amplitud de cada uno de ellos depende únicamente de la magnitud del índice de modulación. Se conoce que a medida que aumenta el índice de modulación aparecen más armónicos en el espectro frecuencial de la señal. Sin embargo, la amplitud de cada parcial no es lineal, sino que sigue un patrón de comportamiento no evidente. Para determinar la amplitud de cada uno de estos parciales se utilizan las funciones de Bessel que complican la programación de esta función para obtener sonidos imitativos.

Este es uno de los motivos por los que no se utiliza la síntesis FM en el presente proyecto. Además de que la síntesis aditiva requiere un previo análisis de los instrumentos a sintetizar para conocer su espectro en frecuencia. Esto hace más interesante utilizar el método de síntesis aditiva.

2.4.2.2 Síntesis por tabla de ondas (Wavetable)

La tabla de ondas, conocido con el anglicismo Wavetable es una técnica de síntesis de sonido utilizado principalmente para producir música digitalmente. Los sistemas de muestreo digital almacenan sonido de alta calidad digitalmente y reproducen estos sonidos bajo demanda.

Los avances tecnológicos de principios de los ochenta hicieron posible la sustitución de las ondas periódicas simples que se venían utilizando como material base, por pequeños fragmentos de muestras procedentes de sonidos reales, digitalizados y almacenados en memoria. Estos fragmentos pueden ser tan breves como un ciclo, ya que el sintetizador se encarga de repetirlos periódicamente. Un sintetizador compatible, por ejemplo, con el protocolo MIDI, deberá contener suficientes fragmentos para reconstruir 128 instrumentos, más 59 sonidos de percusión. Esta técnica permite muchas variaciones y refinamientos, como la combinación o la alternancia de varios fragmentos en un único instrumento, mediante sofisticados algoritmos. También utiliza a fondo todos los mecanismos descritos en la post-síntesis (envolventes, filtros y moduladoras).

En los sonidos naturales, es frecuente que el timbre varíe mucho en el ataque, permaneciendo más o menos constante a continuación, por lo que en muchas ocasiones las dos partes se almacenan por separado y el sintetizador las combina en tiempo real. Más de la mitad de los sintetizadores fabricados en los últimos diez años implementan alguna variante de este método de síntesis. Es también el utilizado en una gran mayoría de las tarjetas de sonido.

Sin embargo, pese a ser uno de los métodos de síntesis que mejor funciona, la síntesis por tabla de ondas no suscita ningún tipo de motivación el implementarla en el presente proyecto. Esto se debe a que, la simple reproducción de archivos de muestra, no requiere ningún tipo de estudio de los parámetros acústicos de los sonidos de una guitarra, y a su vez, no sería necesaria la investigación de cómo poder imitar un sonido característico.

3. Modelado de instrumentos

Como se ha comentado en la introducción y en algunos apartados el objetivo del presente proyecto final de carrera es implementar un secuenciador para poder realizar composiciones musicales más o menos sencillas. Para que una composición sea más atractiva, es necesario que disponga de más de un instrumento musical. Por este motivo se ha decidido implementar en el secuenciador a desarrollar un total de 3 instrumentos. Una guitarra que podría hacer el acompañamiento, un sintetizador que se podría utilizar para crear melodías y una pista de percusión (caja y bombo) para poder marcar el ritmo de la composición musical. Además, como curiosidad, se ha implementado uno de los efectos más utilizados en la música, la distorsión. La distorsión podrá ser aplicada en las pistas de guitarra y sintetizador.

Es necesario detacar que la síntesis de estos instrumentos se va a realizar con MatLab. Por este motivo en las posteriores explicaciones de análisis y síntesis se harán referencias a los comandos y líneas de código utilizadas.

3.1 La guitarra

3.1.1 Registro de la guitarra

La guitarra es un instrumento de seis cuerdas formado por una caja de resonancia y un mástil. El mástil tiene en su parte frontal una placa de madera llamada diapasón. El diapasón esta dividido con unas tiras de metal llamadas trastes. Al pulsar una cuerda sin pisar ningún traste, suena lo que llamamos nota al aire. Ordenando las cuerdas de más grave a más aguda, las notas al aire en una afinación estándar son:
MI, LA, RE, SOL, SI, MI.

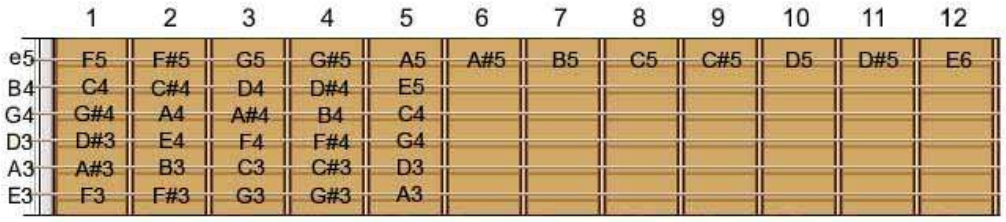


Fig. 3.1.1.1. Mastil de una guitarra con su propio registro musical

Como se puede ver en la imagen, el registro musical de una guitarra llega hasta 3 octavas. Es decir, una guitarra estándar de 6 cuerdas y con afinación normal tan sólo puede reproducir desde la nota Mi3 hasta la nota Mi6. Por este motivo, tan sólo será necesario analizar este rango de notas musicales.

3.1.2 Análisis de la guitarra

Como se ha explicado anteriormente, para poder sintetizar el sonido de una guitarra se necesita conocer sus componentes espectrales. De esta forma se puede conseguir un timbre bastante fiel al sonido real.

Con un software de audio (‘FL Studio 9’) se han obtenido muestras reales de las notas de las 3 octavas que puede reproducir una guitarra. Primeramente calculamos la transformada de Fourier de cada una de las notas de la guitarra. Para hacer este cálculo se ha creado en MatLab un archivo llamado analize.m.

A continuación se muestra el código de la función implementada y la explicación de cada una de las líneas de código.

```
function analize=analize(sonido)

x=wavread(sonido); % Se lee el archivo .wav de la nota a analizar
x=x(:,1);   % Al ser estéreo nos quedamos sólo con un canal
l=length(x);  % Calculamos la longitud del vector de la nota musical
figure(1);plot(x);Title('representación en tiempo')  % representamos
la nota en funcion del tiempo
Y=fft(x/(l/2)); % Calculamos la Transformada de Fourier normalizando
este calculo dividiendo la señal por el numero de muestras partido 2
f=44100*linspace(0,1,l); % Creamos un vector de la longitud de la nota
para poder representarla en frecuencia
figure(2); plot(f,abs(Y)); Title('representación en frec') %
Representamos el modulo de la Transf. Fourier para observar las
componentes espectrales
```

Al representar esta señal en frecuencia se pueden identificar las distintas deltas que representan la frecuencia fundamental y sus armónicos. Desde la misma gráfica se leen las amplitudes de cada una de las deltas y se crea un vector con estas amplitudes.

Se ha decidido que cada vector contenga 15 posiciones. En notas graves existirán posiciones del vector vacías ya que tendrá un número reducido de armónicos. Sin embargo en notas musicales agudas habrá incluso más de 15 armónicos. Por este motivo se decide que el vector de componentes tenga un número suficiente de posiciones para poder simular el timbre con fidelidad. En la siguiente imagen se muestra la representación en frecuencia de la nota Sol5 cuyo vector de componentes es:

```
comp=[ 0.0274   0.0170   0.0140   0.0002   0.0004   0.0165   0.0020   0.0003   0.0004
       0.0001   0.0011   0.0038   0.0018   0.0007   0.0003];
```

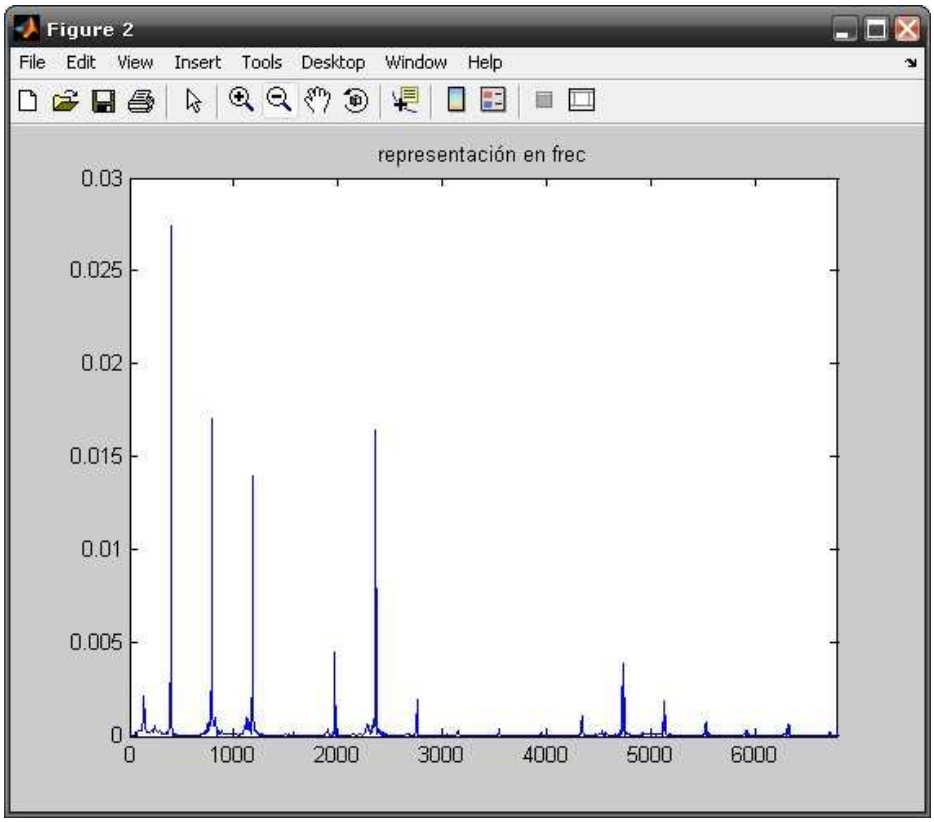


Fig. 3.1.2.1. Representación frecuencial de una nota Sol5 de guitarra

Este procedimiento es repetido con cada una de las notas de la guitarra obteniendo tantos vectores de componentes como notas tiene la guitarra (37 notas implementadas). En las siguientes tablas se exponen todos y cada uno de los vectores de componentes frecuenciales de cada una de las notas.

	Mi3	Fa3	Fa#3	Sol3	Sol#3	La3	La#3	Si3
f/2	0.0068	0.0213	0.0334	0.0645	0.0927	0.1316	0.1659	0.1132
f	0.0961	0.0974	0.0948	0.0724	0.0645	0.0421	0.0237	0.0158
3f/2	0.0237	0.0111	0.0092	0.0132	0.0066	0.0145	0.0011	0.0026
2f	0.0050	0.0050	0.0058	0.0066	0.0016	0.0039	0.0008	0.0004
5f/2	0.0005	0.0011	0.0000	0.0009	0.0009	0.0005	0	0
3f	0.0016	0.0005	0.0020	0.0018	0.0021	0.0009	0.0005	0.0005
7f/2	0.0011	0.0014	0.0009	0.0021	0.0004	0.0032	0.0004	0.0008
4f	0.0003	0.0011	0.0014	0.0041	0	0.0037	0.0026	0.0007
9f/2	0.0016	0.0004	0	0	0.0003	0.0007	0.0001	0.0004
5f	0.0004	0.0004	0	0	0.0013	0	0	0
11f/2	0	0	0	0	0	0	0	0
6f	0	0	0	0	0	0	0	0
13f/2	0	0	0	0	0	0	0	0
7f	0	0	0	0	0	0	0	0
15f/2	0	0	0	0	0.0001	0	0	0

	Do4	Do#4	Re4	Re#4	Mi4	Fa4	Fa#4	Sol4
f/2	0.2040	0.1685	0.1290	0.0927	0.0816	0.0684	0.0763	0.0487
f	0.0105	0.0066	0.0079	0.0090	0.0197	0.0184	0.0266	0.0408
3f/2	0.0145	0.0058	0.0092	0.0047	0.0058	0.0013	0.0016	0.0074
2f	0.0013	0.0007	0.0008	0.0013	0.0022	0.0016	0.0018	0.0134
5f/2	0	0.0007	0.0014	0.0007	0	0.0061	0.0037	0.0007
3f	0.0021	0.0008	0.0018	0.0012	0.0013	0.0050	0.0005	0.0012
7f/2	0.0011	0.0013	0.0021	0.0009	0.0008	0.0008	0.0007	0.0001
4f	0.0008	0	0.0032	0.0006	0	0	0	0.0002
9f/2	0	0	0.0007	0	0	0	0.0002	0.0005
5f	0	0	0.0002	0.0002	0	0	0.0002	0.0005
11f/2	0	0	0	0.0002	0	0.0001	0.0002	0.0001
6f	0	0	0.0001	0.0003	0.0006	0.0002	0.0001	0.0005
13f/2	0	0	0.0001	0.0005	0	0.0009	0.0005	0.0005
7f	0	0	0.0003	0.0003	0	0.0005	0.0002	0.0004
15f/2	0	0	0.0006	0.0001	0	0	0.0002	0.0001

	Sol#4	La4	La#4	Si4	Do5	Do#5	Re5	Re#5
f/2	0.0658	0.0276	0.0284	0.0211	0.0111	0.0153	0.0176	0.0240
f	0.0190	0.0153	0.0092	0.0125	0.0100	0.0108	0.0090	0.0084
3f/2	0.0033	0.0037	0.0066	0.0082	0.0090	0.0026	0.0132	0.0187
2f	0.0030	0.0008	0.0037	0.0116	0.0029	0.0045	0.0019	0.0022
5f/2	0.0005	0.0014	0.0008	0.0020	0.0008	0.0001	0.0021	0.0005
3f	0.0005	0.0001	0.0006	0.0003	0.0001	0.0001	0.0004	0.0004
7f/2	0.0003	0	0.0004	0.0009	0.0010	0.0029	0.0042	0.0006
4f	0.0001	0.0003	0.0006	0.0016	0.0015	0.0016	0.0092	0.0041
9f/2	0.0004	0.0005	0.0002	0.0007	0.0047	0.0013	0.0003	0
5f	0.0006	0.0001	0.0017	0.0005	0.0009	0.0001	0.0007	0.0001
11f/2	0.0003	0.0002	0.0002	0.0001	0	0	0.0003	0.0003
6f	0.0002	0.0001	0.0002	0.0001	0	0.0002	0	0.0001
13f/2	0.0001	0.0002	0.0001	0.0001	0	0	0.0001	0.0004
7f	0.0002	0.0001	0	0.0001	0	0.0009	0.0008	0
15f/2	0	0	0	0.0001	0.0004	0.0006	0.0020	0.0004

	Mi5	Fa5	Fa#5	Sol5	Sol#5	La5	La#5	Si5
f/2	0.0132	0.0140	0.0150	0.0274	0.0197	0.0108	0.0076	0.0092
f	0.0118	0.0076	0.0129	0.0170	0.0284	0.0571	0.0384	0.0295
3f/2	0.0408	0.0376	0.0091	0.0140	0.0051	0.0011	0.0010	0.0012
2f	0.0039	0.0020	0.0036	0.0002	0.0004	0.0003	0.0012	0.0053
5f/2	0.0009	0.0004	0.0009	0.0004	0.0068	0.0039	0.0043	0.0032
3f	0.0010	0.0012	0.0030	0.0165	0.0057	0.0042	0.0004	0.0026
7f/2	0.0155	0.0121	0.0029	0.0020	0.0001	0.0020	0.0005	0.0007
4f	0.0093	0.0014	0.0038	0.0003	0.0000	0.0004	0.0001	0.0006
9f/2	0.0023	0.0003	0.0002	0.0004	0.0008	0.0013	0.0003	0.0003
5f	0.0005	0.0002	0.0002	0.0001	0.0017	0.0013	0.0020	0.0024
11f/2	0.0001	0.0001	0.0016	0.0011	0.0030	0.0022	0.0020	0.0016
6f	0.0001	0.0005	0.0014	0.0038	0.0018	0.0008	0.0017	0
13f/2	0.0003	0.0007	0.0023	0.0018	0	0.0016	0.0001	0
7f	0.0010	0.0029	0.0017	0.0007	0.0002	0.0014	0.0007	0.0008
15f/2	0.0015	0.0011	0.0012	0.0003	0.0001	0	0.0003	0.0003

	Do6	Do#6	Re6	Re#6	Mi6
f/2	0.0121	0.0085	0.0116	0.0190	0.0095
f	0.0140	0.0074	0.0121	0.0016	0.0042
3f/2	0.0016	0.0007	0.0004	0.0037	0.0059
2f	0.0088	0.0008	0.0182	0.0051	0.0088
5f/2	0.0072	0.0012	0.0020	0.0017	0.0008
3f	0.0004	0.0006	0.0022	0.0002	0.0012
7f/2	0.0002	0.0006	0.0007	0.0034	0.0053
4f	0.0011	0.0002	0.0014	0.0038	0.0021
9f/2	0.0019	0.0025	0.0013	0.0008	0.0008
5f	0.0021	0.0016	0.0030	0.0027	0.0002
11f/2	0.0011	0.0004	0.0013	0.0009	0.0004
6f	0.0009	0.0018	0.0025	0.0003	0.0005
13f/2	0.0006	0.0004	0.0036	0.0001	0.0003
7f	0.0004	0	0.0007	0.0001	0.0001
15f/2	0.0004	0.0001	0.0002	0.0002	0.0001

3.1.3 Síntesis de la guitarra

Una vez obtenidos todos los vectores de componentes termina el trabajo de análisis para pasar a la síntesis. Se ha explicado anteriormente que para la síntesis se va a utilizar la síntesis aditiva. Con este método se van sumando las diferentes señales senoidales con su respectiva amplitud, la cual estará almacenada en el vector de componentes creado anteriormente en el análisis.

Como se ha explicado anteriormente una señal senoidal pura está definida por la ecuación del M.A.S:

$$x = A\ sen(2\pi\ f\ t)$$

Esta señal senoidal tiene un argumento cuyas variables habrá que definir previamente para construir la señal de la nota musical a sintetizar. Estas variables son el tiempo y la frecuencia de la nota.

El tiempo será un vector cuya duración vendrá determinada por la nota musical. Es necesario definir una frecuencia de muestreo para caracterizar este vector. La frecuencia de muestreo es el número de muestras por unidad de tiempo y su unidad es el hercio (Hz). A la hora de seleccionar una frecuencia de muestreo se tendrá en cuenta que se ha de cumplir el **criterio de Nyquist** con la finalidad de realizar una representación adecuada de la forma de onda de la señal que queremos sintetizar. El teorema dice que la frecuencia de muestreo debe ser igual o superior al doble de la frecuencia máxima, es decir:

$$F_m \geq 2{*}F_{\max}$$

Así pues habrá que definir una frecuencia de muestreo acorde con la frecuencia máxima que se va a reproducir en este trabajo. En la pista de guitarra la frecuencia más aguda pertenecerá a la nota Mi6. Sin embargo, en la pista de sintetizador, la cual se explicará en puntos posteriores, podrá reproducir notas todavía más agudas y para que no existan errores en el muestreo de algunas pistas habrá que fijar la frecuencia de muestreo a todo el proyecto por igual. De este modo, la frecuencia de la nota Si8 es de 7902 Hz quedando el cálculo de la siguiente manera:

$$F_m \geq\ 2 * 7902 = 15804\ \text{Hz}$$

Así pues seleccionamos una frecuencia de muestreo de 22050 Hz ya que es necesario trabajar con un estándar y esta es la frecuencia que más se aproxima al cálculo realizado. Los estándares de frecuencia de muestreo son 8000, 22050, 44100, 48000 y 96000 Hz.

Teniendo definida la frecuencia de muestreo a utilizar se puede programar en Matlab el vector de tiempo utilizado en el argumento de la señal senoidal como:

```
t=0:1/fm:dur; % vector de tiempo definido por la frecuencia de
muestreo y la duración de la nota
```

de donde la variable dur será la duración de la nota.

En el punto 2.3 que trata sobre el sonido dentro de la música se ha comentado que se creó una nota de referencia y de ésta se pueden obtener el resto de notas que componen las 8 octavas del piano. Esta nota es el LA_4=440Hz. Para obtener la frecuencia fundamental de cada una de las notas a sintetizar se partirá de está nota de referencia. Sabiendo que cada semitono equivale a un desplazamiento en frecuencia de $F_{n+1} = F_n * 2^{(1/12)}$, se puede calcular la frecuencia fundamental desplazándose tantos semitonos hacia arriba o hacia abajo a partir de LA_4. Es decir, si se quiere desplazar 7 semitonos hacia arriba la formula quedará $F_{n+1} = F_n * 2^{(7/12)}$. Si por el contrario se quiere desplazar 7 semitonos hacia abajo $F_{n+1} = F_n * 2^{(-7/12)}$.

El comando en MatLab para calcular la frecuencia fundamental es:

```
A=440;
r=2^(1/12);
f=r^rel_oct(oct)*A; % Cálculo de la frecuencia fundamental teniendo
como referencia a A
```

donde *rel_oct* será un vector de 8 posiciones creado para cada nota donde se indica el desplazamiento de semitonos que es necesario hacer con respecto a la referencia (La_4). La variable *oct* indica la posición del vector que contendrá el desplazamiento para poder calcular la frecuencia fundamental. En la siguiente tabla se indican los vectores *rel_oct* de cada nota.

	Oct 1	Oct 2	Oct 3	Oct 4	Oct 5	Oct 6	Oct 7	Oct 8
Do	-45	-33	-21	-9	3	15	27	39
Do#	-44	-32	-20	-8	4	16	28	40
Re	-43	-31	-19	-7	5	17	29	41
Re#	-42	-30	-18	-6	6	18	30	42
Mi	-41	-29	-17	-5	7	19	31	43
Fa	-40	-28	-16	-4	8	20	32	44
Fa#	-39	-27	-15	-3	9	21	33	45
Sol	-38	-26	-14	-2	10	22	34	46
Sol#	-37	-25	-13	-1	11	23	35	47
La	-36	-24	-12	0	12	24	36	48
La#	-35	-23	-11	1	13	25	37	49
Si	-34	-22	-10	2	14	26	38	50

Con las variables tiempo y frecuencia calculadas ya se puede implementar la ecuación de síntesis en MatLab. Tan sólo habrá que tener en cuenta que habrá que hacer la suma de tantos senos como armónicos se han analizado anteriormente. En este caso se hará una superposición de un total de 15 senos y para ello se utilizará un bucle "for" con 15 iteraciones. Como se sabe por análisis de Fourier los armónicos estarán situados a frecuencias i/2 de la frecuencia fundamental, siendo i el número de armónico. La

expresión de cálculo del sumatorio de senos implementada en MatLab quedará de la siguiente forma:

```
for i=1:15
    y=y+(comp(i).*sin(2*pi*(i*f/2)*t)); % Se calcula la señal de la
nota mediante el sumatorio de senos multiplicados por sus componentes
end
```

La señal *y* es la señal de la nota musical implementada. Gracias a los cálculos realizados se ha podido obtener un timbre bastante similar al sonido original de una guitarra. Pero no varía en el tiempo como marca la curva ADSR. Para simular este efecto se crea una señal exponencial decreciente con una curva parecida a la de la curva ADSR. Al multiplicar la señal por esta exponencial la nueva señal tendrá un decaimiento parecido al sonido típico de una guitarra. Para crear la señal exponencial se tiene en cuenta que ha de depender de la duración de la nota y la amplitud de la misma. Así pues los comandos para crear esta exponencial son:

```
caida=max(y(:))+4/dur %  La caida de la exponencial dependerá del
valor máximo se la señal y de la duración de la misma
x=exp(-(caida)*t);          %  Definimos la exponencial segun la
duración de la nota
```

Y para terminar la síntesis de la guitarra tan sólo quedará multiplicar la señal de la nota musical por la exponencial creada.

En la siguiente figura se muestra una comparación de los espectros en frecuencia de dos notas Mi3 de una guitarra. La señal de color rojo representa el sonido real y la señal azul el sonido sintetizado. Se puede comprobar que prácticamente las dos señales están solapadas. Esto significa que la síntesis de la nota ha sido satisfactoria.

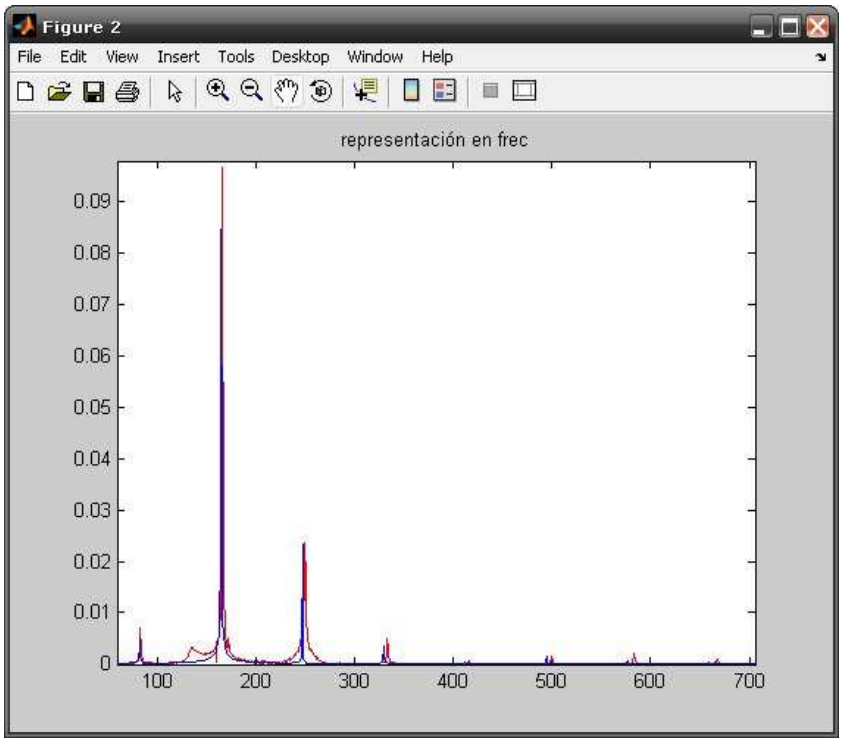


Fig. 3.1.3.1. Representación frecuencial de la nota Mi3 real (rojo) y la nota Mi3 sintetizada (azul)

3.2 El sintetizador

La pista de sintetizador es creada para apoyar a la pista de guitarra y así poder realizar composiciones musicales un poco más elaboradas. Mientras la guitarra va haciendo una base, el sintetizador puede ir recreando melodías más complejas aportando una mayor armonía a la composición musical.

En el caso de la recreación del sintetizador no será necesario el análisis de ningún instrumento musical ya que se ha decidido implementar este sonido como tonos puros sin tener que imitar un timbre determinado. Si se hubiese decidido implementar un sonido con un timbre característico habría que seguir el mismo procedimiento que el análisis y síntesis de la guitarra.

De todos modos lo que se deberá tener claro es el registro musical del instrumento sintetizador. Se ha decidido que este instrumento sea capaz de reproducir las 8 octavas del piano para así poder tener más rango frecuencial a la hora de componer melodías y acompañamientos.

Para realizar la síntesis del sintetizador habrá que seguir el mismo procedimiento seguido en la guitarra con la diferencia que en este caso no hay que realizar un sumatorio de señales senoidales. La nota vendrá definida por su frecuencia fundamental y su duración. La frecuencia fundamental se calculará del mismo modo que en la

guitarra. Se utilizarán los vectores de cada nota que relacionan su desplazamiento entre octavas tomando como referencia la frecuencia LA_4.

La ecuación de síntesis de este intrumento implementada en MatLab queda:

```
y=sin(2*pi*f*t);
```

3.3 Efecto: La distorsión

Se entiende por **distorsión** la diferencia entre señal que entra a un equipo o sistema y la señal de salida del mismo. Por tanto, puede definirse como la "deformación" que sufre una señal tras su paso por un sistema. La distorsión puede ser lineal o no lineal.

Normalmente, cuando en audio se habla de distorsión se hace referencia a la distorsión armónica. Es un parámetro técnico utilizado para definir la señal de audio que sale de un sistema.

La distorsión armónica se produce cuando la señal de salida de un sistema no equivale a la señal que entró en él. Esta falta de linealidad afecta a la forma de la onda, porque el equipo ha introducido armónicos que no estaban en la señal de entrada. En todo sistema de audio siempre se produce una pequeña distorsión de la señal, dado que todos los equipos actuales introducen alguna no linealidad.

Pero la distorsión armónica no siempre implica pérdida de calidad. De hecho, la distorsión se considera un efecto de sonido imprescindible para ciertos géneros musicales (básicamente rock) y así, se suele saturar artificialmente la señal básica producida por ciertos instrumentos (como guitarras). Este es el motivo principal por el cual se ha decidido implementar este efecto y no cualquier otros.

Para generar el efecto de distorsión habrá que deformar la señal intentandola hacer más cuadrada. De esta forma se generarán un mayor número de armónicos produciendo este peculiar efecto. El código MatLab implementado para realizar este efecto ha sido encontrado en la pagina www.musicdsp.org y, básicamente, su función es recortar la señal. Este código aplica la distorsión dándole como parametro una ganancia (a) que debe estar entre los valores: $-1 \leq a \leq 1$. El código se muestra a continuación:

```
k = 2*a/(1-a); % Calcula la amplificación que se le aplica a la señal
y=(1+k)*(y)./(1+k*abs(y)); % Aplica la distorsión dividiendo ambas
señales
```

Para que este código quede más claro a continuación se muesta una figura que contiene la señal del numerador en color rojo, la señal del denominador en azul y la señal distorsionada en verde. La figura se ha obtenido aplicando una distorsion con ganancia *a=0.75*

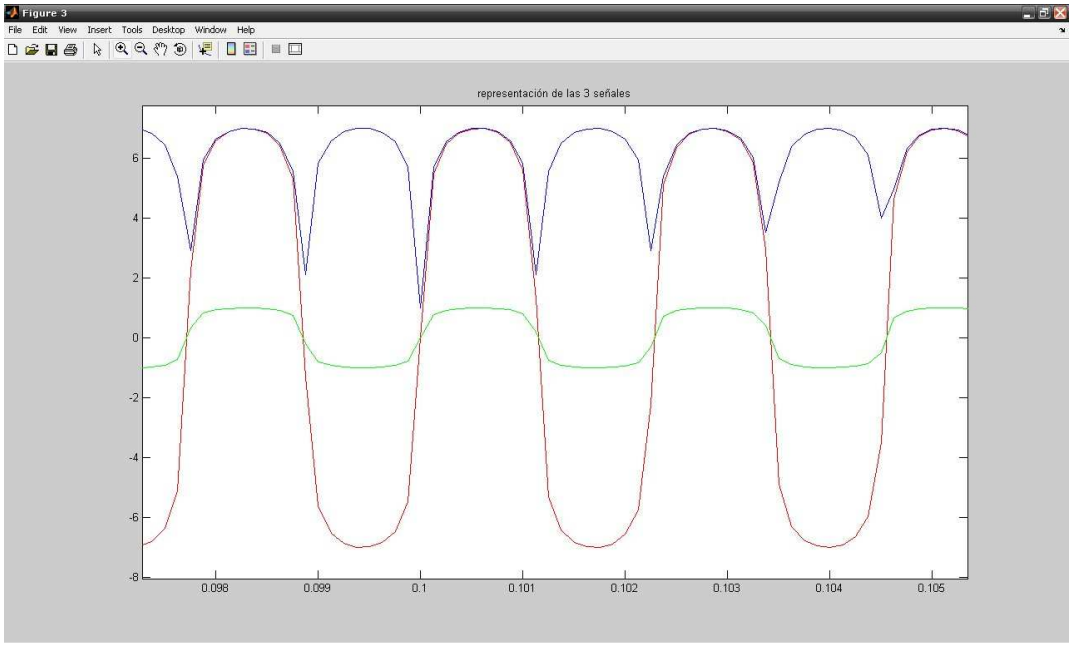


Fig. 3.3.1. Representación temporal de las señales del numerador (rojo), denominador (azul) y la salida (verde) del cálculo de la distorsión. a=0.75;

Se puede observar que la señal de salida se ha deformado notablemente. Esto se debe a que cuando la señal roja tiene amplitud cero la señal de salida también tendrá amplitud nula. A medida que las amplitudes de las señales roja y azul son similares o iguales, al realizar la división entre ambas, se obtendrán valores de amplitud próximos a 1, quedando la señal de salida (verde) recortada y deformada.

Para comprobar como afecta el valor de la ganancia se muestran a continuación una serie de figuras con distinto valor de *a*.

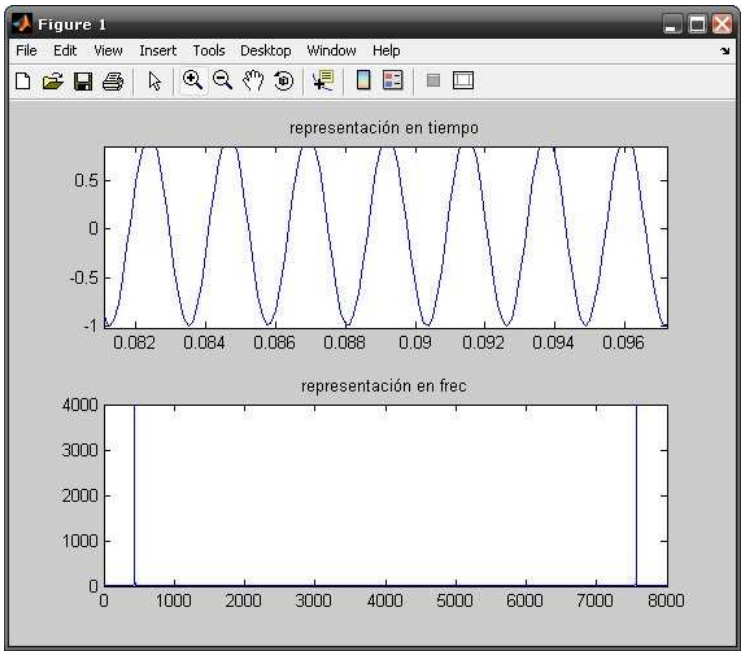


Fig. 3.3.2. Señal distorsionada con a=0;

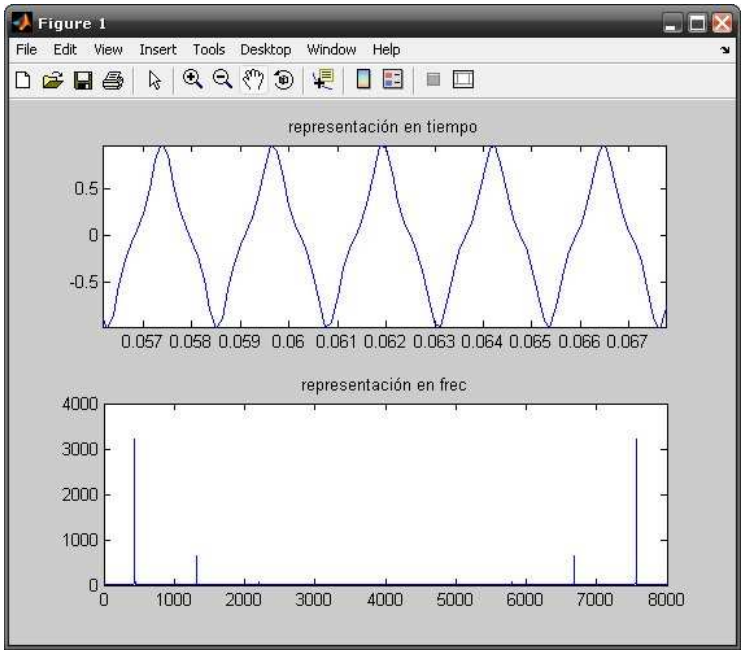


Fig. 3.3.3. Señal distorsionada con a=-0.5;

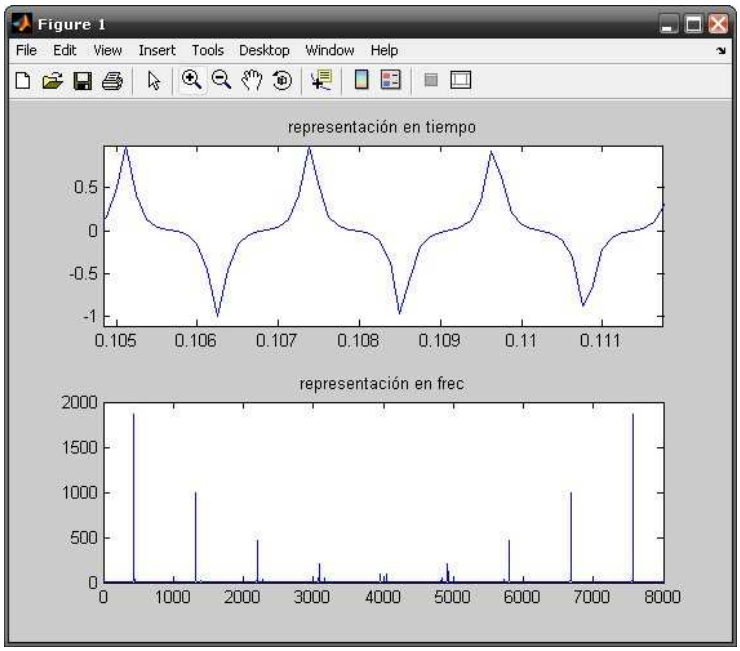


Fig. 3.3.4. Señal distorsionada con a=-0.9;

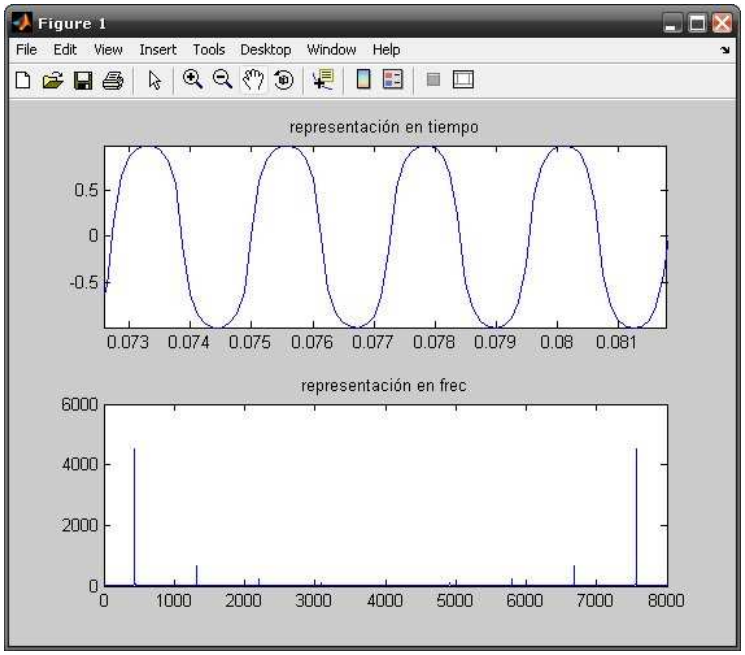


Fig. 3.3.5. Señal distorsionada con a=0.5;

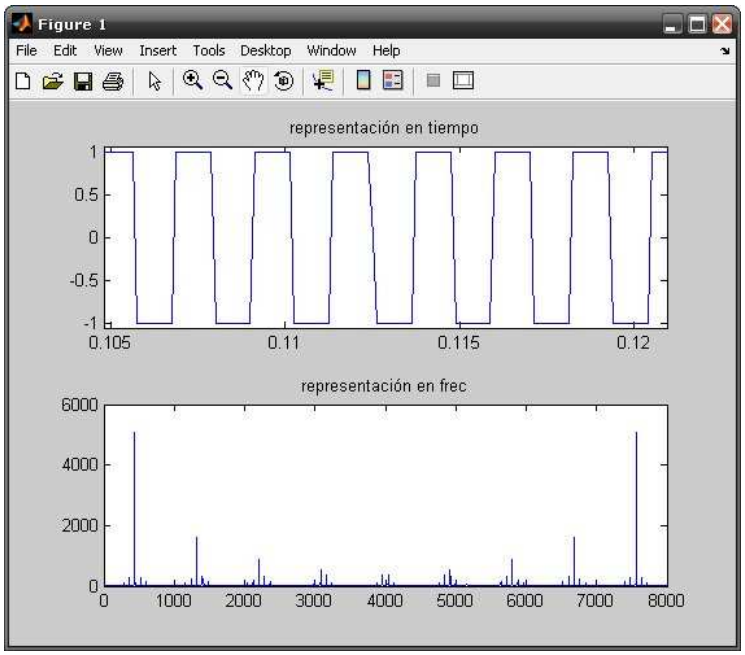


Fig. 3.3.6. Señal distorsionada con a=0.9999;

Las figuras demuestran que conforme aumenta la ganancia *a* la deformación de la onda es mayor y a su vez se introducen un mayor número de armónicos en el espectro frecuencial. Por este motivo se puede decir que la distorsión armónica que se quería conseguir se ha implementado correctamente.

Es necesario comentar que en la implementación de la distorsón se ha escogido un valor fijo de ganancia. De esta forma el usuario sintetizará su sonido indicando si quiere aplicarle distorsión o no. El valor que se ha elegido es *a=0.9999* ya que con este valor la distorsión es más exagerada siendo más destacable dicho efecto.

3.4 La percusión

Además de los instrumentos guitarra y sintetizador ya implementados se ha decidido completar el trabajo con un tercer instrumento llamado percusión. Esta nueva pista intentará imitar los sonidos percutidos de una caja y un bombo. De esta forma la composición musical compuesta por la aplicación será lo más completa posible teniendo un instrumento que marque el ritmo de la canción. Para poder sintetizar estos dos nuevos instrumentos antes se deberán analizar e investigar sobre cómo funcionan y reproducen su sonido.

Los instrumentos de percusión son aquellos que deben ser golpeados para producir su sonido. Pueden ser excitados por percusión directa o indirecta. Se trata de la familia más heterogénea de la orquesta, pues aunque estos instrumentos son bastante parecidos en cuanto a su manejo, su forma es muy variada. Para estudiarlos es

conveniente desdoblarlos en dos grandes grupos según puedan o no ser afinados. En el primen grupo estarían los instrumentos de sonido determinado o melódicos los cuales son capaces de reproducir melodías, es decir, pueden afinarse (Xilófono, marimba, campanas tubulares, etc.). El segundo grupo lo forman los instrumentos de sonido indeterminado, o no melódicos (bombo, platillos, caja, triángulo, etc.). Estos instrumentos no son afinables y su sonido es indeterminado y concreto, es decir, el sonido siempre es el mismo y sólo admite la gradación dinámica. En líneas generales, puede decirse que la función musical de los instrumentos de percusión es marcar el ritmo de una composición musical.

Es interesante conocer un poco más a fondo la física de los instrumentos de este segundo grupo que es donde se sitúan los instrumentos que vamos a sintetizar. Este grupo se caracteriza por producir su sonido con membranas o placas. Las placas y membranas son cuerpos de superficie grande con relación a su espesor; excitadas por percusión o fricción emiten sonidos caracterizados por un complejo grande de parciales discordantes. Las placas, debido a su rigidez, sólo necesitan un punto de apoyo, mientras que las membranas necesitan tensión previa para vibrar. Los instrumentos bombo y caja son dos instrumentos de membrana así que el estudio de las placas no es necesario.

El físico alemán Florencio Chladni realizó profundos estudios sobre las vibraciones de membranas y descubrió que en estos cuerpos no existen nodos y vientres propiamente dichos, sino líneas de puntos donde la vibración es nula o pequeña, llamadas líneas nodales, y zonas demarcadas por estas líneas donde la vibración alcanza valores máximos llamadas zonas ventrales. Chladni formuló dos leyes referentes a las vibraciones en membranas:

1. La frecuencia de dos membranas de igual superficie es inversamente proporcional a su espesor
2. La frecuencia de dos membranas de idéntico espesor varía inversamente al cuadrado de su diámetro

Como curiosidad, en la siguiente figura aparecen representados los doce primeros modos de vibración de una membrana circular ideal de un timbal, en orden creciente de frecuencias. Para denominarlos se utiliza una notación compuesta por dos dígitos: con el primero se indica el número de nodos diametrales y con el segundo el número de nodos circulares. . En el modo fundamental (0,1) toda la membrana se mueve en fase. La frecuencia de vibración se expresa como múltiplo de la del modo fundamental y aparece debajo de cada diagrama particular. La secuencia no forma una serie armónica.

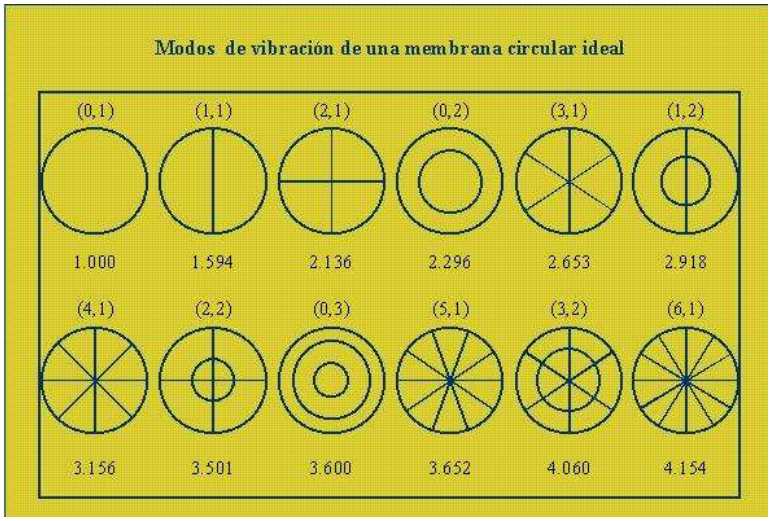


Fig. 3.4.1. Modos de vibración de una membrana circular de un timbal

De este tipo de intrumentos de percusión cabe destacar algunas de sus características más relevantes. Entre ellas se puede decir que espectralmente este tipo de sonidos no se compone de superposición de senos, si no que emiten ruido aleatorio y no hay un patrón definido en el espectro frecuencial. Es decir, en el caso de la guitarra se sabe que cada armónico dista del anterior y del siguiente un factor ½ de la frecuencial fundamental. En el caso de la percusión no ocurre de esta forma. Otra caracteristica es que estos sonidos mantienen una duración fija siendo imposible variar este parámetro. Además en cuanto su envolvente o curva ADSR se puede decir que tienen un ataque y un sostenimiento muy cortos.

Así pues, conociendo un poco mejor como se caracterizan estos instrumentos se dispone a hacer el análisis y síntesis de los dos instrumentos a implementar, el bombo y la caja.

3.4.1 El bombo

El bombo es un gran tambor con dos parches paralelos. Instrumento de gran poder sonoro que se caracteriza por su peso y su volumen.

Para realizar la síntesis se hace primeramente un análisis frecuencial y temporal de una muestra de un sonido de un bombo obtenido con el software de audio “FL Studio 9”. Este tipo de análisis se hará con MatLab de la misma forma que se ha hecho el análisis de guitarra y se utilizará una función implementada que calcula la transformada de Fourier de la señal. El ejecutar el archivo “analize.m” se obtienen las siguientes figuras:

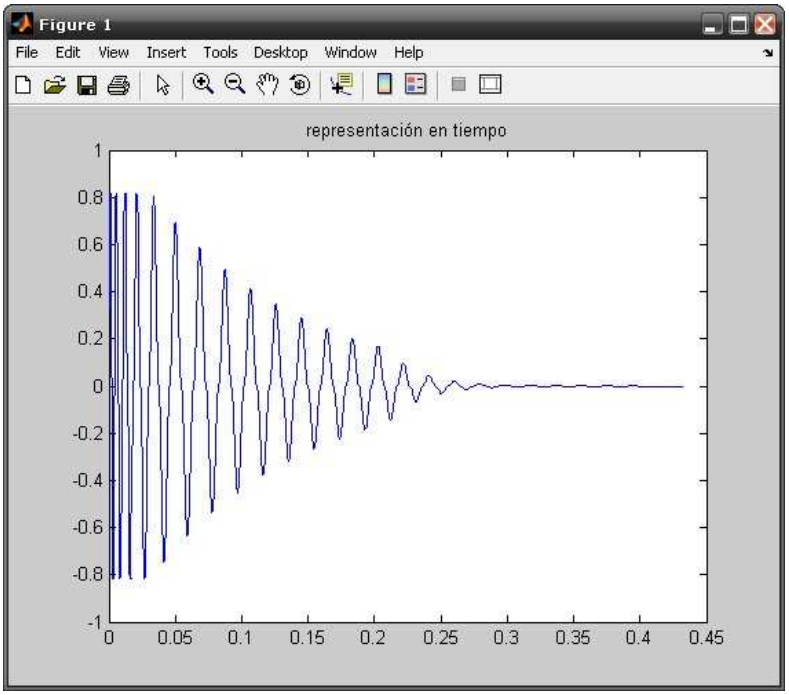


Fig. 3.4.1.1. Análisis temporal del sonido real de un bombo

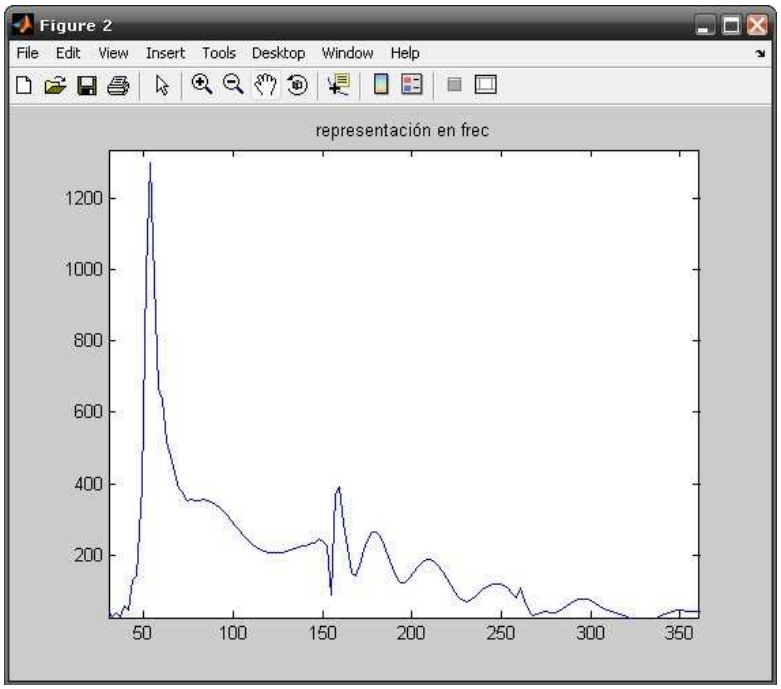


Fig. 3.4.1.2. Análisis frecuencial del sonido real de un bombo

Observando las dos figuras se pueden identificar algunas de sus características para realizar la síntesis. En la figura de la representación temporal se puede ver que la señal es bastante parecida a una señal senoidal que decrece con el tiempo. Además en la

representación en frecuencia se puede distinguir una delta clarísima en 55 Hz además de una pequeña cantidad de ruido apartir de 100 Hz. Así pues la síntesis del bombo se podría hacer perfectamente mediante la ecuación de síntesis del M.A.S.

$$x = A\ sen(2\pi\ f\ t)$$

Se sabe que la frecuencia sería 55 Hz ya que es donde se sitúa la delta del armónico. En cuanto al tiempo se conoce que la duración de este tipo de sonidos es fija, así que se puede tomar como duración 0.25 segundos tal y como se puede ver en la figura 3.4.1.1.

La frecuencia de muestreo debe ser en todo el proyecto por igual. Se ha explicado en el capitulo de la síntesis de guitarra que se toma una frecuencia de muestreo de 22050 Hz debido al criterio de Nyquist.

Teniendo todos estos parámetros se puede sintetizar el bombo con los siguientes comandos en MatLab:

```
fm=22050;
t=0:1/fm:0.25;  % Se crea un vector de tiempo con duración fija
y=3.*sin(2*pi*55*t); % Se crea la señal de bombo mediante la ecuación
de sintesis a una frecuencia de 55Hz
```

Además, la envolvente puede ser caracterizada por una señal exponencial decreciente como se ha hecho en la síntesis de la guitarra. Asi que para finalizar la síntesis del bombo se genera la exponencial y se multiplica por la señal del bombo.

```
 x=exp(-16*t);  % Se genera una señal exponencial decreciente que le
dará la envolvente caracteristica del sonido original
 y=y.*x;    % Se multiplica la señal exponencial por la señal para
darle la envolvente temporal caracteristica
```

En las siguientes figuras se muestra la comparación tanto temporal como frecuencial entre la señal original de muestra de bombo y la señal sintetizada. La señal roja es la señal original y la señal azul la señal sintetizada. Se puede comprobar que apenas existen diferencias. La única diferencia apreciable podría ser que la señal sintetizada tiene un poco más de amplitud debido a que se ha decidido aumentar un poco el volumen con respecto a la original.

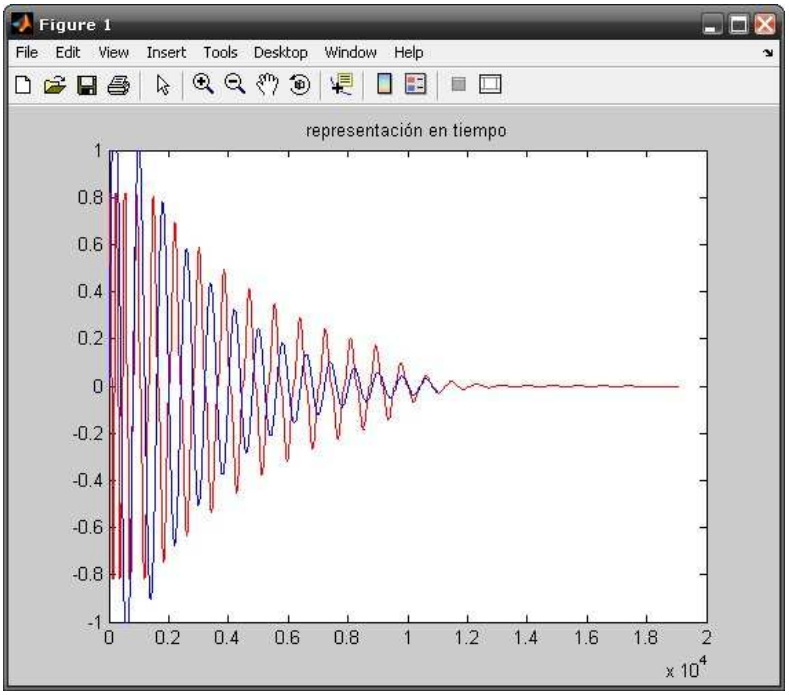


Fig. 3.4.1.3. Comparación temporal entre señales real y sintetizada del sonido de un bombo

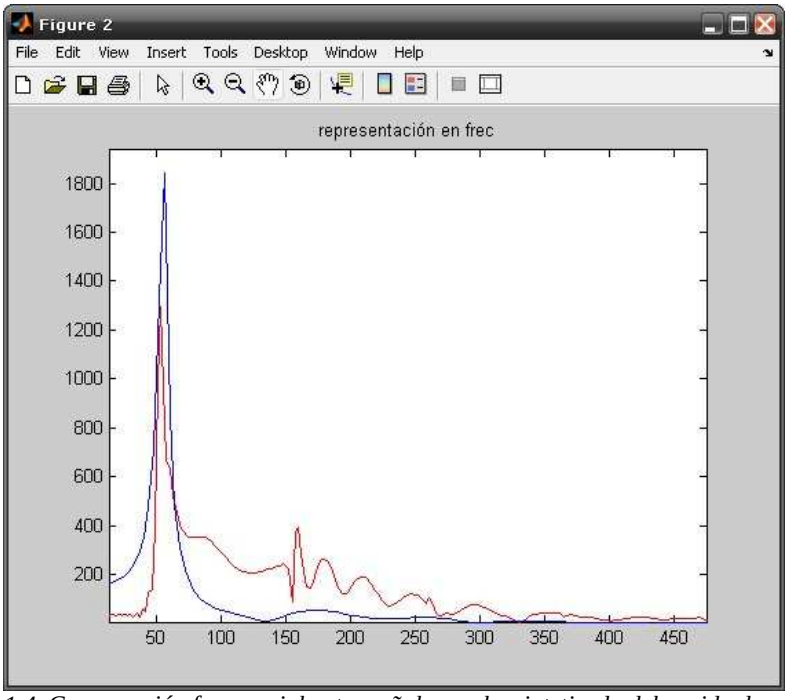


Fig. 3.4.1.4. Comparación frecuencial entre señales real y sintetizada del sonido de un bombo

3.4.2 La caja

La caja es un instrumento típico militar que acompaña a la orquesta en tiempos muy rítmicos y marcados. Es un cilindro de metal con dos membranas paralelas que se conoce vulgarmente como tambor. Dispone de unas cuerdas metálicas llamadas bordones en contacto con la membrana inferior que aumentan su resonancia y hacen menos seco su sonido. Los bordones suelen llevar acoplado un sistema por el que pueden ponerse en contacto o no en contacto con el parche.

De la misma forma que se ha hecho el análisis del resto de instrumentos, se analiza una muestra de audio de una caja obtenida con el programa 'FL Studio9'. Las figuras de la representación en tiempo y frecuencia se muestran a continuación.

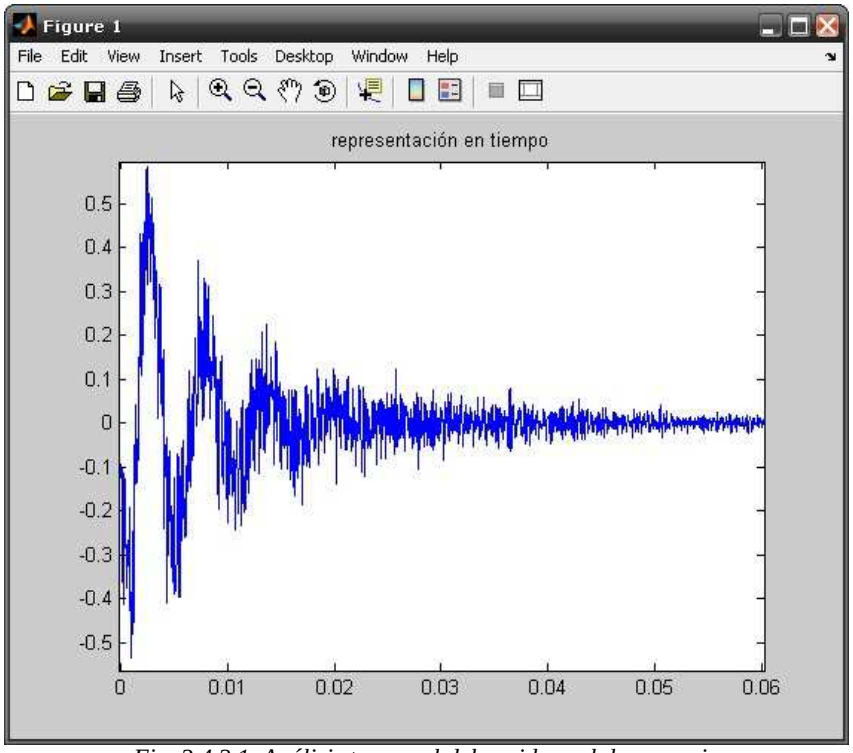


Fig. 3.4.2.1. Análisis temporal del sonido real de una caja

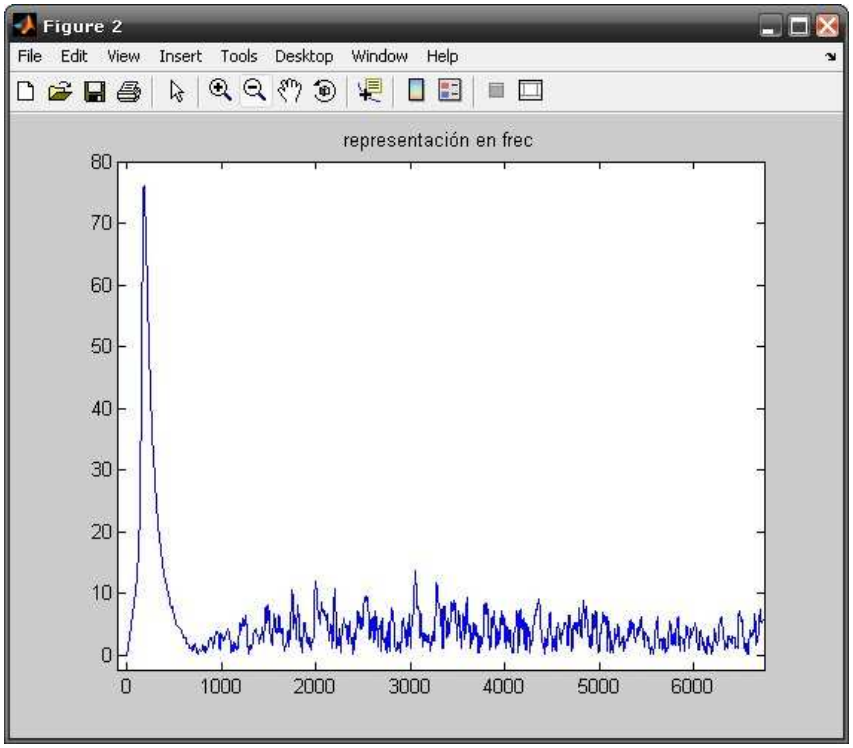


Fig. 3.4.2.2. Análisis frecuencial del sonido real de una caja

De estas dos figuras se puede analizar que hay una gran cantidad de ruido aleatorio superpuesto a una onda senoidal a aproximadamente 180 Hz que viene indicado por la delta de la representación frecuencial. En esta representación se hace notable la característica de que este tipo de instrumentos emiten gran cantidad de ruido sin tener un patrón frecuencial definido.

Así pues para poder imitar el sonido característico de una caja se pueden sumar dos señales distintas. Una de ellas será una señal senoidal de duración 0.05 segundos y frecuencia de 180 Hz y la otra será una señal de la misma duración pero con valores aleatorios que a la hora de reproducirla generará un sonido parecido al ruido blanco. A la suma de estas dos señales se le puede multiplicar una señal exponencial decreciente. De esta forma se puede imitar la envolvente temporal del instrumento que es apreciable en la figura que representa la señal original en función del tiempo.

Los comandos utilizados en Matlab para poder sintetizar este tipo de sonido son los siguientes:

```
        fm=22050;
        t=0:1/fm:0.05;  % Se crea el vector tiempo con duración fija
        y=randn(1,length(t));  % se genera una señal de ruido blando
con la duración del vector tiempo
        s=sin(2*pi*180*t);   % Se genera la onda senoidal a 180 Hz
mediante la ecuacion de sintesis
        y=y+s;  % Se superponen las dos señales para crear el sonido
similar de la caja
        x=exp(-80*t); % Se genera una señal exponencial decreciente
        y=y.*x;  % Se multiplica la señal de la caja por la
exponencial para darle la envolvente caracteristica
```

En las siguientes figuras se muestra una comparación en tiempo y frecuencia de las señales real y sintetizada. La señal de color rojo representa el sonido real y la azul representa la señal sintetizada. Se puede comprobar que la síntesis de este instrumento ha sido satisfactoria ya que las dos señales son bastante parecidas. La señal de color azul, es decir, la sintetizada, tiene un poco más de amplitud debido a que se ha querido darle un poco más de volumen.

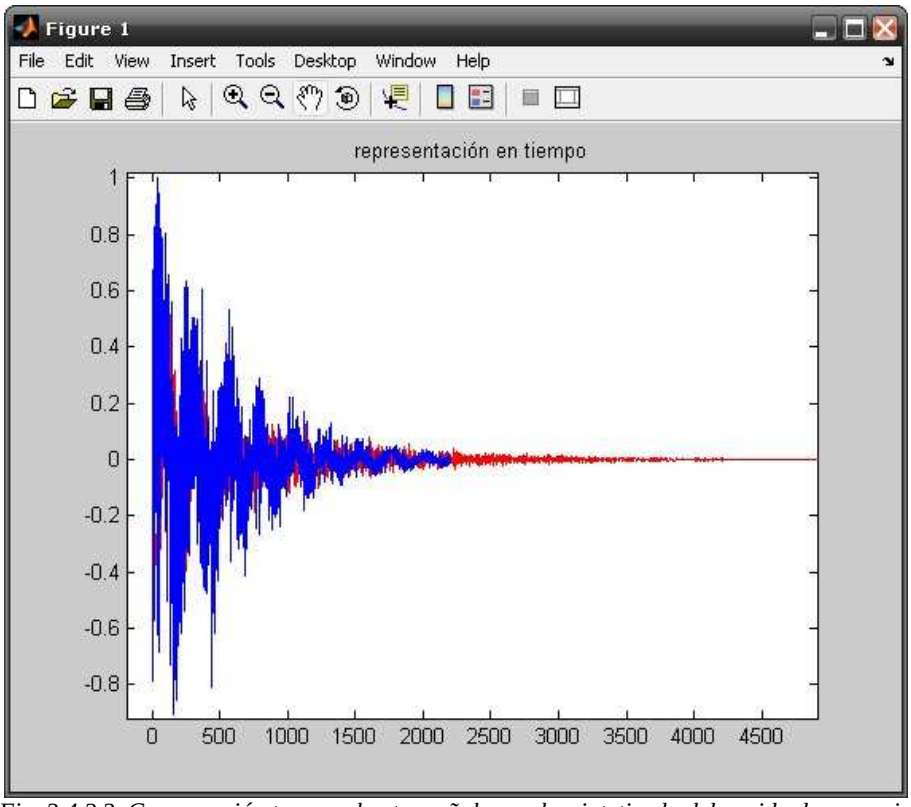


Fig. 3.4.2.3. Comparación temporal entre señales real y sintetizada del sonido de una caja

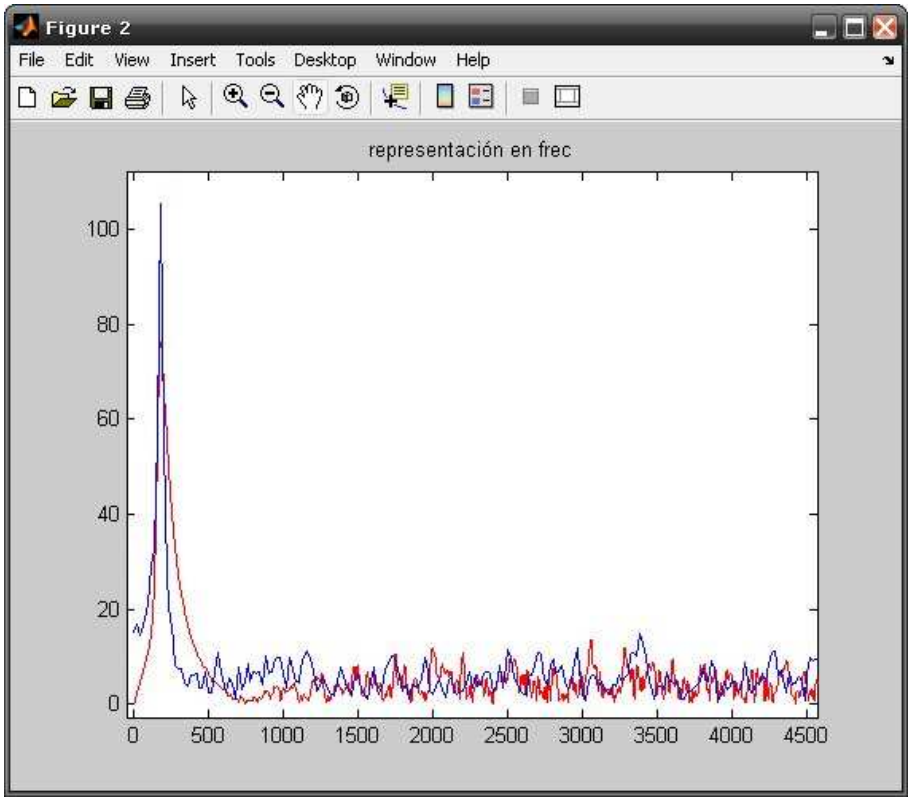


Fig. 3.4.2.4. Comparación frecuencial entre señales real y sintetizada del sonido de una caja

4. Desarrollo de la aplicación “Secuenciador”

Una vez conseguida la síntesis de los instrumentos guitarra, sintetizador y percusión será necesaria una aplicación para poder unificar estos tres instrumentos y poder realizar composiciones musicales sencillas. Esta aplicación constará de una interfaz gráfica con la que el usuario indicará los parámetros a insertar para poder definir su sonido. De esta forma el usuario podrá componer sus piezas musicales de una forma agil, sencilla y amena.

4.1 MatLab y Guide

El entorno de programación elegido para el desarrollo de la interfaz gráfica es mediante MatLab.

MatLab es la abreviatura de *MATrix LABoratory*, "laboratorio de matrices". Es un software matemático que ofrece un entorno de desarrollo integrado con un lenguaje de programación propio (lenguaje M). Está disponible para las plataformas Unix, Windows y Apple Mac OS X.

Entre sus prestaciones básicas se hallan: la manipulación de matrices, la representación de datos y funciones, la implementación de algoritmos, la creación de interfaces de usuario (GUI) y la comunicación con programas en otros lenguajes y con otros dispositivos hardware. Gracias a todas estas prestaciones MatLab se hace el mejor candidato para poder desarrollar este proyecto. Además es un software muy usado en universidades y centros de investigación y desarrollo y en los últimos años ha aumentado el número de prestaciones.

Mediante la herramienta *“Guide”* se pueden desarrollar interfaces gráficas de cualquier tipo y de una forma sencilla y rápida. Es un entorno de programación visual disponible en MatLab para realizar y ejecutar programas que necesiten ingreso continuo de datos. Tiene las características básicas de todos los programas visuales como Visual Basic o Visual C++. Esta será la herramienta utilizada para el desarrollo de la aplicación “*secuenciador.m”* ya que es posible implementar todas aquellas funciones que se necesitan.

En la siguiente figura se muestra el entorno de trabajo de la herramienta *“Guide”*. A la izquierda se sitúa la paleta de componentes que es posible implementar en una interfaz (push button, slider, checkbox, ejes, etc.). Con estos componentes y la variación de sus propiedades es posible desarrollar interfaces bastante complejas. En la parte superior se encuentran algunas de las herramientas para poder trabajar con el archivo que se está creando (abrir, guardar, ejecutar, etc.). Y por último, en la parte central de la ventana es el panel de la plantilla en blanco donde se va a ir componiendo la interfaz.

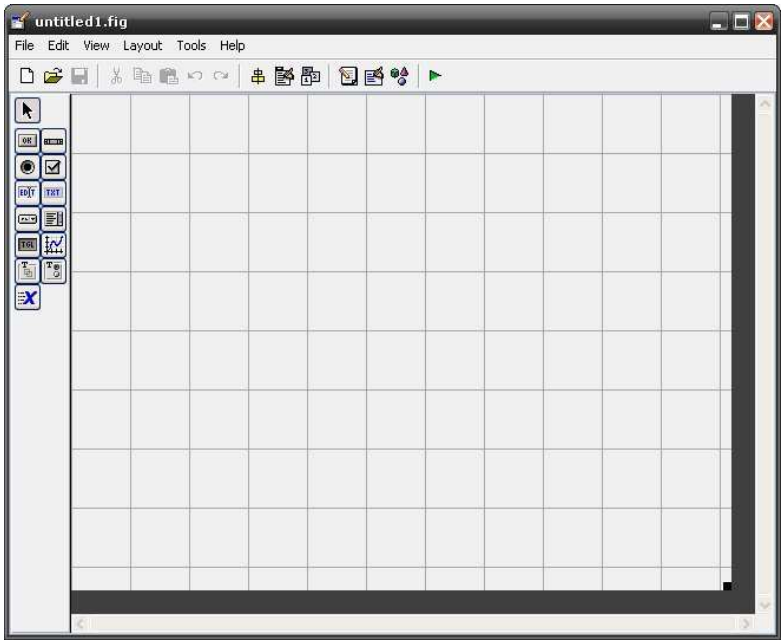


Fig. 4.1.1. Entorno de trabajo de "Guide"

Una aplicación consta de dos archivos: *.m* y *.fig*. El fichero *.fig* hace referencia a la consola de edición de la parte gráfica de la aplicación a implementar, es decir, permite diseñar los elementos que formarán la interfaz. En cambio el archivo *.m* es el ejecutable de Matlab donde se determinan las subrutinas que van asociadas a cada elemento creado en el .fig. Permite determinar que pasará al pulsar cualquiera de los botones. Mediante esta herramienta *"Guide"* se crean estos dos tipos de archivos estando estos dos asociados entre ellos.

4.2 Intención y explicación de la interfaz

Para poder desarrollar una interfaz gráfica con éxito es necesario tener claro desde un principio la intención del propio programa. El objetivo de la aplicación "Secuenciador" es que el usuario pueda componer sus piezas musicales indicando los paramentros de síntesis de cada instrumento. Por este motivo se hace necesario construir un total de cuatro interfaces, una interfaz para la composición de cada instrumento, y una general que actuará a modo de mezclador de los tres instrumentos y contendrá algunas de las herramientas generales a las tres pistas.

4.2.1 Secuenciador

La interfaz perteneciente al secuenciador dispondrá de todas aquellas herramientas generales y comunes a las tres pistas de los instrumentos. Además actúa a modo de mezclador, por lo tanto también contiene funciones de volumen de cada una de las pistas. Mediante esta ventana se pueden acceder a las ventanas particulares de cada instrumento.

Todo aquello que se componga también se representa en los ejes de esta interfaz representándose cada instrumento en su particular eje. La representación vendrá definida por unas barras de la misma duración que la nota que representan. Además, cabe destacar, que estos ejes estan linkados de forma que siempre representan la misma escala temporal y siempre estan sincronizados en el tiempo (eje x).

Para que la explicación quede más clara y detallada, a continuación se muestra la figura de la propia interfaz y seguidamente se explican cada una de las funciones y herramientas que esta contiene.

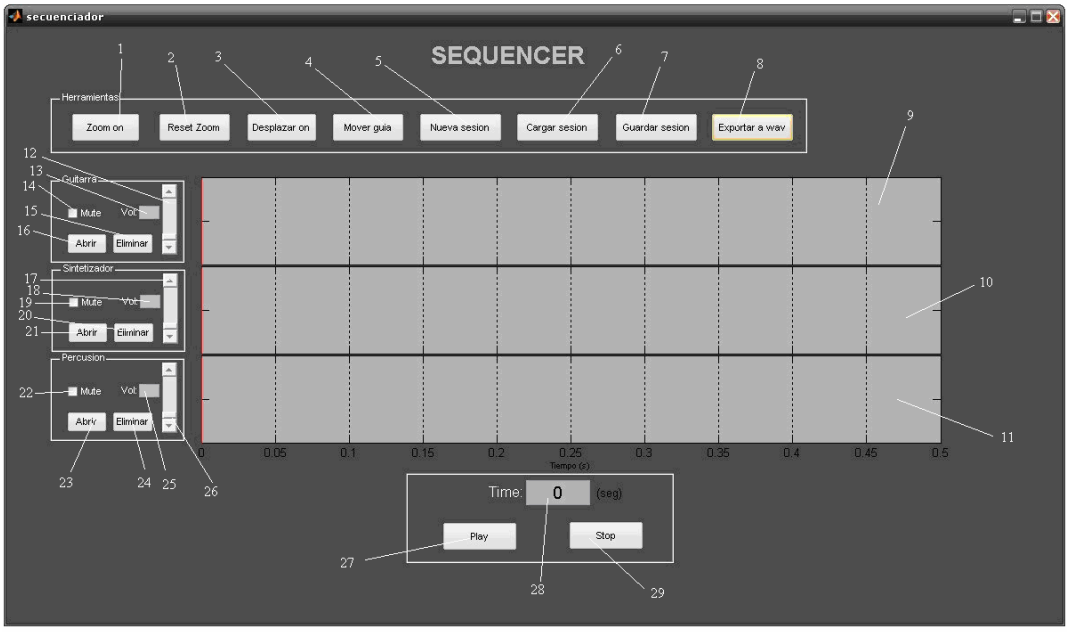


Fig. 4.2.1.1. Ventana de la aplicación "secuenciador.m"

1. Zoom on: Amplia los ejes posicionandose encima de ellos. Para que esta función deje de estar activa es necesario volver a pulsar para desactivar la ampliación de zoom.
2. Reset Zoom: Vuelve a posicionar los ejes en vista minima pudiendo visualizar toda la señal en función del tiempo.
3. Desplazar on: Aparece una mano al posicionarse en cualquiera de los ejes pudiendo arrastrarlo y moverte sobre él tanto en eje x como y. Será necesario volver a pulsar para desactivar esta opción.
4. Mover guia: Sitúa una guia en el instante que el usuario quiera para poder empezar la reproducción desde ahí. Inicialmente la guia se sitúa en cero.
5. Nueva sesión: Inicializa una nueva sesión poniendo todo a cero.
6. Cargar sesión: Con esta herramienta se podrán recuperar sesiones previamente guardadas en formato .mat.
7. Guardar sesión: Guarda la sesión de la composición musical con el nombre que el usuario indique en formato .mat.

8. Exportar a wav: Guardar la composición creada en formato de audio .wav
9. Eje guitarra: Es el eje donde se representa la composición perteneciente a la pista de guitarra en función del tiempo y mediante unas barras con la duración de cada nota.
10. Eje sintetizador. Es el eje donde se representa la composición perteneciente a la pista de sintetizador en función del tiempo y mediante unas barras con la duración de cada nota.
11. Eje percusión: Es el eje donde se representa la composición perteneciente a la pista de percusión en función del tiempo y mediante unas barras con la duración de cada instrumento. Se representa tanto la caja como el bombo.
12. Slider guitarra: Con este slider es posible cambiar el volumen de la pista guitarra.
13. Volumen guitarra: En esta casilla viene indicado el volumen de la guitarra asociado al slider que controla el volumen de la pista
14. Mute guitarra: Silencia la pista de guitarra al reproducirla.
15. Eliminar guitarra: Elimina e inicializa la pista de guitarra
16. Abrir guitarra: Abre la interfaz guitarra.m con la que se puede componer el instrumento guitarra
17. Slider sintetizador: Controla el volumen de la pista sintetizador.
18. Volumen sintetizador: Indica el volumen de la pista de sintetizador.
19. Mute sintetizador: Silencia la pista sintetizador al reproducirla.
20. Eliminar sintetizador: Elimina e inicializa la pista de sintetizador
21. Abrir sintetizador: Abre la interfaz sintetizador.m con la que se puede componer el instrumento sintetizador.
22. Mute percusión: Silencia la pista de percusión al reproducirla.
23. Abrir percusión: Abre la interfaz percusión.m con la que se puede componer el instrumento de percusión.
24. Eliminar percusión: Elimina e inicializa la pista de percusión.
25. Volumen percusión: Indica el volumen asociado a la pista de percusión.
26. Slider percusión: Controla el volumen de la pista percusión.
27. Play: Reproduce las tres pistas simultaneamente dependiendo de cada uno de sus volúmenes y mutes
28. Time: Indica la posición en segundos de donde se sitúa la guía.
29. Stop: Detiene la reproducción comenzada.

4.2.2 Guitarra

Como se ha explicado previamente cada instrumento tiene una interfaz que es llamada desde la interfaz general secuenciador.m. Para la composición de la guitarra el usuario tan solo tiene que indicar la duración, el instante y la nota a insertar. También se tiene la opción de posicionar una nota indicada dándole un instante inicial y otro final. A medida que se van introduciendo notas, estas se ven representadas en función del tiempo mediante unas barras de la misma duración que la nota.

A continuación se muestra una figura de la propia ventana de la interfaz gráfica y seguidamente se explican algunas de las herramientas que la componen.

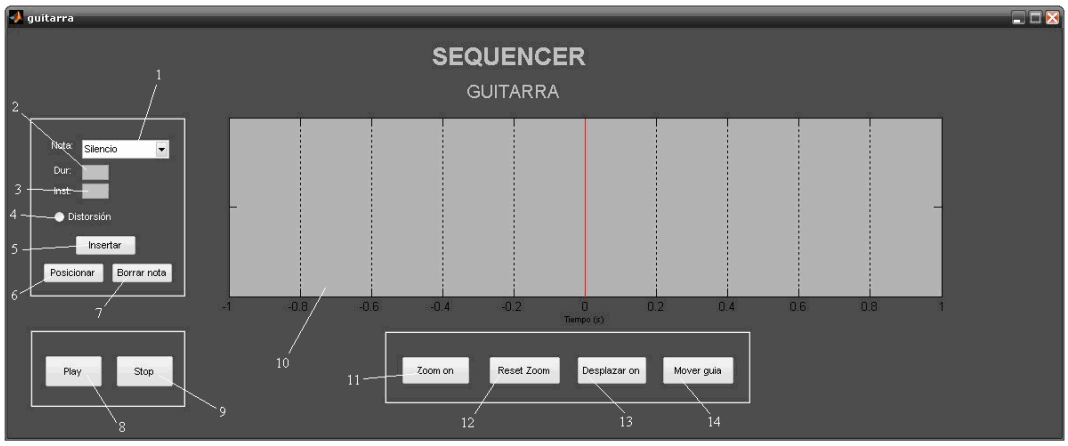


Fig. 4.2.2.1. Ventana de la aplicación "guitarra.m"

1. Selector de notas: Mediante un menu desplegable se puede seleccionar le nota que se se desea insertar, desde Mi3 hasta Mi6
2. Duración: En esta casilla se indica la duración de la nota que se quiere insertar.
3. Instante: Se indica el instante concreto en segundos donde se desea insertar la nota. En el caso que no se indique ningún valor la nota se insertará detrás de la última.
4. Distorsión: Si selecciona esta opción la nota que se inserte tendrá el efecto característico de la distorsión armónica.
5. Insertar: Inserta la nota que se le haya indicado con su duración y el instante determinado.
6. Posicionar: Posiciona la nota que esté seleccionada en el menu, indicándole un instante inicial y otro final seleccionandolos con el ratón en el propio gráfico.
7. Borrar nota: Borra la nota que se le indique con el ratón en el gráfico.
8. Play: Reproduce la pista de guitarra desde el instante donde esté posicionada la guia.
9. Stop: Detiene la reproducción.
10. Eje guitarra: Es el gráfico donde se van representando las notas que se añaden a la propia pista de guitarra. Cada nota se representa mediante una barra de la misma duración que la nota.

El resto de herramientas no es necesaria su explicación ya que realizan la misma función que en el resto de interfaces creadas *(Véase la explicación en el apartado 4.2.1).*

4.2.3 Sintetizador

La interfaz del sintetizador prácticamente es igual a la de guitarra teniendo como diferencia que en este instrumento también será necesario insertar la octava de la nota a sintetizar. Esto se debe a que este instrumento está diseñado para que sea capaz de reproducir las 8 octavas del piano. De esta forma, en esta interfaz gráfica, se dipone de dos menús desplegables:

- Selector de nota: En el cual se puede seleccionar la nota a insertar independientemente de la octava (desde Do hasta Si).
- Selector de octava: En este menú desplegable se puede seleccionar una de las 8 octavas que puede reproducir este instrumento.

Combinando estos dos selectores se obtiene la nota final a sintetizar. Por ejemplo: Si5, La3, Do4, etc.

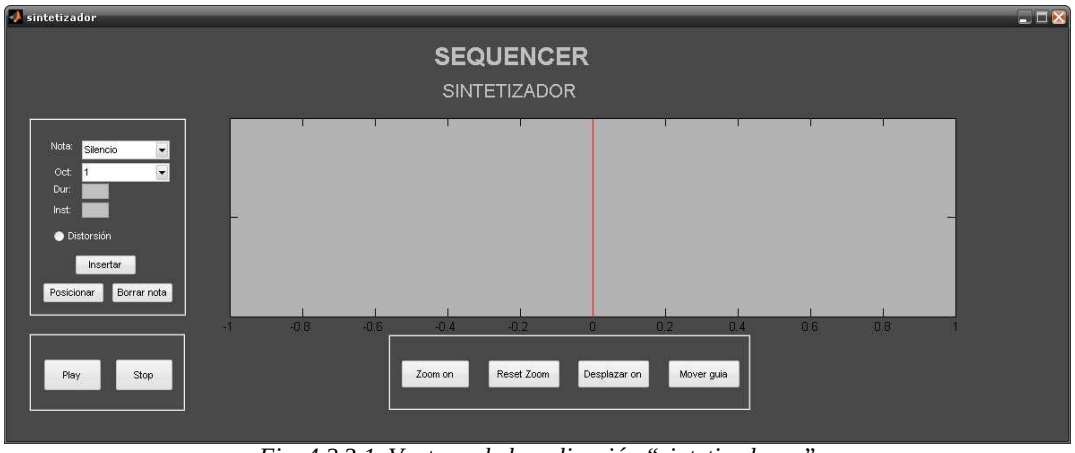


Fig. 4.2.3.1. Ventana de la aplicación "sintetizador.m"

El resto de herramientas no es necesaria su explicación ya que realizan la misma función que en el resto de interfaces creadas *(Véase la explicación en el apartado 4.2.1).*

4.2.4 Percusión

En cuanto a la interfaz de percusión si que se puede decir que difiere en algunos aspectos con las otras dos. En este caso, la composición de esta pista no viene definida mediante notas musicales, si no que está compuesta por los instrumentos caja y bombo y silencios. Así pues, el usuario para componer esta pista indicará el instrumento y el instante donde desea insertarlo. También se dispone de las opciones de posicionar un instrumento indicando en el gráfico la posición inicial y la posibilidad de borrar un intrumento ya sintetizado. En el caso que se desee insertar un silencio se debera indicar su duración.

A continuación se muestra una figura de la propia ventana de la interfaz gráfica y seguidamente se explican algunas de las herramientas que la componen.

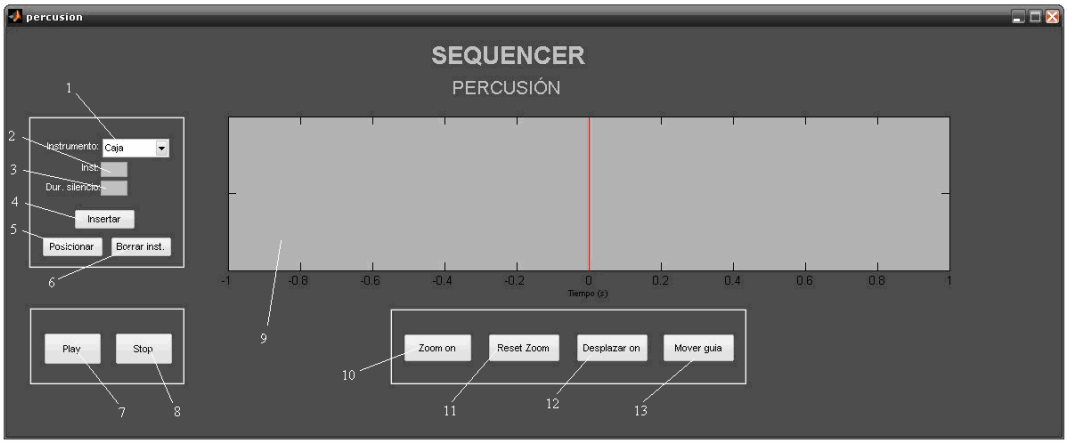


Fig. 4.2.4.1. Ventana de la aplicación "percusion.m"

1. Selector de instrumentos: Se selecciona que intrumento se quiere insertar (caja, bombo o silencio).
2. Instante: Se indica el instante concreto en segundos donde se desea insertar el instrumento seleccionado. En el caso que no se indique ningún valor el instrumento se inserta detrás del último ya insertado.
3. Duración del silencio: Se indica la duración del silencio en segundos. Sólo funciona si en el selector se ha escogido "silencio".
4. Insertar: Inserta el instrumento seleccionado en el instante indicado
5. Posicionar: Posiciona el instrumento que esté seleccionado indicándole el instante inicial con el ratón en el gráfico.
6. Borrar instrumento: Borra aquel instrumento que se seleccione con el ratón en el gráfico.

El resto de herramientas no es necesaria su explicación ya que realizan la misma función que en el resto de interfaces creadas *(Véase la explicación en el apartado 4.2.1).*

4.3 Aspectos de programación

Una vez creadas las cuatro interfaces gráficas, se ha de realizar la tarea de programar cada uno de los ficheros .m. Estos ficheros se generan automáticamente al guardar los ficheros .fig de las interfaces gráficas.

En los anexos se incluyen cada uno de los códigos de programación de los ficheros *secuenciador.m, guitarra.m, sintetizador.m* y *percusión.m* debidamente comentados y explicados. En estos ficheros se puede ver como funcionan y como se implementan cada una de las subrutinas.

De todas formas, se hace necesario explicar el aspecto más importante de la programación de esta aplicación. Este aspecto es cómo se va generando la pista de cada instrumento y se van sumando las distintas notas que se van insertando.

Cada pista viene inicializada como una matriz vacía. El número de columnas corresponde al número de muestras de la pista del instrumento, mientras que el número de filas corresponde al número de notas insertadas. A medida que se van insertando notas, la matriz va aumentando su número de filas posicionando cada nota en su fila correspondiente. La última fila de la matriz corresponderá con la última nota insertada.

El motivo de definir las pistas de cada instrumento como matrices y no como vectores se debe a que, de esta forma es posible posicionar dos notas en el mismo intante temporal pudiendo reproducirlas simultaneamente. Con esta ventaja se hace posible la síntesis de acordes musicales que están compuestos por más de una nota a la vez.

Además, gracias a definir estas variables como matrices es más comodo y sencillo implementar la función de eliminar una nota. Simplemente, seleccionando un punto en el gráfico se asociará dicho punto con una fila de la matriz. De este modo, tan solo se tendrá que eliminar la fila seleccionada de la matriz borrándose tambien la nota que contenía.

A la vez que se van generando las matrices de cada instrumento también se generan unos vectores de texto que contienen el nombre de cada nota insertada o, en el caso de la percusión, el instrumento insertado. Al igual que ocurre con las matrices de los instrumentos, el nombre de la última nota insertada se posicionará en la última posición del vector. Estos vectores de texto se hacen indispensables a la hora de la representación gráfica de cada instrumento, indicando en cada nota insertada el nombre de la nota que le corresponde. Gracias a esto la visualización de la representación gráfica será mucho más clara y concisa.

Por último comentar que estas variables, tanto las matrices como los vectores, tendrán que ser definidas en la programación como variables globales. Esto es porque son utilizadas en diversas interfaces compartiéndose los datos que contienen.

A continuación se muestra una imagen de la aplicación *"secuenciador.m"* en funcionamiento donde se puede ver cómo se representan las notas mediante barras y respetando el orden de las filas donde están insertadas.

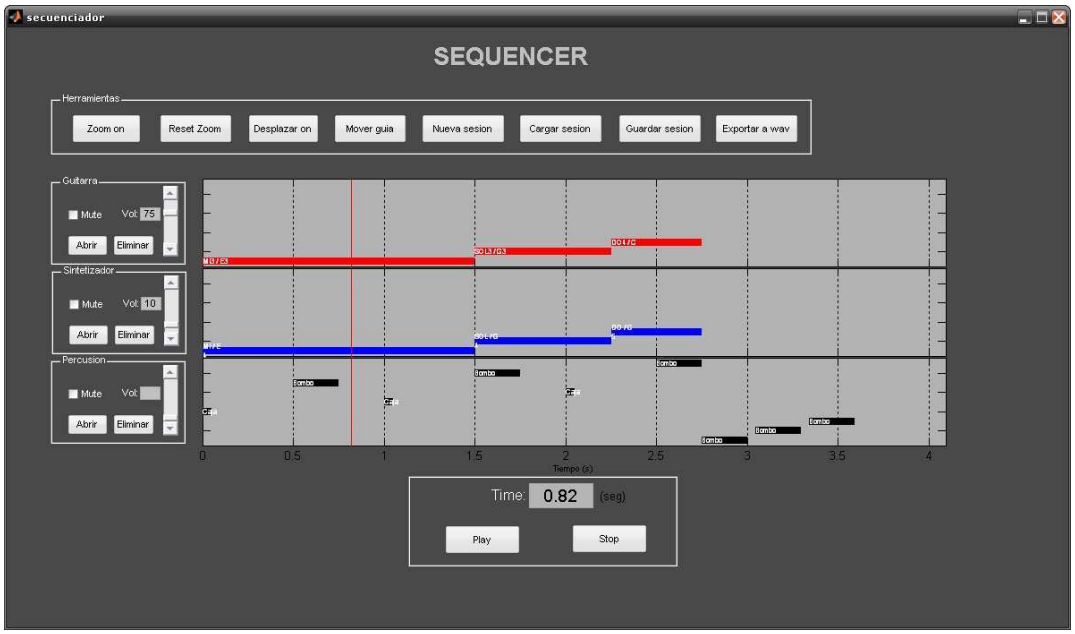


Fig. 4.3.1. Ejemplo de ejecución de la aplicación "secuenciador.m"

5. Conclusiones

Como se ha visto en el transcurso del presente proyecto final de carrera, se ha diseñado y desarrollado, tal y como marcaba el objetivo principal, una aplicación en Matlab que permite el secuenciado musical, de forma similar al secuenciado utilizado en los ficheros de audio MIDI.

Mediante la composición de una pieza musical de prueba llamada "bamba.mat" que se puede cargar mediante la herramienta *"Cargar sesión"*, se puede demostrar que los objetivos del proyecto han sido cubiertos y que los resultados obtenidos son más que satisfactorios. Entre ellos se puede destacar que el sonido sintetizado de la guitarra es bastante fiel al sonido real y su espectro en frecuencia tambíen es comparable al espectro del sonido real. Además, en cuanto a la percusión, ocurre lo mismo que con la guitarra. El sonido y el espectro en frecuencia del sonido sintetizado son bastante similares al sonido real.

Por otro lado, se ha creado la interfaz gráfica la cual resulta bastante clara y sencilla a la hora de su utilización. De esta forma las composiciones serán menos costosas y más amenas para el usuario. Además, esta interfaz incluye prácticamente todas las funciones comunes a los secuenciadores actuales (control de volumen, mute, desplazamiento entre ejes, etc.). Al poder compararlo con las funciones de los secuenciadores actuales, todavía hace que el desarrollo e implementación de está aplicación sea más satisfactoria.

Sin embargo, la gran mayoría de secuenciadores que existen en el mercado (Cubase, Pro Tools, Guitar Pro, etc.) pueden sintetizar un mayor número de instrumentos. Además también tienen la posibilidad de insertar dispositivos externos a través del protocolo MIDI lo que hace que estos secuenciadores sean mucho más funcionales y prácticos.

De todas formas la aplicación implementada deja abierta la posibilidad de realizar nuevas mejoras. Por un lado se podrían implementar y sintetizar un mayor número de instrumentos. La forma de hacerlo sería siguiendo el método utilizado en este proyecto mediante síntesis aditiva, o bien buscando la posibilidad de generar distintos tipos de timbre mediante otros metodos de síntesis como podrian ser la síntesis FM o la síntesis por tabla de ondas. Por otro lado se podría realizar la programación de la interfaz en otro tipo de lenguaje como el Visual Basic o Java. Así se podrían ampliar algunas funciones y desarrollar otras nuevas, además de que no sería necesario disponer de un programa madre para ejecutar la aplicación.

En conclusión, y como resumen, se puede decir que en lineas generales el proyecto final de carrera ha sido productivo, ya que se han recordado conocimientos cursados en la carrera de asignaturas como física, analisis de sistemas discretos, audio digital o acústica. Por otra parte también se han adquirido conocimientos nuevos los cuales pueden servir para un futuro.

6. Bibliografía y recursos

- PFC “Conversor Wave a Midi en tiempo real para guitarras eléctricas”. *Alberto Molina Reverte. 2006*

- PFC “Diseño de software en MatLab para el análisis de sonidos musicales”. *Miguel Ángel Gónzalez Hernández. 2005.*

- “Fundamentos de síntesis de audio con frecuencia modulada”. *Juan Reyes, Center for Computer Research in Music and Acoustics Stanford, California, EUA.*

- Master of Science Thesis Project **“Implementation and analysis of pitch tracking algorithms”**. *Stefan Upgard, departamento de Señales, Sensores y Sistemas de KTH, Estocolmo, Suecia*

- Apuntes de asignaturas:

    - Fundamentos físicos de la ingeniería

    - Análisis de sistemas discretos

    - Audio digital

    - Fundamentos de acústica

- Páginas web:

    - www.matpic.com

    - www.mathworks.es

    - www.mathkb.com

    - www.ingenierosderadio.com

    - www.pcpaudio.com

    - www.rocoblog.blogspot.com

    - www.musicdsp.org

7. Anexos

7.1 Código secuenciador.m

```
function varargout = secuenciador(varargin)
%
%    Ejecutando el archivo secuenciador.m se abre
%    interfaz gráfica correspondiente al fichero secuenciador.fig.
%    Esta ventana hace las funciones de mezclador de las tres pistas y
%    contiene aquellas herramientas generales que pueden ser asociadas
%    a las tres pistas conjuntas.
%
%    Ademas mediante esta ventana se puede acceder a las interfaces
%    pertenecientes a cada instrumento (guitarra.m, sintetizador.m y
%    percusion.m) para poder componer cada pista por separado.
%
%    Ver tambien: guitarra.m, sintetizador.m, percusion.m

%  ------------------------------------------------------------------------


% Comienza el código de inicialización. Este código es programado
% automáticamente por la aplicación Guide -- NO EDITABLE

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @secuenciador_OpeningFcn, ...
    'gui_OutputFcn',  @secuenciador_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
    'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% Fin del código de inicialización -- NO EDITABLE


%  ------------------------------------------------------------------------
function secuenciador_OpeningFcn(hObject, eventdata, handles,
varargin)

% INICIALIZACIONES DE VARIABLES GLOBALES
global matrizguit etiquetaNota_guit matrizsinte etiquetaNota_sinte
etiquetaOctava_sinte matrizpercu etiquetaInstrumento

% INICIALIZACIONES DE MATRICES DE INTRUMENTOS Y ETIQUETAS
matrizguit=[];
etiquetaNota_guit={};
matrizsinte=[];
etiquetaNota_sinte={};
```

```
etiquetaOctava_sinte={};
matrizpercu=[];
etiquetaInstrumento={};

% --------------------------------------------------------------------
% INICIALIZACIONES DE VARIABLES HANDLES PARA PODER UTILIZARLAS EN
% DIVERSAS FUNCIONES
handles.posguia=0;  % Se posiciona la guia en posicion inicial 0

% Inicialización de los volumenes iniciales de cada intrumento
handles.volumen_guit=0.5;
handles.volumen_sinte=0.5;
handles.volumen_percu=0.5;
% --------------------------------------------------------------------

linkaxes; % Se linkan los ejes para que haya un sincronismo entre ejes

representar_Callback(hObject, eventdata, handles)


% --------------------------------------------------------------------


handles.output = hObject;


guidata(hObject, handles); % Actualiza los manejadores handles


% --------------------------------------------------------------------
function varargout = secuenciador_OutputFcn(hObject, eventdata,
handles)

varargout{1} = handles.output;


% --------------------------------------------------------------------
function mute_guit_Callback(hObject, eventdata, handles)
% Inicializa la funcion del mute de guitarra


% --------------------------------------------------------------------
function abrir_guit_Callback(hObject, eventdata, handles)
%  Al presionar el boton abrir_guit se abrirá la interfaz guitarra.m

guitarra  % Abre la interfaz guitarra.m
uiwait
representar_Callback(hObject, eventdata, handles)


% --------------------------------------------------------------------
function eliminar_guit_Callback(hObject, eventdata, handles)
%  Al presionar el boton eliminar pondrá a cero la matriz de la
guitarra

global matrizguit
matrizguit=[];
```

```
representar_Callback(hObject, eventdata, handles) % Llama a la función
representar

% --------------------------------------------------------------------
function volslide_guit_Callback(hObject, eventdata, handles)
% Inicializa el slider con el que se puede controlar el volumen de la
% guitarra

handles.volumen_guit=get(handles.volslide_guit,'Value');
set(handles.volnum_guit,'String',handles.volumen_guit)
handles.volumen_guit=handles.volumen_guit/100;
guidata(hObject, handles);

% --------------------------------------------------------------------
function volnum_guit_Callback(hObject, eventdata, handles)
%  Inicializa la casilla de texto donde se indicará el numero de
volumen de guitarra



% --------------------------------------------------------------------
function mute_sinte_Callback(hObject, eventdata, handles)
% Inicializa la función de mute del sintetizador


% --------------------------------------------------------------------
function abrir_sinte_Callback(hObject, eventdata, handles)
% Al presionar el boton de abrir_sinte se abrirá la interfaz
sintetizador.m

sintetizador
uiwait
representar_Callback(hObject, eventdata, handles)


% --------------------------------------------------------------------
function eliminar_sinte_Callback(hObject, eventdata, handles)
% Al presionar el boton de eliminar el sintetizador eliminara la
matriz del sintetizador

global matrizsinte
matrizsinte=[];
representar_Callback(hObject, eventdata, handles)

% --------------------------------------------------------------------
function volslide_sinte_Callback(hObject, eventdata, handles)
% Inicializa el slider con el que se puede controlar el volumen del
sintetizador.


handles.volumen_sinte=get(handles.volslide_sinte,'Value');
set(handles.volnum_sinte,'String',handles.volumen_sinte)
handles.volumen_sinte=handles.volumen_sinte/100;
guidata(hObject, handles);


% --------------------------------------------------------------------
function volnum_sinte_Callback(hObject, eventdata, handles)
```

```
%  Inicializa la casilla de texto donde se indicará el numero de
volumen del sintetizador

% --------------------------------------------------------------------
function mute_percu_Callback(hObject, eventdata, handles)
% Inicializa la función de mute del percusion


% --------------------------------------------------------------------
function abrir_percu_Callback(hObject, eventdata, handles)
% Al presionar el boton de abrir_percu se abrirá la interfaz
percusion.m

percusion
uiwait
representar_Callback(hObject, eventdata, handles)


% --------------------------------------------------------------------
function eliminar_percu_Callback(hObject, eventdata, handles)
% Al presionar el boton de eliminar la percusion eliminara la matriz
de la percusion

global matrizpercu
matrizpercu=[];
representar_Callback(hObject, eventdata, handles)


% --------------------------------------------------------------------
function volslide_percu_Callback(hObject, eventdata, handles)
% Inicializa el slider con el que se puede controlar el volumen de la
% percusion

handles.volumen_percu=get(handles.volslide_percu,'Value');
set(handles.volnum_percu,'String',handles.volumen_percu)
handles.volumen_percu=handles.volumen_percu/100;
guidata(hObject, handles);


% --------------------------------------------------------------------
function volnum_percu_Callback(hObject, eventdata, handles)
%  Inicializa la casilla de texto donde se indicará el numero de
volumen de la percusion


% --------------------------------------------------------------------
function play_all_Callback(hObject, eventdata, handles)
% Reproduce los tres instrumentos simultaneamente teniendo en cuenta
la posicion de la guia y las opciones mute

global matrizguit matrizsinte matrizpercu
fm=22050;

% Se comprueba si está marcada la opcion mute de cada instrumento
g=get(handles.mute_guit,'Value');
s=get(handles.mute_sinte,'Value');
p=get(handles.mute_percu,'Value');
```

```
% Se pone a 0 el instrumento si está marcado el mute y se introduce en
una nueva variable
if g==0
    guitar=matrizguit;
else
    guitar=0;
end

if s==0
    sinte=matrizsinte;
else
    sinte=0;
end

if p==0
    percu=matrizpercu;
else
    percu=0;
end

% Se calculan los tamaños de cada nueva variable de instrumento y se
% convierten a vectores
sgui=size(guitar);
if sgui(1)==1
else
    guitar=sum(guitar);
end

ssinte=size(sinte);
if ssinte(1)==1
else
    sinte=sum(sinte);
end

spe=size(percu);
if spe(1)==1
else
    percu=sum(percu);
end

% Se crea un vector con las longitudes de los instrumentos
h=[sgui(1,2),ssinte(1,2),spe(1,2)]


% Se crean 3 nuevos vectores con la misma longitud
% (la mayor de entre los 3) para poder realizar la suma
guitar2=zeros(1,max(h));
guitar2(1:sgui(1,2))=guitar;
sinte2=zeros(1,max(h));
sinte2(1:ssinte(1,2))=sinte;
percusion2=zeros(1,max(h));
percusion2(1:spe(1,2))=percu;

% Multiplicamos por sus volumenes
guitar2=handles.volumen_guit*guitar2;
sinte2=handles.volumen_sinte*sinte2;
percusion2=handles.volumen_percu*percusion2;

% Se suman los 3 vectores
```

```
handles.compo=guitar2+sinte2+percusion2;
L_compo=length(handles.compo);

% % % Reproducimos dependiendo de la posicion de la guia
handles.composicion=audioplayer(handles.compo,fm);

if handles.posguia(1,1) <= 0
    play(handles.composicion)
elseif handles.posguia(1,1)*fm >L_compo
    play(handles.sonido_guit)
else
    play(handles.composicion, handles.posguia(1,1)*fm)
end

guidata(hObject,handles)


% --------------------------------------------------------------------
function time_num_Callback(hObject, eventdata, handles)
%  Se inicializa la casilla donde se indica la posicion de la guia


% --------------------------------------------------------------------
function stop_all_Callback(hObject, eventdata, handles)
% Se para la reproduccion
stop(handles.composicion)


% --------------------------------------------------------------------
function zoom_all_Callback(hObject, eventdata, handles)
% Se inicializa cómo actúa el comando zoom

state=get(handles.zoom_all,'Value');
if state == 1
    zoom on
    set(handles.zoom_all,'String','Zoom off');
else
    zoom off
    set(handles.zoom_all,'String','Zoom on');
end

% --------------------------------------------------------------------
function desplazar_all_Callback(hObject, eventdata, handles)
%  Se inicializa cómo actúa el comando desplazar los ejes

state=get(handles.desplazar_all,'Value');
if state == 1
    set(handles.desplazar_all,'String','Desplazar off');
    pan on
else
    set(handles.desplazar_all,'String','Desplazar on');
    pan off
end


% --------------------------------------------------------------------
function resetzoom_all_Callback(hObject, eventdata, handles)
%  Resetea el zoom actualizando los ejes
```

```
representar_Callback(hObject, eventdata, handles)


% --------------------------------------------------------------------
function moverguia_all_Callback(hObject, eventdata, handles)
%  Desplaza la guia indicandoselo con el ratón sobre los ejes

posguia=ginput(1);
pos=roundn(posguia(1,1),-2);
pos=num2str(pos);
set(handles.time_num,'String',pos);
handles.posguia=posguia;
guidata(hObject,handles)
representar_Callback(hObject, eventdata, handles)


% --------------------------------------------------------------------
function exportar_Callback(hObject, eventdata, handles)
% Exporta la composición creada en un fichero en formato wav

[nombre,path]=uiputfile('*.wav','Exportar a wav');
if nombre==0, return, end
archivo=fullfile(path,nombre);
wavwrite(handles.compo,22050,archivo)


% --------------------------------------------------------------------
function abrir_sesion_Callback(hObject, eventdata, handles)
%  Abre una sesion que se ha guardado anteriormente

uiload
representar_Callback(hObject, eventdata, handles)


% --------------------------------------------------------------------
function guardar_sesion_Callback(hObject, eventdata, handles)
% Guarda todas las variables globales en un fichero .mat guardando así
la sesion de la composicion creada
global matrizguit etiquetaNota_guit matrizsinte etiquetaNota_sinte
etiquetaOctava_sinte matrizpercu etiquetaInstrumento

uisave({'matrizguit','etiquetaNota_guit','matrizsinte','etiquetaNota_s
inte','etiquetaOctava_sinte','matrizpercu','etiquetaInstrumento'});


% --------------------------------------------------------------------
function nueva_sesion_Callback(hObject, eventdata, handles)
% Inicializa una nueva sesion

global matrizguit etiquetaNota_guit matrizsinte etiquetaNota_sinte
etiquetaOctava_sinte matrizpercu etiquetaInstrumento
matrizguit=[];
etiquetaNota_guit={};
matrizsinte=[];
etiquetaNota_sinte={};
etiquetaOctava_sinte={};
matrizpercu=[];
```

```
etiquetaInstrumento={};
representar_Callback(hObject, eventdata, handles)

% --------------------------------------------------------------------
function representar_Callback(hObject, eventdata, handles)
%  Represena sobre los ejes cada uno de los instrumentos dibujando
cada nota como barras con la misma duracion de la nota.


global matrizguit etiquetaNota_guit matrizsinte etiquetaNota_sinte
etiquetaOctava_sinte matrizpercu etiquetaInstrumento
fm=22050;
Ts=1/fm;


% Calculamos tiempos y tamaños de las matrices para dimensionar los
ejes
durTiempo_guit=length(matrizguit)*Ts;
durTiempo_sinte=length(matrizsinte)*Ts;
durTiempo_percu=length(matrizpercu)*Ts;
xlim=[durTiempo_guit durTiempo_sinte durTiempo_percu];
sgui=size(matrizguit);
ssinte=size(matrizsinte);
spercu=size(matrizpercu);
ylim=[sgui(1) ssinte(1) spercu(1)];


% Definimos los punto de la posicion de la guia
x=[handles.posguia(1,1),handles.posguia(1,1)];
z=[-5,max(ylim)+5];



% REPRESENTA LA GUITARRA
% % ----------------------
set(gcf,'CurrentAxes',handles.eje1guit) % Cambio los ejes que quiero
utilizar
hold off

for i=1:sgui(1);
    inicio_nota=find(matrizguit(i,:),1,'first');
    fin_nota=find(matrizguit(i,:),1,'last');
    T=[inicio_nota*Ts,fin_nota*Ts];
    L=[i,i];
    vacio=isempty(inicio_nota);
    if vacio==1
    else
      plot(T,L,'r','linewidth',6);
      text((T(1)),L(1),[etiquetaNota_guit(i,1)],'Color',[1,1,1],
      'FontSize',6)
       hold on
    end
end
plot(x,z,'r')
set(handles.eje1guit,'Color',[0.7,0.7,0.7],'xticklabel',[],
'yticklabel',[],'XGrid','on')
```

```
% REPRESENTA EL SINTE
% ------------------------------
set(gcf,'CurrentAxes',handles.eje1sinte) % Cambio los ejes que quiero
utilizar
hold off
for i=1:ssinte(1);
    inicio_nota=find(matrizsinte(i,:),1,'first');
    fin_nota=find(matrizsinte(i,:),1,'last');
    T=[inicio_nota*Ts,fin_nota*Ts];
    L=[i,i];
    vacio=isempty(inicio_nota);
    if vacio==1
    else
        plot(T,L,'b','linewidth',6);
        text((T(1)),L(1),[etiquetaNota_sinte(i,1)
        etiquetaOctava_sinte(i,1)],'Color',[1,1,1],'FontSize',6)
        hold on
    end
end
plot(x,z,'r')
set(handles.eje1sinte,'Color',[0.7,0.7,0.7],'xticklabel',[],'yticklabe
l',[],'XGrid','on')


% REPRESENTA LA PERCUSION
% ------------------------------------
set(gcf,'CurrentAxes',handles.eje1percu) % Cambio los ejes que quiero
utilizar
hold off
for i=1:spercu(1);
    inicio_nota=find(matrizpercu(i,:),1,'first');
    fin_nota=find(matrizpercu(i,:),1,'last');
    T=[inicio_nota*Ts,fin_nota*Ts];
    L=[i,i];
    vacio=isempty(inicio_nota);
    if vacio==1
    else
      plot(T,L,'k','linewidth',6);
      text((T(1)),L(1),[etiquetaInstrumento(i,1)],'Color',[1,1,1],
      'FontSize',6)
       hold on
    end
end
plot(x,z,'r')
set(handles.eje1percu,'Color',[0.7,0.7,0.7],'yticklabel',[],
'XGrid','on')
axis([0 max(xlim)+0.5 0.5 max(ylim)+0.6])
xlabel('Tiempo (s)','FontSize',7)
```

7.2 Código guitarra.m

```
function varargout = guitarra(varargin)
% Abre la interfaz gráfica guitarra.m que es llamada desde la interfaz
% general secuenciador.m
% Con esta aplicación se puede componer la pista de guitarra. Tiene
% diferentes funciones como aplicar distorsión, posicionar notas o
eliminar
% notas ya creadas.

%  ------------------------------------------------------------------



% Comienza el código de inicialización. Este código es programado
% automáticamente por la aplicación Guide -- NO EDITABLE
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @guitarra_OpeningFcn, ...
    'gui_OutputFcn',  @guitarra_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
    'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% Fin del código de inicialización -- NO EDITABLE



%  ------------------------------------------------------------------
function guitarra_OpeningFcn(hObject, eventdata, handles, varargin)

% INICIALIZACIONES DE VARIABLES GLOBALES
global matrizguit etiquetaNota_guit

% ------------------------------------------------------------------
% INICIALIZACIONES DE VARIABLES HANDLES PARA PODER UTILIZARLAS EN
DIVERSAS FUNCIONES
handles.L_gui=length(matrizguit);
handles.PN=1;
handles.dist_guit=0;
handles.posguia=0;

representar_Callback(hObject, eventdata, handles)
% -----------------------------------------------------

handles.output = hObject;

guidata(hObject, handles);
```

```
% --------------------------------------------------------------------
function varargout = guitarra_OutputFcn(hObject, eventdata, handles)

varargout{1} = handles.output;


% --------------------------------------------------------------------
function nota_guit_Callback(hObject, eventdata, handles)
% Inicializa el selector de notas de la guitarra


% --------------------------------------------------------------------
function insert_guit_Callback(hObject, eventdata, handles)
%  Al presionar este botón inserta la nota con los parámetros
indicados en la matriz de guitarra

global matrizguit etiquetaNota_guit
fm=22050;
Ts=1/fm;
A=440;
y=0;
r=2^(1/12);


% Se calcula la duración y el instante de la nota
if handles.PN==0  % En el caso que el usuario posicione la nota
    instante_guit=handles.posicion(1,1)*fm;
    dur=handles.posicion(2,1)-handles.posicion(1,1)
else % Si la nota viene dada por la duracion e instante escritos
    dur=str2double(get(handles.dur_guit,'String'));
    instante_guit=str2double(get(handles.instante_guit,'String'));
    if instante_guit>=0   % En el caso de que la casilla instante no
sea rellenada colocará la nota al final de la matriz
        instante_guit=instante_guit*fm;
    else
        instante_guit=handles.L_gui;
    end
end

t=0:1/fm:dur;

% ------------------------------------------------
% SELECTOR DE NOTA DE GUITARRA
% ------------------------------------------------
N=get(handles.nota_guit,'Value');
textnota=get(handles.nota_guit,'String');

switch N

    case 1  % SILENCIO
        rel_oct=0;
        oct=1;
        comp=[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0];

    case 2     % MI3
        rel_oct=[-41 -29 -17 -5 7 19 31 43];
        oct=3;
        comp=[ 0.0068 0.0961 0.0237 0.0050 0.0005 0.0016 0.0011 0.0003
               0.0016 0.0004 0 0 0 0 0 ];
```

```
case 3      %  FA3

    rel_oct=[-40 -28 -16 -4 8 20 32 44];
    oct=3;
    comp=[0.0213 0.0974 0.0111 0.0050 0.0011 0.0005 0.0014 0.0011
          0.0004 0.0004 0 0 0 0 0 ];

case 4      % FA#3

    rel_oct=[-39 -27 -15 -3 9 21 33 45];
    oct=3;
    comp=[0.0334 0.0948 0.0092 0.0058 0 0.0020 0.0009 0.0014 0 0 0
           0 0 0 0 ];

case 5     % SOL3
    rel_oct=[-38 -26 -14 -2 10 22 34 46];
    oct=3;
    comp=[0.0645 0.0724 0.0132 0.0066 0.0009 0.0018 0.0021 0.0041
          0 0 0 0 0 0 0];

case 6       %  SOL#3
    rel_oct=[-37 -25 -13 -1 11 23 35 47];
    oct=3;
    comp=[0.0927 0.0645 0.0066 0.0016 0.0009 0.0021 0.0004 0
           0.0003 0.0013 0 0 0 0 0.0001];

case 7       % LA3
    rel_oct=[-36 -24 -12 0 12 24 36 48];
    oct=3;
    comp=[0.1316 0.0421 0.0145 0.0039 0.0005 0.0009 0.0032 0.0037
          0.0007 0 0 0 0 0 0];

case 8   % LA#3
    rel_oct=[-35 -23 -11 1 13 25 37 49];
    oct=3;
    comp=[0.1659 0.0237 0.0011 0.0008 0 0.0005 0.0004 0.0026
        0.0001 0 0 0 0 0 0];

case 9      % SI3
    rel_oct=[-34 -22 -10 2 14 26 38 50];
    oct=3;
    comp=[0.1132 0.0158 0.0026 0.0004 0 0.0005 0.0008 0.0007
          0.0004 0 0 0 0 0 0];

case 10     % DO4
    rel_oct=[-45 -33 -21 -9 3 15 27 39];
    oct=4;
    comp=[0.2040 0.0105 0.0145 0.0013 0 0.0021 0.0011 0.0008 0 0 0
           0 0 0 0];

case 11     % DO#4
    rel_oct=[-40 -32 -20 -8 4 16 28 40];
    oct=4;
    comp=[0.1685 0.0066 0.0058 0.0007 0.0007 0.0008 0.0013 0 0 0 0
          0 0 0 0];

case 12     % RE4
    rel_oct=[-43 -31 -19 -7 5 17 29 41];
```

```
        oct=4;
        comp=[0.1290 0.0079 0.0092 0.0008 0.0014 0.0018 0.0021 0.0032
              0.0007 0.0002 0 0.0001 0.0001 0.0003 0.0006];

    case 13    % RE#4
        rel_oct=[-42 -30 -18 -6 6 18 30 42];
        oct=4;
        comp=[0.0927 0.0090 0.0047 0.0013 0.0007 0.0012 0.0009 0.0006
              0 0.0002 0.0002 0.0003 0.0005 0.0003 0.0001];

    case 14  % MI4
        rel_oct=[-41 -29 -17 -5 7 19 31 43];
        oct=4;
        comp=[0.0816 0.0197 0.0058 0.0022 0 0.0013 0.0008 0 0 0 0
             0.0006 0 0 0];

    case 15  % FA4
        rel_oct=[-40 -28 -16 -4 8 20 32 44];
        oct=4;
        comp=[0.0684 0.0184 0.0013 0.0016 0.0061 0.0050 0.0008 0 0 0
              0.0001 0.0002 0.0009 0.0005 0];

    case 16  %FA#4
        rel_oct=[-39 -27 -15 -3 9 21 33 45];
        oct=4;
        comp=[0.0763 0.0266 0.0016 0.0018 0.0037 0.0005 0.0007 0
              0.0002 0.0002 0.0002 0.0001 0.0005 0.0002 0.0002];

    case 17  %SOL4
        rel_oct=[-38 -26 -14 -2 10 22 34 46];
        oct=4;
        comp=[0.0487 0.0408 0.0074 0.0134 0.0007 0.0012 0.0001 0.0002
              0.0005 0.0005 0.0001 0.0005 0.0005 0.0004 0.0001];

    case 18  % SOL#4
        rel_oct=[-37 -25 -13 -1 11 23 35 47];
        oct=4;
        comp=[0.0658 0.0190 0.0033 0.0030 0.0005 0.0005 0.0003 0.0001
              0.0004 0.0006 0.0003 0.0002 0.0001 0.0002 0];

    case 19  % LA4
        rel_oct=[-36 -24 -12 0 12 24 36 48];
        oct=4;
        comp=[0.0276 0.0153 0.0037 0.0008 0.0014 0.0001 0 0.0003
              0.0005 0.0001 0.0002 0.0001 0.0002 0.0001 0];

    case 20  % LA#4
        rel_oct=[-35 -23 -11 1 13 25 37 49];
        oct=4;
        comp=[0.0284 0.0092 0.0066 0.0037 0.0008 0.0006 0.0004 0.0006
              0.0002 0.0017 0.0002 0.0002 0.0001 0 0];

    case 21  % SI4
        rel_oct=[-34 -22 -10 2 14 26 38 50];
        oct=4;
        comp=[0.0211 0.0125 0.0082 0.0116 0.0020 0.0003 0.0009 0.0016
              0.0007 0.0005 0.0001 0.0001 0.0001 0.0001 0.0001];

    case 22  % DO5
```

```
        rel_oct=[-45 -33 -21 -9 3 15 27 39];
        oct=5;
        comp=[0.0111 0.0100 0.0090 0.0029 0.0008 0.0001 0.0010 0.0015
              0.0047 0.0009 0 0 0 0 0.0004];

    case 23  % DO#5
        rel_oct=[-40 -32 -20 -8 4 16 28 40];
        oct=5;
        comp=[0.0153 0.0108 0.0026 0.0045 0.0001 0.0001 0.0029 0.0016
              0.0013 0.0001 0 0.0002 0 0.0009 0.0006];

    case 24  % RE5
        rel_oct=[-43 -31 -19 -7 5 17 29 41];
        oct=5;
        comp=[0.0176 0.0090 0.0132 0.0019 0.0021 0.0004 0.0042 0.0092
              0.0003 0.0007 0.0003 0 0.0001 0.0008 0.0020];

    case 25  % RE#5
        rel_oct=[-42 -30 -18 -6 6 18 30 42];
        oct=5;
        comp=[0.0240 0.0084 0.0187 0.0022 0.0005 0.0004 0.0006 0.0041
              0 0.0001 0.0003 0.0001 0.0004 0 0.0004];

    case 26  % MI5
        rel_oct=[-41 -29 -17 -5 7 19 31 43];
        oct=5;
        comp=[0.0132 0.0118 0.0408 0.0039 0.0009 0.0010 0.0155 0.0093
              0.0023 0.0005 0.0001 0.0001 0.0003 0.0010 0.0015];

    case 27  % FA5
        rel_oct=[-40 -28 -16 -4 8 20 32 44];
        oct=5;
        comp=[0.0140 0.0076 0.0376 0.0020 0.0004 0.0012 0.0121 0.0014
              0.0003 0.0002 0.0001 0.0005 0.0007 0.0029 0.0011];

    case 28  %FA#5
        rel_oct=[-39 -27 -15 -3 9 21 33 45];
        oct=5;
        comp=[0.0150 0.0129 0.0091 0.0036 0.0009 0.0030 0.0029 0.0038
              0.0002 0.0002 0.0016 0.0014 0.0023 0.0017 0.0012];

    case 29  % SOL5
        rel_oct=[-38 -26 -14 -2 10 22 34 46];
        oct=5;
        comp=[0.0274 0.0170 0.0140 0.0002 0.0004 0.0165 0.0020 0.0003
              0.0004 0.0001 0.0011 0.0038 0.0018 0.0007 0.0003];

    case 30  % SOL#5
        rel_oct=[-37 -25 -13 -1 11 23 35 47];
        oct=5;
        comp=[0.0197 0.0284 0.0051 0.0004 0.0068 0.0057 0.0001 0.0000
              0.0008 0.0017 0.0030 0.0018 0 0.0002 0.0001];

    case 31  % LA5
        rel_oct=[-36 -24 -12 0 12 24 36 48];
        oct=5;
        comp=[0.0108 0.0571 0.0011 0.0003 0.0039 0.0042 0.0020 0.0004
              0.0013 0.0013 0.0022 0.0008 0.0016 0.0014 0];
```

```
    case 32  % LA#5
        rel_oct=[-35 -23 -11 1 13 25 37 49];
        oct=5;
        comp=[0.0076 0.0384 0.0010 0.0012 0.0043 0.0004 0.0005 0.0001
              0.0003 0.0020 0.0020 0.0017 0.0001 0.0007 0.0003];

    case 33  % SI5
        rel_oct=[-34 -22 -10 2 14 26 38 50];
        oct=5;
        comp=[0.0092 0.0295 0.0012 0.0053 0.0032 0.0026 0.0007 0.0006
              0.0003 0.0024 0.0016 0 0 0.0008 0.0003];

    case 34  % DO6
        rel_oct=[-45 -33 -21 -9 3 15 27 39];
        oct=6;
        comp=[0.0121 0.0140 0.0016 0.0088 0.0072 0.0004 0.0002 0.0011
              0.0019 0.0021 0.0011 0.0009 0.0006 0.0004 0.0004];
    case 35  % DO#6
        rel_oct=[-40 -32 -20 -8 4 16 28 40];
        oct=6;
        comp=[0.0085 0.0074 0.0007 0.0008 0.0012 0.0006 0.0006 0.0002
              0.0025 0.0016 0.0004 0.0018 0.0004 0 0.0001];

    case 36  % RE6
        rel_oct=[-43 -31 -19 -7 5 17 29 41];
        oct=6;
        comp=[0.0116 0.0121 0.0004 0.0182 0.0020 0.0022 0.0007 0.0014
              0.0013 0.0030 0.0013 0.0025 0.0036 0.0007 0.0002];

    case 37  % RE#6
        rel_oct=[-42 -30 -18 -6 6 18 30 42];
        oct=6;
        comp=[0.0190 0.0016 0.0037 0.0051 0.0017 0.0002 0.0034 0.0038
              0.0008 0.0027 0.0009 0.0003 0.0001 0.0001 0.0002];

    case 38  % MI6
        rel_oct=[-41 -29 -17 -5 7 19 31 43];
        oct=6;
        comp=[0.0095 0.0042 0.0059 0.0088 0.0008 0.0012 0.0053 0.0021
            0.0008 0.0002 0.0004 0.0005 0.0003 0.0001 0.0001];

end

f=r^rel_oct(oct)*A; % Cálculo de la frecuencia fundamental teniendo
como referencia A

for i=1:15
    y=y+(comp(i).*sin(2*pi*(i*f/2)*t)); % Se calcula la señal de la
nota mediante el sumatorio de senos multiplicados por sus componentes
end

caida=max(y(:))+4/dur; %  La caida de la exponencial dependerá del
valor máximo se la señal y de la duración de la misma
x=exp(-(caida)*t);          % Se define la exponencial segun la
duración de la nota
y=(5*y).*x; % Se amplifica y se aplica la exponencial a la señal
```

```
% ----------------------------------
% EN EL CASO DE QUE LA CASILLA DISTORSIÓN SEA SELECCIONADA
dist=get(handles.distorsion_guit,'Value');
if dist==1
    a=0.9999;
    k = 2*a/(1-a);
    y=(1+k)*(y)./(1+k*abs(y));
    y=y./x; % Se aplica la inversa de la exponencial para que no
decaiga con el tiempo teniendo así un mayor sostenimiento
else
end
% ----------------------------------

sound(y,fm)
L=length(y);


% Se  inserta la nota creada en la matriz global de "GUITARRA"
durguit_total=instante_guit+L;
s=size(matrizguit);

if s(1)==0
    sonido_guitarra=zeros(1,durguit_total);
else
    if s(2)>durguit_total
        durguit_total=s(2);
    end
    sonido_guitarra=zeros(s(1)+1,durguit_total);
    sonido_guitarra(1:end-1,1:s(2))=matrizguit;
end
sonido_guitarra(end,instante_guit+1:instante_guit+L)=y;
matrizguit=sonido_guitarra;

% Se calcula las nuevas longitudes y tamaños
handles.L_gui=length(matrizguit);
s=size(matrizguit);

% Se inserta el texto de la nota elegida en el vector de texto
etiquetaNota_guit(s(1),1)=textnota(N);


representar_Callback(hObject, eventdata, handles)


handles.PN=1; % Vuelve a poner este valor a 1
guidata(hObject,handles)


% --------------------------------------------------------------------
function borrarnota_guit_Callback(hObject, eventdata, handles)
%  Al presionar el botón borra la nota que se le indique con el raton
%  en el gráfico
global matrizguit

[posborrar]=ginput(1);
posborrar=round(posborrar(2));
matrizguit(posborrar,1:end)=0;
guidata(hObject,handles)
representar_Callback(hObject, eventdata, handles)
```

```
% --------------------------------------------------------------------
function pos_guit_Callback(hObject, eventdata, handles)
% Posiciona la nota que esté seleccionada indicandole con el ratón un
% instante inicial y otro final en el gráfico

handles.PN=1;
[handles.posicion]=ginput(2);
handles.PN=handles.PN-handles.PN;
insert_guit_Callback(hObject, eventdata, handles)


% --------------------------------------------------------------------
function dur_guit_Callback(hObject, eventdata, handles)
% Inicializa la casilla donde se le indica la duración de la nota


% --------------------------------------------------------------------
function instante_guit_Callback(hObject, eventdata, handles)
% Inicializa la casilla de texto donde se le indica el instante
inicial de
% la nota


% --------------------------------------------------------------------
function distorsion_guit_Callback(hObject, eventdata, handles)
%  Inicializa la casilla donde se marcará si se desea que la nota
tenga distorsión


% --------------------------------------------------------------------
function play_guit_Callback(hObject, eventdata, handles)
% Reproduce la pista de guitarra teniendo en cuenta donde está
posicionada la guia

global matrizguit
fm=22050;

% % %  Comprobamos si la matriz guitarra contiene mas de una fila
tam_guit=size(matrizguit);

if tam_guit(1)==1
    reproguit=matrizguit;
else
    reproguit=sum(matrizguit);
end


% % % Reproducimos dependiendo de la posicion de la guia
handles.sonido_guit=audioplayer(reproguit,fm);

if handles.posguia(1,1) <= 0
    play(handles.sonido_guit)
elseif handles.posguia(1,1)*fm >handles.L_gui
    play(handles.sonido_guit)
else
    play(handles.sonido_guit, handles.posguia(1,1)*fm)
```

```
end

guidata(hObject,handles)


% --------------------------------------------------------------------
function stop_guit_Callback(hObject, eventdata, handles)
%  Detiene la reproducción
stop(handles.sonido_guit)


% --------------------------------------------------------------------
function zoom_guit_Callback(hObject, eventdata, handles)
% Se inicializa cómo actúa el comando zoom

state=get(handles.zoom_guit,'Value');
if state == 1
    zoom on
    set(handles.zoom_guit,'String','Zoom off');
else
    zoom off
    set(handles.zoom_guit,'String','Zoom on');
end


% --------------------------------------------------------------------
function desplazareje_guit_Callback(hObject, eventdata, handles)
%  Se inicializa cómo actúa el comando desplazar los ejes

state=get(handles.desplazareje_guit,'Value');
if state == 1
    set(handles.desplazareje_guit,'String','Desplazar off');
    pan on
else
    set(handles.desplazareje_guit,'String','Desplazar on');
    pan off
end



% --------------------------------------------------------------------
function resetzoom_guit_Callback(hObject, eventdata, handles)
%  Resetea el zoom actualizando los ejes

representar_Callback(hObject, eventdata, handles)


% --------------------------------------------------------------------
function moverguia_guit_Callback(hObject, eventdata, handles)
%  Desplaza la guia indicandoselo con el ratón sobre los ejes

posguia=ginput(1);
handles.posguia=posguia;
guidata(hObject,handles)

representar_Callback(hObject, eventdata, handles)
```

```
% --------------------------------------------------------------------
function representar_Callback(hObject, eventdata, handles)
% Representa sobre el eje la matriz de la guitarra dibujando las notas
como
% unas barras de la misma duracción que sus respectivas notas

global matrizguit etiquetaNota_guit

fm=22050;
Ts=1/fm;
hold off
s=size(matrizguit);


for i=1:s(1);
    inicio_nota=find(matrizguit(i,:),1,'first'); % te devuelve la
posicion del primer valor que no es 0
    fin_nota=find(matrizguit(i,:),1,'last');
    T=[inicio_nota*Ts,fin_nota*Ts];
    L=[i,i];
    vacio=isempty(inicio_nota); % devuelve un 1 si la variable
inicio_nota esta vacia (la fila de la matrizguit esta vacia)
    if vacio==1
    else
      plot(T,L,'r','linewidth',6);
      text((T(1)),L(1),[etiquetaNota_guit(i,1)],'Color',[1,1,1],
      'FontSize',6)
      hold on
    end
end

durTiempo=handles.L_gui*Ts;
axis([0 durTiempo+0.5 0.5 s(1)+0.6])
% % % Representamos la guia de los ejes
x=[handles.posguia(1,1),handles.posguia(1,1)];
y=[-5,s(1)+5];
plot(x,y,'r')
% % %  Propiedades del grafico
set(handles.ejeguit2,'Color',[0.7,0.7,0.7],'yticklabel',[],
'XGrid','on')
xlabel('Tiempo (s)','FontSize',7)

guidata(hObject,handles)
```

7.3 Código sintetizador.m

```
function varargout = sintetizador(varargin)
% Abre la interfaz gráfica sintetizador.m que es llamada desde la
% interfaz general secuenciador.m
% Con esta aplicación se puede componer la pista de sintetizador.
% Tiene diferentes funciones como aplicar distorsión, posicionar notas
% o eliminar
% notas ya creadas.

%  ------------------------------------------------------------------------


% Comienza el código de inicialización. Este código es programado
% automáticamente por la aplicación Guide -- NO EDITABLE
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @sintetizador_OpeningFcn, ...
    'gui_OutputFcn',  @sintetizador_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
    'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% Fin del código de inicialización -- NO EDITABLE


%  ------------------------------------------------------------------------
function sintetizador_OpeningFcn(hObject, eventdata, handles,
varargin)

% INICIALIZACIONES DE VARIABLES GLOBALES
global matrizsinte etiquetaNota_sinte etiquetaOctava_sinte

% ------------------------------------------------------------------------
% INICIALIZACIONES DE VARIABLES HANDLES PARA PODER UTILIZARLAS EN
DIVERSAS FUNCIONES
handles.L_sinte=length(matrizsinte);
handles.dist_sinte=0;
handles.PN=1;
handles.posguia=0;

representar_Callback(hObject, eventdata, handles)
% -----------------------------------------------------

handles.output = hObject;

guidata(hObject, handles);
```

```
% --------------------------------------------------------------------
function varargout = sintetizador_OutputFcn(hObject, eventdata,
handles)

varargout{1} = handles.output;


% --------------------------------------------------------------------
function nota_sinte_Callback(hObject, eventdata, handles)
%  Inicializa el selector de notas del sintetizador


% --------------------------------------------------------------------
function insert_sinte_Callback(hObject, eventdata, handles)
%  Al presionar este botón inserta la nota con los parámetros
indicados en la matriz de sintetizador

global matrizsinte etiquetaNota_sinte etiquetaOctava_sinte

fm=22050;
Ts=1/fm;
A=440;
r=2^(1/12);
y=0;


% Se calcula la duración y el instante de la nota
if handles.PN==0  % En el caso que el usuario la nota
    instante_sinte=handles.posicion(1,1)*fm;
    dur=handles.posicion(2,1)-handles.posicion(1,1)
else % Si la nota viene dada por la duracion e instante escritos
    dur=str2double(get(handles.dur_sinte,'String'));
    instante_sinte=str2double(get(handles.instante_sinte,'String'));
    if instante_sinte>=0   % En el caso de que la casilla instante no
sea rellenada colocará la nota al final de la matriz
        instante_sinte=instante_sinte*fm;
    else
        instante_sinte=handles.L_sinte;
    end
end

t=0:1/fm:dur; % vector de tiempo definido por la frecuencia de
muestreo y la duración de la nota

% ------------------------------------------------
% SELECTOR DE NOTA DE SINTETIZADOR
% ------------------------------------------------
N=get(handles.nota_sinte,'Value');
textnota=get(handles.nota_sinte,'String');
v=get(handles.octava_sinte,'Value');
textoctava=get(handles.octava_sinte,'String');

switch N

    case 1 % SILENCIO
        A=0;
        v=1;
        rel_oct=0;
```

```
    case 2  % DO
        rel_oct=[-45 -33 -21 -9 3 15 27 39];

    case 3     % DO#
        rel_oct=[-44 -32 -20 -8 4 16 28 40];

    case 4      %  RE
        rel_oct=[-43 -31 -19 -7 5 17 29 41];

    case 5      % RE#
        rel_oct=[-42 -30 -18 -6 6 18 30 42];

    case 6     % MI
        rel_oct=[-41 -29 -17 -5 7 19 31 43];

    case 7       %  FA
        rel_oct=[-40 -28 -16 -4 8 20 32 44];

    case 8       % FA#
        rel_oct=[-39 -27 -15 -3 9 21 33 45];

    case 9   % SOL
        rel_oct=[-38 -26 -14 -2 10 22 34 46];

    case 10      % SOL#
        rel_oct=[-37 -25 -13 -1 11 23 35 47];

    case 11     % LA
        rel_oct=[-36 -24 -12 0 12 24 36 48];

    case 12     % LA#
        rel_oct=[-35 -23 -11 1 13 25 37 49];

    case 13     % SI
        rel_oct=[-34 -22 -10 2 14 26 38 50];


end

f=r^rel_oct(v)*A; % Cálculo de la frecuencia fundamental teniendo como
referencia A
y=sin(2*pi*f*t); % Cálculo de la señal de la nota


% ------------------------------------
% EN EL CASO DE QUE LA CASILLA DISTORSIÓN SEA SELECCIONADA
dist=get(handles.distorsion_sinte,'Value');
if dist==1
    a=0.9999;
    k = 2*a/(1-a);
    y=(1+k)*(y)./(1+k*abs(y));
    y=0.03*y;
else
end
% ------------------------------------
```

```
sound(y,fm)
L=length(y);


% Se  inserta la nota creada en la matriz global de "SINTETIZADOR"
dursinte_total=instante_sinte+L;
s=size(matrizsinte);

if s(1)==0
    sonido_sinte=zeros(1,dursinte_total);
else
    if s(2)>dursinte_total
        dursinte_total=s(2);
    end
    sonido_sinte=zeros(s(1)+1,dursinte_total);
    sonido_sinte(1:end-1,1:s(2))=matrizsinte;
end
sonido_sinte(end,instante_sinte+1:instante_sinte+L)=y;
matrizsinte=sonido_sinte;

% Se calcula las nuevas longitudes y tamaños
handles.L_sinte=length(matrizsinte);
s=size(matrizsinte);

% Se inserta el texto de la nota y octavas elegidas en sus vectores de
texto
etiquetaNota_sinte(s(1),1)=textnota(N);
etiquetaOctava_sinte(s(1),1)=textoctava(v);


representar_Callback(hObject, eventdata, handles)


handles.PN=1;  % Vuelve a poner este valor a 1
guidata(hObject,handles)



% --------------------------------------------------------------------
function borrarnota_sinte_Callback(hObject, eventdata, handles)
%  Al presionar el botón borra la nota que se le indique con el raton
%  en el gráfico
global matrizsinte

[posborrar]=ginput(1);
posborrar=round(posborrar(2));
matrizsinte(posborrar,1:end)=0;
guidata(hObject,handles)
representar_Callback(hObject, eventdata, handles)



% --------------------------------------------------------------------
function pos_sinte_Callback(hObject, eventdata, handles)
% Posiciona la nota que esté seleccionada indicandole con el ratón un
% instante inicial y otro final en el gráfico
```

```
handles.PN=1;
[handles.posicion]=ginput(2);
handles.PN=handles.PN-handles.PN;
insert_sinte_Callback(hObject, eventdata, handles)


% --------------------------------------------------------------------
function dur_sinte_Callback(hObject, eventdata, handles)
% Inicializa la casilla donde se le indica la duración de la nota


% --------------------------------------------------------------------
function instante_sinte_Callback(hObject, eventdata, handles)
% Inicializa la casilla de texto donde se le indica el instante
inicial de la nota


% --------------------------------------------------------------------
function distorsion_sinte_Callback(hObject, eventdata, handles)
%  Inicializa la casilla donde se marcará si se desea que la nota
tenga distorsión


% --------------------------------------------------------------------
function octava_sinte_Callback(hObject, eventdata, handles)
%  Inicializa el selector de octavas del sintetizador


% --------------------------------------------------------------------
function play_sinte_Callback(hObject, eventdata, handles)
% Reproduce la pista de sintetizador teniendo en cuenta donde está
% posicionada la guia

global matrizsinte
fm=22050;

% % %  Comprobamos si la matriz guitarra contiene mas de una fila
tam_sinte=size(matrizsinte);

if tam_sinte(1)==1
    reprosinte=matrizsinte;
else
    reprosinte=sum(matrizsinte);
end


% % % Reproducimos dependiendo de la posicion de la guia
handles.sonido_sinte=audioplayer(reprosinte,fm);

if handles.posguia(1,1) <= 0
    play(handles.sonido_sinte)
elseif handles.posguia(1,1)*fm >handles.L_sinte
    play(handles.sonido_sinte)
else
    play(handles.sonido_sinte, handles.posguia(1,1)*fm)
end

guidata(hObject,handles)
```

```
% --------------------------------------------------------------------
function stop_sinte_Callback(hObject, eventdata, handles)
% Detiene la reproduccion
stop(handles.sonido_sinte)


% --------------------------------------------------------------------
function zoom_sinte_Callback(hObject, eventdata, handles)
% Se inicializa como actúa el comando zoom

state=get(handles.zoom_sinte,'Value');
if state == 1
    zoom on
    set(handles.zoom_sinte,'String','Zoom off');
else
    zoom off
    set(handles.zoom_sinte,'String','Zoom on');
end



% --------------------------------------------------------------------
function desplazareje_sinte_Callback(hObject, eventdata, handles)
% Se inicializa como actúa el comando desplazar los ejes

state=get(handles.desplazareje_sinte,'Value');
if state == 1
    set(handles.desplazareje_sinte,'String','Desplazar off');
    pan on
else
    set(handles.desplazareje_sinte,'String','Desplazar on');
    pan off
end


% --------------------------------------------------------------------
function resetzoom_sinte_Callback(hObject, eventdata, handles)
% Resetea el zoom actualizando los ejes

representar_Callback(hObject, eventdata, handles)

% --------------------------------------------------------------------
function moverguia_sinte_Callback(hObject, eventdata, handles)
% Desplaza la guia indicandoselo con el raton sobre los ejes

posguia=ginput(1);
handles.posguia=posguia;
guidata(hObject,handles)

representar_Callback(hObject, eventdata, handles)


% --------------------------------------------------------------------
function representar_Callback(hObject, eventdata, handles)
% Representa sobre el eje la matriz del sintetizador dibujando las
notas como unas barras de la misma duracción que sus respectivas notas
```

```
global matrizsinte etiquetaNota_sinte etiquetaOctava_sinte

fm=22050;
Ts=1/fm;
hold off
s=size(matrizsinte);

for i=1:s(1);
    inicio_nota=find(matrizsinte(i,:),1,'first'); % te devuelve la
posicion del primer valor que no es 0
    fin_nota=find(matrizsinte(i,:),1,'last');
    T=[inicio_nota*Ts,fin_nota*Ts];
    L=[i,i];
    vacio=isempty(inicio_nota);
    if vacio==1
    else
      plot(T,L,'b','linewidth',6);grid on;
      text((T(1)),L(1),[etiquetaNota_sinte(i,1)
      etiquetaOctava_sinte(i,1)],'Color',[1,1,1],'FontSize',6)
      hold on
    end
end

durTiempo=handles.L_sinte*Ts;
axis([0 durTiempo+0.5 0.5 s(1)+0.6])
% % % Representamos la guia de los ejes
x=[handles.posguia(1,1),handles.posguia(1,1)];
y=[-5,s(1)+5];
plot(x,y,'r')
% % %  Propiedades del grafico
set(handles.ejesinte2,'Color',[0.7,0.7,0.7],'yticklabel',[])
xlabel('Tiempo (s)','FontSize',7)

guidata(hObject,handles)
```

7.4 Código percusión.m

```
function varargout = percusion(varargin)
% Abre la interfaz gráfica percusion.m que es llamada desde la
interfaz
% general secuenciador.m
% Con esta aplicación se puede componer la pista de percusion dandole
a la
% composición general de la cancion un ritmo caracteristico.
% Se pueden aplicar dos unicos instrumentos de percusión: caja y
bombo.

%  ------------------------------------------------------------------


% Comienza el código de inicialización. Este código es programado
% automáticamente por la aplicación Guide -- NO EDITABLE
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @percusion_OpeningFcn, ...
    'gui_OutputFcn',  @percusion_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
    'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% Fin del código de inicialización -- NO EDITABLE


%  ------------------------------------------------------------------
function percusion_OpeningFcn(hObject, eventdata, handles, varargin)

% INICIALIZACIONES DE VARIABLES GLOBALES
global matrizpercu

% ------------------------------------------------------------------
% INICIALIZACIONES DE VARIABLES HANDLES PARA PODER UTILIZARLAS EN
DIVERSAS FUNCIONES
handles.L_pe=length(matrizpercu);
handles.PN=1;
handles.posguia=0;

representar_Callback(hObject, eventdata, handles)
% ------------------------------------------------------

handles.output = hObject;

guidata(hObject, handles);
```

```
% --------------------------------------------------------------------
function varargout = percusion_OutputFcn(hObject, eventdata, handles)

varargout{1} = handles.output;


% --------------------------------------------------------------------
function elecpercu_Callback(hObject, eventdata, handles)
%  Inicializa el selector de instrumentos de percusion


% --------------------------------------------------------------------
function insert_percu_Callback(hObject, eventdata, handles)
% Al presionar este botón inserta el instrumento de percusión elegido
en el instante indicado

global matrizpercu etiquetaInstrumento

fm=22050;
Ts=1/fm;


% Se calcula la duración y el instante de la nota
if handles.PN==0  % En el caso que el usuario posicione la nota
    instante_pe=handles.posicion(1,1)*fm;
else % Si la nota viene dada por la duracion e instante escritos
    instante_pe=str2double(get(handles.instante_percu,'String'));
    if instante_pe>=0   % En el caso de que la casilla instante no la
rellenemos nos colocará la nota al final del vector
        instante_pe=instante_pe*fm;
    else
        instante_pe=handles.L_pe+1000;
    end
end


% ------------------------------------------------
% SELECTOR DEL INSTRUMENTO PERCUSION
% ------------------------------------------------
N=get(handles.elecpercu,'Value');
textinstrumento=get(handles.elecpercu,'String');

switch N
    case 1  % caja
        t=0:1/fm:0.05;  % Se crea el vector tiempo con duración fija
        y=randn(1,length(t));  % se genera una señal de ruido blando
con la duración del vector tiempo
        s=sin(2*pi*180*t);   % Se genera la onda senoidal a 180 Hz
mediante la ecuacion de sintesis
        y=y+s;  % Se superponen las dos señales para crear el sonido
similar de la caja
        x=exp(-80*t); % Se genera una señal exponencial decreciente
        y=(0.4*y).*x;  % Se multiplica la señal de la caja por la
exponencial para darle la envolvente caracteristica
```

```
    case 2  % bombo
        t=0:1/fm:0.25;  % Se crea un vector de tiempo con duración
fija
        y=3.*sin(2*pi*55*t); % Se crea la señal de bombo mediante la
ecuación de sintesis a una frecuencia de 55Hz
        x=exp(-16*t);  % Se genera una señal exponencial decreciente
que le dará la envolvente caracteristica del sonido original
        y=y.*x;    % Se multiplica la señal exponencial por la señal
para modularla en amplitud

    case 3 % Silencio
        dur=str2double(get(handles.dursil_percu,'String'));
        z=0:1/fm:dur;
        y=sin(2*pi*0*z);
end
% ----------------------------------


sound(y,fm)
L=length(y);
handles.L_pe=length(matrizpercu);


% Se  inserta el instrumento creado en la matriz global de "PERCUSION"
durpercu_total=instante_pe+L;
s=size(matrizpercu);

if s(1)==0
    sonido_percusion=zeros(1,durpercu_total);
else
    if s(2)>durpercu_total
        durpercu_total=s(2);
    end
    sonido_percusion=zeros(s(1)+1,durpercu_total);
    sonido_percusion(1:end-1,1:s(2))=matrizpercu;
end
sonido_percusion(end,instante_pe+1:instante_pe+L)=y;
matrizpercu=sonido_percusion;

% Se calcula las nuevas longitudes y tamaños
handles.L_pe=length(matrizpercu);
s=size(matrizpercu);

% Se inserta el texto del instrumento elegido en su vector de texto
etiquetaInstrumento(s(1),1)=textinstrumento(N);


representar_Callback(hObject, eventdata, handles)


handles.PN=1; % Vuelve a poner este valor a 1
guidata(hObject,handles)


% --------------------------------------------------------------------
function borrarinst_percu_Callback(hObject, eventdata, handles)
%  Al presionar el botón borra el instrumento que se le indique con el
raton
%  en el gráfico
```

```
global matrizpercu

[posborrar]=ginput(1);
posborrar=round(posborrar(2));
matrizpercu(posborrar,1:end)=0;
guidata(hObject,handles)
representar_Callback(hObject, eventdata, handles)


% --------------------------------------------------------------------
function pos_percu_Callback(hObject, eventdata, handles)
% Posiciona el instrumento que esté seleccionado indicandole con el
ratón
% el instante inicial

handles.PN=1;
[handles.posicion]=ginput(1);
handles.PN=handles.PN-handles.PN;
insert_percu_Callback(hObject, eventdata, handles)



% --------------------------------------------------------------------
function dursil_percu_Callback(hObject, eventdata, handles)
%  Inicializa la casilla donde se le indica la duración del silencio
en el
%  caso que se vaya a insertar un silencio


% --------------------------------------------------------------------
function instante_percu_Callback(hObject, eventdata, handles)
%  Inicializa la casilla donde se le indica en que instante se quiere
%  insertar el instrumento seleccionando


% --------------------------------------------------------------------
function play_percu_Callback(hObject, eventdata, handles)
%  Reproduce la pista de percusion teniendo en cuenta donde está
%  posicionada la guia

global matrizpercu
fm=22050;

% % %  Comprobamos si la matriz percusion contiene mas de una fila
tam_percu=size(matrizpercu);

if tam_percu(1)==1
    repropercu=matrizpercu;
else
    repropercu=sum(matrizpercu);
end


% % % Reproducimos dependiendo de la posicion de la guia
handles.sonido_percu=audioplayer(repropercu,fm);

if handles.posguia(1,1) <= 0
```

```
    play(handles.sonido_percu)
elseif handles.posguia(1,1)*fm >handles.L_pe
    play(handles.sonido_percu)
else
    play(handles.sonido_percu, handles.posguia(1,1)*fm)
end

guidata(hObject,handles)


% --------------------------------------------------------------------
function stop_percu_Callback(hObject, eventdata, handles)
% Detiene la reproduccion
stop(handles.sonido_percu)


% --------------------------------------------------------------------
function zoom_percu_Callback(hObject, eventdata, handles)
% Se inicializa como actúa el comando zoom

state=get(handles.zoom_percu,'Value');
if state == 1
    zoom on
    set(handles.zoom_percu,'String','Zoom off');
else
    zoom off
    set(handles.zoom_percu,'String','Zoom on');
end


% --------------------------------------------------------------------
function desplazareje_percu_Callback(hObject, eventdata, handles)
% Se inicializa cómo actúa el comando desplazar ejes

state=get(handles.desplazareje_percu,'Value');
if state == 1
    set(handles.desplazareje_percu,'String','Desplazar off');
    pan on
else
    set(handles.desplazareje_percu,'String','Desplazar on');
    pan off
end


% --------------------------------------------------------------------
function reset_percu_Callback(hObject, eventdata, handles)
% Resetea el zoom actualizando los ejes

representar_Callback(hObject, eventdata, handles)


% --------------------------------------------------------------------
function moverguia_percu_Callback(hObject, eventdata, handles)
%  Desplaza la guia indicándoselo con el raton sobre los ejes

posguia=ginput(1);
handles.posguia=posguia;
guidata(hObject,handles)
```

```
representar_Callback(hObject, eventdata, handles)


% --------------------------------------------------------------------
function representar_Callback(hObject, eventdata, handles)
% Representa sobre el eje la matriz de la percusion dibujando cada
instrumento
% como una barra

global matrizpercu etiquetaInstrumento

fm=22050;
Ts=1/fm;
hold off
s=size(matrizpercu);

for i=1:s(1);
    inicio_nota=find(matrizpercu(i,:),1,'first'); % te devuelve la
posicion del primer valor que no es 0
    fin_nota=find(matrizpercu(i,:),1,'last');
    T=[inicio_nota*Ts,fin_nota*Ts];
    L=[i,i];
    vacio=isempty(inicio_nota);
    if vacio==1
    else
      plot(T,L,'k','linewidth',6);grid on;
      text((T(1)),L(1),[etiquetaInstrumento(i,1)],'Color',[1,1,1],
      'FontSize',6)
      hold on
    end
end

durTiempo=handles.L_pe*Ts;
axis([0 durTiempo+0.5 0.5 s(1)+0.6])
% Representamos la guia de los ejes
x=[handles.posguia(1,1),handles.posguia(1,1)];
y=[-5,s(1)+5];
plot(x,y,'r')
%  Propiedades del grafico
set(handles.ejepercu2,'Color',[0.7,0.7,0.7],'yticklabel',[])
xlabel('Tiempo (s)','FontSize',7)

guidata(hObject,handles)
```