# Diseño de Sistemas de Control con GNU Octave

Abel Alberto Cuadrado Vega

4 de diciembre de 2008

## 1. Introducción

El objetivo de este documento es enumerar las funciones disponibles en Octave para el análisis de sistemas dinámicos y el diseño de sistemas de control, referidos exclusivamente a sistemas continuos. No pretende ser un manual completo, sino una referencia para orientar al usuario, que tendrá que complementar con el uso de la ayuda (comando `help`) y el conocimiento de la teoría básica de control.

Octave es un lenguaje de alto nivel, principalmente orientado a cálculos numéricos, en gran medida compatible con Matlab. Todo lo mencionado en este documento es válido también en Matlab (suponiendo que éste incluye la *toolbox* de control), aunque ciertas cosas, como el formato de los parámetros, pueden variar ligeramente en algunos casos.

## 2. Funciones Básicas

### 2.1. Complejos

Un valor complejo, por ejemplo, se puede representar de la siguiente manera:

```
c = 1+2*j
```

Además, se dispone de una función para obtener el conjugado (`conj`), que se usaría, por ejemplo:

```
cc = conj(c)
```

También hay función para obtener el módulo (`abs`), para obtener el argumento o ángulo (`angle`), para obtener la parte real (`real`) y para obtener la parte imaginaria (`imag`).

### 2.2. Vectores y matrices

Una matriz se especifica listando sus elementos entre corchetes y separando las columnas con espacio o coma y las filas con punto y coma, por ejemplo:

```
M = [ 1 2 3 ; 4 5 6 ; 7 8 9 ]
```

sería la forma de especificar la matriz:

$$\mathbf{M} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Un vector se especifica análogamente, teniendo en cuenta que no es más que una matriz de una fila (vector fila) o una matriz de una columna (vector columna).

Existe una función especial para generar un vector conteniendo `n` valores equiespaciados, comenzando desde el valor `xi` y acabando en el valor `xf`, la función `linspace(xi,xf,n)`, que se usaría, por ejemplo:

```
v = linspace(0,3,7)
```

También se puede especificar dicho vector, no por el número de elementos, sino por el espaciado entre los mismos. Así, el mismo vector anterior sería, de otra manera:

```
v = 0:0.5:3
```

## 2.3. Polinomios

Un polinomio se especifica como un vector que contiene los coeficientes del polinomio ordenados de mayor a menor grado. Por ejemplo, el polinomio:

$$P(s) = s^3 + 2s^2 + 4$$

se definiría como un vector de la forma:

```
P = [ 1 2 0 4 ]
```

Además se dispone de varias funciones para realizar cálculos con polinomios, como `roots`, para obtener sus raíces, o como `poly`, que sirve para obtener el polinomio a partir de sus raíces. Así, si suponemos que $-1$ y $-3$ son raíces de un cierto polinomio $Q$, éste lo obtendríamos como:

```
Q = poly([ -1 -3 ])
```

También existe una función `conv` que permite realizar el producto de dos polinomios. Por ejemplo, el producto de dos polinomios $P$ y $Q$ sería:

```
PQ = conv(P,Q)
```

## 2.4. Gráficas

La función básica para realizar una gráfica, en uno de los casos de uso más simple, es `plot(x,y)`, donde `x` sería un vector con los valores de las abscisas e `y` un vector con los valores correspondientes de ordenadas de cada punto. Posteriormente a su dibujado, la gráfica se puede etiquetar en cada uno de sus dos ejes con las funciones `xlabel` e `ylabel`, se le puede poner un título con `title`, y se puede hacer que se muestre u oculte una rejilla con el comando `grid on` o `grid off`.

También se pueden representar varias variables en la misma gráfica, haciendo que `x` e `y` sean matrices o usando vectores separados en la forma:

```
plot(x1,y1,x2,y2)
```

donde cada variable se representará con un color y podrá ser identificado en la gráfica por medio de una leyenda que se pondrá con la función `legend`.

Cada ejecución de la función `plot` borrará y sobreescribirá la gráfica previa, a menos que se ejecute el comando `hold on`, con el que se sobreescribirá sin borrar hasta que se ejecute el comando `hold off`. Si se quiere crear una figura nueva en lugar de sobreescribir la antigua, se puede usar el comando `figure`. A continuación se muestra un ejemplo completo de todas las funciones mencionadas:

```
t=linspace(0,10,1000);
y1=2*sin(10*t);
y2=t/10;
figure;
plot(t,y1,t,y2);
xlabel('Tiempo (s)');
ylabel('Corriente (A)');
legend('senoidal','lineal');
title('Evolucion temporal de la corriente');
grid on;
```

Como detalle interesante, las etiquetas, leyenda y título admiten comandos matemáticos LaTeXpara poner subíndices, superíndices y letras griegas.

# 3. Modelos dinámicos: transformada de Laplace

Supuesto el modelo de un sistema de ejemplo, especificado por la función de transferencia siguiente:

$$G(s) = \frac{b_1 s + b_2}{a_0 s^2 + a_1 s + a_2}$$

si representamos los valores $a_0$, $a_1$ y $a_2$, por `a0`, `a1`, y `a2`, y los valores $b_1$ y $b_2$, por `b1` y `b2`, la función de transferencia puede definirse de la siguiente manera:

```
numG=[ b1 b2 ];
denG=[ a0 a1 a2 ];
G=tf(numG,denG)
```

También, en general, una función de transferencia puede ponerse de la forma:

$$G(s) = K \frac{\prod_{i=1}^{m} (s - z_i)}{\prod_{i=1}^{n} (s - p_i)}$$

que deja patentes sus ceros $z_i$ y sus polos $p_i$, y que puede conseguirse con:

```
[z,p,k]=tf2zp(numG,denG)
```

donde se obtiene un vector `z` con los ceros $z_i$, un vector `p` con los polos $p_i$, y el valor `k` conteniendo la constante real $K$. También puede realizarse la conversión inversa con `zp2tf`.

Para resolver un problema de respuesta de un sistema por medio de la Transformada de Laplace, normalmente hay que obtener una transformada inversa de una relación de polinomios. Esto implica descomponer dicha relación de polinomios $Y(s)$ en una suma de fracciones simples, con un término de la forma:

$$Y_i(s) = \frac{C_i}{s - p_i}$$

para cada polo $p_i$ simple de $Y(s)$, o con una suma de términos de la forma:

$$Y_i(s) = \frac{C_{i1}}{s - p_i} + \frac{C_{i2}}{(s - p_i)^2} + \ldots + \frac{C_{ir}}{(s - p_i)^r}$$

para cada polo $p_i$ con multiplicidad $r$. También aparece en la suma un término constante $K$ si $Y(s)$ tiene como numerador y denominador polinomios de igual grado. Para obtener los coeficientes $C_i$ y $C_{ik}$, se dispone de la función `residue`:

```
[r,p,k]=residue(num,den)
```

donde se le pasan como parámetros los polinomios del numerador y denominador y se obtiene un vector `p` con los polos, un vector `r` con los coeficientes $C_i$ y $C_{ik}$ correspondientes a cada polo (residuos) y, en caso de existir, la constante $K$ en la variable `k`.

Para la representación gráfica de la posición de los polos y ceros del sistema en el plano complejo, está disponible la función `pzmap`.

# 4. Álgebra de bloques

Las tres operaciones de álgebra de bloques más básicas, serie, paralelo y realimentación, están disponibles como las funciones `series`, `parallel` y `feedback` respectivamente, que se usarían según el ejemplo que se muestra a continuación:

```
G=series(G1,G2)
G=parallel(G1,G2)
Gcc=feedback(G,H);
```

En la línea del ejemplo de realimentación, se muestra el caso de realimentación negativa. Para obtener realimentación positiva, habría que añadir un tercer parámetro de valor 1, que sería $-1$ para realimentación negativa pero pudiendo omitirse en ese caso.

# 5. Respuesta temporal

Para obtener la respuesta temporal de un sistema ante las entradas básicas impulso y escalón, se tienen las funciones `impulse` y `step` respectivamente. Al usarlas sin parámetros de salida, se dibuja automáticamente la respuesta, pudiendo especificarse como parámetro de entrada adicional el vector de instantes de tiempo en los que se quiere evaluar la salida, u omitirlo y dejar que la función escoja el más adecuado para presentar todo el transitorio.

Para obtener la respuesta ante una entrada cualquiera, se puede usar:

```
lsim(G,u,t)
```

donde `G` es el sistema y `u` es el vector con el valor de la entrada en cada uno de los instantes de tiempo especificados en el vector de tiempos `t`, que tendrán que ser equiespaciados.

Una característica importante tanto desde el punto de vista temporal como frecuencial de un sistema es la ganancia estática, que se puede obtener por medio de la función `dcgain`.

# 6. Respuesta frecuencial

La respuesta frecuencial de un sistema, que se denota como $G(j\omega)$, tiene representación gráfica en forma de diagrama de Bode usando la función `bode`, que dibuja tanto el de magnitudes $|G(j\omega)|$ como el de fases $\angle G(j\omega)$.

Para el análisis en el dominio de la frecuencia de la estabilidad del sistema cuando se realimenta, es útil el diagrama de Nyquist, que se representa usando la función `nyquist`.

# 7. Espacio de estados

Para definir un sistema en espacio de estados dado por las ecuaciones:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \quad (1)$$

$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} \quad (2)$$

se usa la función `ss`, pasándole como parámetros las matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ y $\mathbf{D}$:

```
sis = ss(A,B,C,D)
```

Es posible obtener las matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ y $\mathbf{D}$ de la representación en espacio de estados a partir del numerador y denominador de la representación por función de transferencia con la función `tf2ss`:

```
[A,B,C,D] = tf2ss(numG,denG)
```

También es posible la operación contraria usando la función `ss2tf`.

Hay una función general que es interesante mencionar en este apartado por su aplicabilidad al manejar ecuaciones en espacio de estados. La función `eig` devuelve los vectores y valores propios de una matriz y por lo tanto, cuando es aplicada a la matriz $\mathbf{A}$, permite obtener los polos del sistema:

```
[V,D] = eig(A)
```

donde `V` es una matriz donde cada columna es un vector propio y `D` es una matriz diagonal, conteniendo los valores propios en el orden correspondiente a los vectores propios mencionados.

# 8. Otras funciones

Otras funciones de interés para el análisis de sistemas dinámicos y diseño de control son: `inv`, `logspace`, `loglog`, `semilogx`, `rlocus`, `ss2zp`, `zp2ss`, `tzero`, `place` y `lqr`.

# Índice alfabético