

# Nueva metodología de enseñanza de procesamiento digital de la señal utilizando la API “joPAS”

J. VICENTE, B. GARCÍA, I. RUIZ, A. MENDEZ, O. LAGE

*Departamento de Arquitectura de los Computadores, Automática, Electrónica y Telecomunicaciones.  
Universidad de Deusto. España.*

*Este artículo presenta la API de programación “joPAS” desarrollada por el grupo de investigación PAS de la Universidad de Deusto. “joPAS” permite el uso de variables y funciones de “Octave” desde un programa realizado en “Java”. Esta API posibilita a los estudiantes el rápido desarrollo de aplicaciones de procesamiento digital de señal, haciendo uso de la sencillez de diseño y potencia de interfaces gráficas en lenguaje “Java” complementado con el cálculo científico en “Octave”.*

## 1. Introducción

La formación de los ingenieros en informática no incluye necesariamente el dominio de las técnicas algorítmicas empleadas en el procesamiento digital de señales de voz o imagen [1][2]. Esto dificulta el desarrollo de algunas aplicaciones donde sea necesario incluir los citados algoritmos dentro de un entorno “Java”. Por ello, surgió la necesidad de permitir el acceso desde este entorno a una herramienta como “Octave” que ofrece un conjunto de funciones matemáticas preprogramadas.

Esta API proporciona una herramienta para que los alumnos desarrollen sus propias aplicaciones de procesamiento de señal de una forma rápida y sencilla. “joPAS” permite que la mayor parte del esfuerzo del alumno se centre en el desarrollo del algoritmo de procesamiento digital de señal, ya que la implementación de la interfaz gráfica se simplificará mediante el uso de “Java”.

El objetivo inmediato que se plantea para “joPAS” es que la comunidad de estudiantes continúe utilizando esta API en sus prácticas de asignaturas y proyectos de fin de carrera. De esta manera, se conseguirá que la mayor parte de tiempo que dediquen nuestros alumnos a sus proyectos esté orientado a la implementación de los algoritmos de procesamiento, programándolos en el lenguaje científico “Octave” conocido para ellos. Minimizando así el tiempo y esfuerzo del desarrollo de la parte gráfica de la aplicación. Esto no quiere decir que las aplicaciones resultantes no tengan un buen interfaz gráfico de usuario, sino que, gracias al uso de “joPAS”, se logran de una forma más rápida.

Basado en las experiencias resultantes de nuestros alumnos con el uso de “joPAS” se van a identificar puntos de mejora, incorporables a la API. Con la versión actual de “joPAS” se ha conseguido un paso crucial, siendo nuestra intención que “joPAS” sea algo vivo, que evolucione y mejore. Esto lo conseguiremos realimentándonos de las experiencias obtenidas por los alumnos.

## 2. Descripción

### 2.1 Octave

Octave es un lenguaje de alto nivel para el cálculo numérico, siendo su sintaxis compatible con “Matlab”, ampliamente utilizado por el entorno docente, pero desarrollado por la comunidad de software libre[5]. Octave es un lenguaje especialmente orientado al mundo científico.

Las características de Octave permiten que los algoritmos científicos se desarrollen en mucho menos tiempo que en otros lenguajes de programación. De este modo, Octave es el lenguaje ideal para el desarrollo de algoritmos de procesamiento digital de la señal (voz, imagen, etc.) [3][4], de sistemas de control, estadística u otros.

Aunque Octave sea un lenguaje idóneo para desarrollar aplicaciones científicas, tiene algunas desventajas. Uno de los inconvenientes está relacionado con la velocidad de computación. Octave, al ser

un lenguaje de programación interpretado, es más lento que un lenguaje compilable, ya que éste generaría instrucciones nativas para el procesador, que se ejecutarían más rápido.

La segunda desventaja es la inexistencia de un entorno gráfico. Las aplicaciones con Octave se ejecutan sobre consola, con la única posibilidad de realizar representaciones gráficas de datos. Esto imposibilita el desarrollo de interfaces de usuario para que éste pueda interactuar con la aplicación.

## 2.2 Java

Java es un lenguaje de programación orientado a objetos, desarrollado por James Gosling en Sun Microsystems a principio de los años 90 [6]. El lenguaje, fue diseñado para ser independiente de la plataforma, y es un derivado de C++ con una sintaxis más simple, un entorno de ejecución más robusto y gestión simplificada de la memoria.

Las principales características del lenguaje de programación “Java” incluyen un lenguaje simple que no requiere un largo aprendizaje por parte del programador. Los conceptos fundamentales de la tecnología “Java” se adquieren rápidamente [8].

Para la computación científica hay lenguajes más adecuados que “Java” y con más posibilidades de cálculo. Además, estos lenguajes funcionan nativamente con matrices y números complejos, y “Java” no lo hace. Por esa razón, los lenguajes como “Matlab” y “Octave” se utilizan en la computación científica. Es mucho más difícil programar cálculos científicos en “Java” que en los lenguajes específicos para ello. Supone una gran desventaja para “Java”.

Sin embargo, “Java” tiene muchas opciones al desarrollo de interfaces de usuario. Los lenguajes científicos libres como “Octave” se pueden utilizar solamente en línea de comandos y no se puede crear con ellos aplicaciones de usuario. La solución fue desarrollar los interfaces gráficos en “Java” y el cálculo científico se siguiese realizando en “Octave”. Pero no existía ningún nexo en común entre los dos lenguajes de programación.

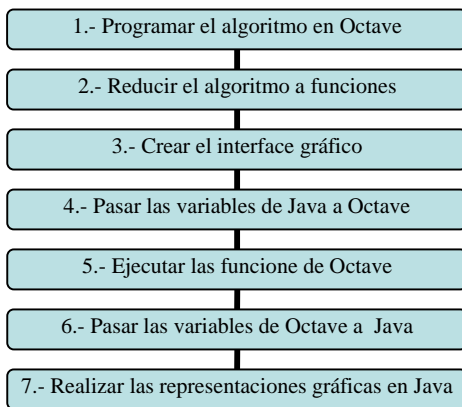
En resumen, el modo de trabajo con “joPAS” sería: desarrollar la aplicación en “Octave”, como se realizaría tradicionalmente y, proveer al algoritmo desarrollado en “Octave” un envoltorio desarrollado en “Java” que le proporcione la interfaz gráfica de usuario.

## 3. Diseño

Tomando como premisas la necesidad de reutilizar el código de los algoritmos desarrollados en Octave en vez de volver a codificar los algoritmos en otro lenguaje y que además dispusiésemos de capacidades de interfaces gráficas de usuario, se buscó cuál podría ser la mejor propuesta. La solución adoptada fue desarrollar los interfaces gráficos en Java y el cálculo científico se siguiese realizando en Octave, pero no existía ningún nexo en común entre los dos lenguajes de programación. Se comenzó, por ello, el desarrollo de la API “joPAS” como un nexo de unión entre Java y Octave. “joPAS” permite intercambiar variables entre ambos lenguajes y ejecutar sentencias de Octave desde Java. La metodología de trabajo con “joPAS” sería la siguiente:

1. Programar el algoritmo que se desea en Octave, utilizando toda la potencia que ofrece.
2. Reducir el algoritmo en funciones de Octave, de modo que todo el cálculo del mismo se realice dentro de estas funciones. Funciones a las que sólo hay que pasar los parámetros fundamentales y que devuelvan los resultados finales del algoritmo.
3. Crear la interfaz gráfica, que se desea que tenga la aplicación, en Java. Este paso se podría simplificar utilizando un entorno de desarrollo como por ejemplo Netbeans.
4. En las funciones de los eventos del interfaz de usuario se recogerían los datos necesarios de la interfaz gráfica y se generarían las correspondientes variables de Octave.
5. Con las variables generadas se invocarían a las funciones deseadas de Octave desde la aplicación de Java, generando las variables de salida.

6. Tras ejecutar los algoritmos de Octave se solicitaría desde Java las variables de salida obtenidas, pasando las variables de Octave a Java.
7. Para finalizar se visualizarían los resultados en la interfaz gráfica y si fuese necesario realizar alguna representación gráfica se invocaría a las funciones añadidas a la API “joPAS”.



**Figura 1.** Organigrama de la metodología de diseño utilizando joPAS

En resumen, el modo de trabajo con “joPAS” sería: desarrollar la aplicación en Octave, como se realizaría tradicionalmente, y proveer al algoritmo desarrollado en Octave un envoltorio desarrollado en Java que le proporcione la interfaz gráfica de usuario. De modo que el tiempo invertido en implementar la aplicación con el entorno gráfico se minimiza ya que en Java solo se implementarían las ventanas de la aplicación, ya que la lógica de la aplicación se obtendría de la invocación de las funciones desarrolladas en Octave.

#### 4. Estructura de “joPAS”

Como ya se comentado, “joPAS” es una API de Java, la que contiene numerosas clases, las más importantes serían las siguientes:

- Jopas → Esta es la clase fundamental de la API, es la encargada de gestionar la comunicación con Octave mediante tres métodos, load, save y execute.
- Matrix → Esta clase contiene el tipo de datos que entiende la clase jopas. Esta clase es un contenedor de matrices, que pueden ser reales o complejas. La clase jopas admite en el método load ese tipo de datos y en el método save siempre devuelve un objeto tipo Matriz.
- JopasLabel → Esta clase es una reimplementación de la clase JLabel de Java, el cual tiene un método importante llamado print. El método print se encarga de realizar la representación grafica de una señal en el label.
- A continuación, se puede observar un breve fragmento de código en el se hace uso de las clases comentadas.

```

Double b = 2;
Matrix mA= new Matrix (a,"a");
jopas.Load(mA);
  
```

```
jopas.execute("b=a+4");  
Matrix mB = jopas.Save("b");
```

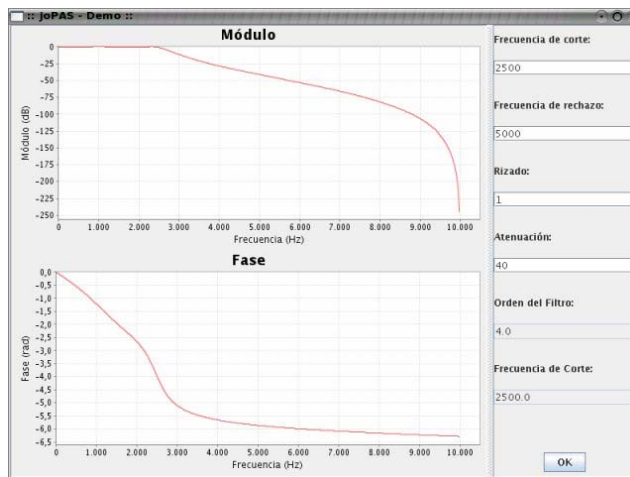
**Algoritmo 1.** Código de ejemplo de uso de “joPAS”.

En el fragmento del Algoritmo 1 se puede observar como se crea un objeto matriz que contendrá un escalar, esta matriz se carga en Octave con el método Load. Después se ejecuta una sentencia de Octave la que suma una unidad a la variable generada. Para finalizar se recupera el resultado de obtenido en Octave, generando un objeto matriz en Java.

La estructura de la API se ha simplificado de tal forma que prácticamente solo hace falta saber utilizar tres clases de la misma, para poder desarrollar aplicaciones con un interfaz gráfico. Todo el proceso de comunicación entre Java y Octave, cargar variables de Java a Octave, ejecutar sentencias de Octave y salvar variables de Octave a Java, las realiza la API de forma transparente al usuario. Por lo que, el tiempo que se tarda en aprender a utilizar “joPAS” es mínimo, para alumnos que sepan programar en Octave y en Java, como es nuestro caso.

## 5. Resultados

Como ya se ha comentado, la implementación de aplicaciones de procesamiento de señal utilizando “joPAS” es extremadamente sencilla. A continuación se describe una aplicación de diseño de filtros paso bajo Chebichev, realizada como demo del uso de “joPAS” para nuestros alumnos. En la figura 1 se puede ver la interface de usuario implementada en Java.



**Figura 2.** Interface de usuario de la aplicación

Primero se busca la sentencias de Octave que permiten diseñar un filtro paso bajo Chebichev y calcular su respuesta frecuencial. En el Algoritmo 2 se pueden ver las instrucciones necesarias.

```
[N,W]=cheb1ord(FC/10000,FR/10000,R,A);  
[B,A]=cheby1(N,R,W);  
[H,F]=freqz(B,A,512,20000);  
modulodB=20*log10(abs(H));
```

```
fase=unwrap(angle(H));  
Fc=W*10000;
```

### Algoritmo 2. Sentencias de Octave.

Por un lado, se diseña la interface de usuario de la aplicación, tarea que puede simplificarse con el uso de un IDE que permita la creación de interfaces gráficas de Java de forma visual. Por otro lado, se implementará en Octave el algoritmo para el diseño del filtro digital.

La interface de usuario consta de:

- Dos JopasLabel, en el superior se representará el módulo de la respuesta frecuencial del filtro diseñada y en el inferior se representará la fase de la respuesta frecuencial del filtro
- Cuatro jTextField en los que se introducirá los siguientes parámetros.
  1. Frecuencia de corte del filtro.
  2. Frecuencia de rechazo del filtro.
  3. Rizado en la banda de paso.
  4. Atenuación de la banda de rechazo.
- Dos jTextField en los que se visualizarán los siguientes datos:
  1. Orden del filtro
  2. Frecuencia de corte del filtro diseñado.
- Un botón para lanzar el proceso de diseño del filtro. En el método de actionPerformed realizará las llamadas necesarias para implementar el filtro. En el Algoritmo 3 se puede observar las sentencias usando “joPAS” para comunicar los datos en Java y Octave.

Como se puede comprobar con el ejemplo, algoritmo 3, mediante el uso de “joPAS” se pueden diseñar aplicaciones de una forma muy rápida y de forma muy sencilla. Por ello, su uso es muy intuitivo para los alumnos.

```
//Read the values of the input textFields of the GUI  
String FC = jTextField1.getText();  
String FR = this.jTFFrecCorte.getText();  
String R = this.jTextField3.getText();  
String A = this.jTextField4.getText();  
  
//Generate de OCTAVE variables  
jopas.Load(Double.parseDouble(FC), "FC");  
jopas.Load(Double.parseDouble(FR), "FR");  
jopas.Load(Double.parseDouble(R), "R");  
jopas.Load(Double.parseDouble(A), "A");  
  
//Executes the Octave commands using local variables  
  
jopas.Execute("[N,W]=cheb1ord(FC/10000,FR/10000, R , A)");  
  
jopas.Execute("[B,A]=cheby1(N, R + W)");  
jopas.Execute("[H,F]=freqz(B,A,512,20000)");  
jopas.Execute("modulodB=20*log10(abs(H))"); jopas.Execute("fase=unwrap(angle(H))");
```

```
jopas.Execute("Fc=W*10000");

//Write the values at the output textFields
this.jTextField6.setText(Double.toString(jopas.Save("N").getRealAt(0,0)));
this.jTextField5.setText(Double.toString(jopas.Save("Fc").getRealAt(0,0)));

//Plot the graphic representation of the frequency response
this.jopasLabel2.paintLabel("F","modulodB", "Módulo","Frecuencia (Hz)", "Módulo (dB)");
this.jopasLabel1.paintLabel("F","fase","Fase", "Frecuencia (Hz)", "Fase (rad)");
```

**Algoritmo 2.** Código Java del método del botón del GUI.

## 6. Planes de futuro

El objetivo inmediato de “joPAS” que nos planteamos es que la comunidad de estudiantes continúen a utilizando esta API en sus proyectos de fin de carrera. De esta manera conseguiremos que la mayor parte de tiempo que dediquen nuestros alumnos a su proyecto fin de carrera, se concentren en la implementación de los algoritmos de procesado, implementándolos en el lenguaje de programación que conocen en profundidad, Octave. Minimizando el tiempo y esfuerzo del desarrollo de la parte gráfica de la aplicación. Esto no quiere decir que las aplicaciones resultantes no tengan un buen interfaz gráfico de usuario, sino que gracias al uso de “joPAS” se logran de una forma más rápida.

En base a las experiencias resultantes de nuestros alumnos con el uso de “joPAS” se van a identificar puntos de mejora, incorporables a la API. Con la versión actual de “joPAS” se ha conseguido un paso crucial, nuestra intención es que “joPAS” sea algo vivo, que evolucione, que mejore. Esto lo conseguiremos realimentándonos de las experiencias obtenidas por nuestros alumnos. Hecho que hará que nos involucremos más en el proceso de desarrollo de nuestros alumnos y que nuestros alumnos se sientan más involucrados en el proyecto.

## 7. Alternativas

La alternativa más cercana la a API propuesta sería el uso de Matlab, lenguaje que comparte la misma sintaxis de programación con Octave y que además permite la realización de interfaces gráficos en el propio entorno. Pero presenta una gran limitación, para poder ejecutar cualquier programa realizado en Matlab se debe poseer una licencia de producto. Hecho que hace que la distribución de programas realizados en Matlab sea inviable.

## 8. Conclusiones

Para finalizar, resaltar los beneficios que se consiguen al utilizar la API “joPAS” para el diseño de aplicaciones de procesado de señal que requieran una interfaz gráfica de usuario. Estas ventajas son: por un lado, mediante esta API se permite que la implementación del algoritmo de procesado digital de señal se realice en un lenguaje de programación indicado para esta tarea, “Octave”. Por el otro, la implementación de la interfaz de usuario en “Java”, un lenguaje de programación más indicado para este tipo de tareas. Mediante esta división entre la parte algorítmica y la de visualización se consigue que las aplicaciones se diseñen de una forma más rápida. Siendo la API “joPAS” la que permite realizar este nexo de unión.

## Referencias

- [1] B. García, J. Vicente, I. Ruiz, A. Alonso, E. Loyo, "Regeneration Model for Esophageal Voices" in Proc. BIOMED 2005, Innsbruck, Austria, 2005.
- [2] B. García, J. Vicente, I. Ruiz, A. Alonso, E. Loyo, "Esophageal Voices: Glottal Flow Regeneration" in Proc. ICASSP '05, Philadelphia, EEUU, 2005.
- [3] B. García, J. Vicente, I. Ruiz, A. Alonso, E. Loyo, "Noise Reduction Algorithm for Esophageal Voices" in Proc. IWSSIP'04, Poznan, Polonia, 2004.
- [4] B. García, J. Vicente, "Adaptative Pitch Scaling Algorithm for Esophageal Speech" in Proc. BioSignal 2004, Brno, Czech Republic, 2004.
- [5] Kurt Hornik, Friedrich Leisch, Achim Zeileis, "Ten Years of Octave Recent Developments and Plans for the Future" in Proc. DSC 2003, Vienna, Austria, 2004.
- [6] Ken Arnold and James Gosling, "The Java Programming Language" The Java Series. Addison-Wesley, Reading, MA, 1996.
- [7] J. Angulo, B. García, J. Vicente, I. Angulo, "Microcontroladores avanzados dsPIC", International Thomson Editores, 2005.
- [8] Roberts, E. "Resources to Support the Use of Java in Introductory Computer Science". Bibliographic details 2004, vol 36; part 1, pages 233-234. Acm association for computing machinery.