ANÁLISIS NUMÉRICO II

ISABEL MARÍA RAMOS CORONADO
3º I.T. INFORMÁTICA DE GESTIÓN
CURSO 2002/2003

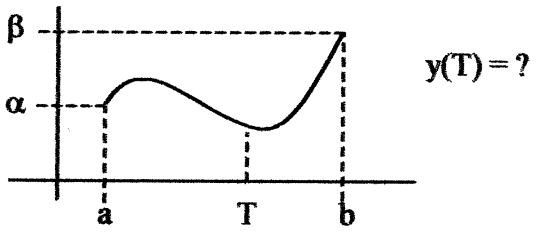
PROBLEMAS DE CONTORNO NO LINEALES

Método del disparo simple (utilizando secante)

El problema que nos encontramos es el siguiente:

$$\begin{cases} y'' = f(t,y,y') \\ y(a) = \alpha \\ y(b) = \beta \end{cases}$$

Dado lo que vale la función y en dos puntos (extremos) y su ecuación diferencial de segundo orden, queremos saber lo que vale y en un punto determinado (T) que se encontrará entre a y b. Gráficamente es lo siguiente:



Inconvenientes:

1.- No lo sabemos resolver.
2.- No sabemos si tiene solución.
3.- En caso de que sí tenga no sabemos cuántas soluciones tiene.

El inconveniente 1 no existiría si en vez del problema que tenemos tuviésemos es siguiente:

$$\begin{cases} y_1' = f(t,y_1,y_2) \\ y_2' = g(t,y_1,y_2) \\ y_1(a) = A \\ y_2(a) = B \end{cases}$$

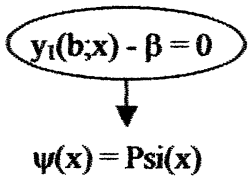
Este sistema lo podemos resolver fácilmente, porque es un problema de valor inicial, el cual se podría resolver usando Euler, Heun, Grag, Runge_kutta ... De tal forma que si podemos pasar el problema de contorno a problema de valor inicial hemos resuelto el primer inconveniente.

Para pasar el problema de contorno a problema de valor inicial lo primero que hay que hacer es pasar la ecuación de segundo orden a sistema de primer orden.

$$y'' = f(t,y,y') \qquad \begin{matrix} y_1 = y \\ \longrightarrow \\ y_2 = y' \end{matrix} \qquad \begin{cases} y_1' = y_2 \\ y_2' = f(t,y_1,y_2) \\ y_1(a) = \alpha \\ y_2(a) = x \end{cases} \Rightarrow F$$

$$y(a) = \alpha$$
$$y'(a) = x$$

Ahora se debe calcular un valor para x tal que $y_1(b) = \beta$, que es lo mismo que decir :



Por lo tanto lo que hay que hacer es definir esa ecuación en una función. Para definir esta ecuación es necesario recurrir a un método que resuelva problemas de valor inicial, por ejemplo Runge_kutta para calcular el valor de y_1:

```
function w0=runge_kutta (f,a,alfa,T,n)
h=(T-a)/n;
t=a;
w0=alfa;
for i=1:n
   f1=feval(f,t,w0);
   f2=feval(f,t+h/2,w0+f1*h/2);
   f3=feval(f,t+h/2,w0+f2*h/2);
   f4=feval(f,t+h,w0+h*f3);
   w1=w0+(h*(f1+2*f2+2*f3+f4)/6);
   t=t+h;
   w0=w1;
end
```

```
function z=psi(x)
y=runge_kutta('F',a,[α,x],b,n)
z=y(1) - β
```

Una vez definida la función, se debe calcular un valor de x que haga cero la función psi, eso es lo mismo que calcularle las raíces a la función psi, lo cual se puede hacer usando el método de bisección, secante, Newton..., en este caso se va a usar el método de la secante, pero antes habrá que hacer un análisis gráfico para obtener las aproximaciones iniciales que hay que pasarle.

El nuevo código del método de la secante es el que se muestra a continuación. Tiene algunos cambios para que no haga calculos redundantes y para que diga si el método converge o no, para saber así si el problema va a tener solución. La salida del método será el punto x en el que se encuentra una raíz y el número de iteraciones (si el método no converge el número de iteraciones será cero).

```
function p0=secante(f,p0,p1,tol,nmax)
converge=0;
fp0=feval(f,p0);  %Para evitar hacer cálculos
fp1=feval(f,p1);  %repetitivos
p2=p1-((p1-p0)*fp1)/(fp1-fp0);
if (abs(p2-p1)<tol)
   n=1;
   converge=1;
   break;
end
p0=p1;
p1=p2;
fp0=fp1;
for n=2:nmax
   fp1=feval(f,p1);
   p2=p1-((p1-p0)*fp1)/(fp1-fp0);
   if (abs(p2-p1)<tol)
      converge=1;
      salir=1;
      p0=p1;
      p1=p2;
      break;
   end
   p0=p1;
   p1=p2;
   fp0=fp1;
end
if (converge==0)  %no converge
   n=nmax;
end
sol(1)=p0;
if (n==nmax) sol(2)=0;
else
   sol(2)=n;
end
```

Una vez se ha aplicado secante, ya se sabe el valor de x que se debe tomar para que se cumpla el problema de contorno, y con ese valor de x ya se puede resolver el problema de valor inicial fácilmente usando cualquier método. Lo que al final se obtenga será $y_1(T) = y(T)$, que es lo que se buscaba.

Volviendo a los inconvenientes que teníamos al principio decir que el tercer inconveniente (numero de soluciones) dependerá del número de raíces que tenga la función psi. Si tiene más de una raíz habrá que resolver el problema para cada una de ellas. El número de soluciones se verá en el análisis gráfico que se le haga al principio a psi.

Finalmente, el código del método del disparo es el siguiente:

```
function z=disparo_sec(F,psi,x0,x1,t)
%x0,x1 son las dos aproximaciones para secante
sec=secante(psi,x0,x1,10^-10,20);
if (sec(2)==0) z=0;  %no tiene solución
else
   x=sec(1);
   y=runge_kutta(F,-1,[-2 x],t,20);
   z=y(1);
end
```

EJERCICIOS:

1.- Resolver el siguiente sistema:

$$\begin{cases} y'' + 6t^2 + t^3 - 1 = 2y' + y + 6t \\ y(-1) = -2 \\ y(1) = 0 \end{cases}$$

a) Calcular y(3)
b) Obtener una tabla con lo que vale y(t) siendo t = (-1,1).

NOTA: Para probar los resultados tener en cuenta que $y(t) = t^3 - 1$

Solución:

a)

1.- Definir el problema como un problema de valores iniciales:

A y se le llama y_1 → $y_1 = y$
A y' se le llama y_2 → $y_2 = y'$
$y'' + 6t^2 + t^3 - 1 = 2y' + y + 6t$ → $y'' = 2y' + y + 6t - 6t^2 - t^3 + 1$

$$\begin{cases} y_1' = y_2 \\ y_2' = 2y_2 + y_1 + 6t - 6t^2 - t^3 + 1 \\ y_1(-1) = -2 \\ y_2(-1) = x \end{cases}$$

```
function z=F(t,y)
z(1)=y(2);
z(2)=2*y(2)+y(1)+6*t-6*t.^2-t.^3+1;
```

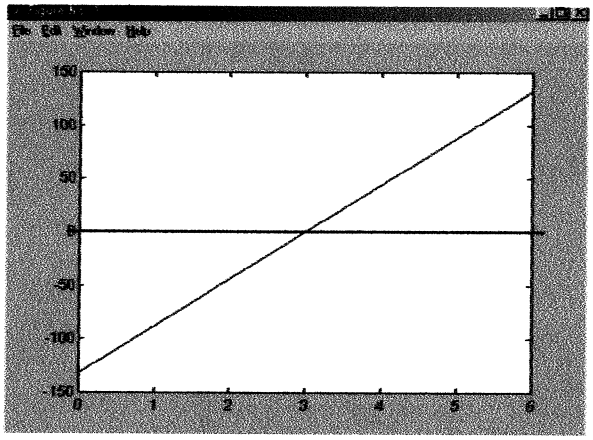
2.- Definir la función psi

```
function z=psi(x)
y=runge_kutta('F',-1,[-2 x],1,10);
z=y(1)-0;
```

3.- Realizar un análisis gráfico de la función psi

Para realizar el análisis gráfico creo una función que dibuje entre los puntos que se le pasen como parámetro, la función que se le indique.

```
function dibujar(x1,x2,f)
paso=x2-x1;
paso=paso/1000;
for i=1:1000
   a(i)=x1;
   b(i)=feval(f,x1);
   x1=x1+paso;
end
plot(a,b);
```



Como puede verse la ecuación sólo tiene una raíz, de tal forma que el problema sólo va a tener una solución.

4.- Ejecutar disparo para obtener el resultado, dándole como aproximaciones 3 y 3.2

disparo_sec('F','psi',3,3.2,3) ⟶ **25.9756**

b)

1.- Defino una función que ayudándose del disparo proporcione la tabla

```
function tabla=disparo_tabla(F,psi,a,b,x0,x1)
tabla=zeros(10,2);
n=(b-a)/10;
i=1;
t=a;
while (t<b+n)
   tabla(i,1)=t;
   tabla(i,2)=disparo_sec(F,psi,x0,x1,t);
   t=t+n;
   i=i+1;
end
```

2.- Ejecuto el método y obtengo el resultado.

t	y(t)
-1.0000	-2.0000
-0.8000	-1.5119
-0.6000	-1.2158
-0.4000	-1.0636
-0.2000	-1.0074
-0.0000	-0.9990
0.2000	-0.9904
0.4000	-0.9337
0.6000	-0.7811
0.8000	-0.4853
1.0000	0.0000

NOTA: El problema era lineal, así que se podría haber usado un método para resolver problemas de contorno lineales.

2.- Teniendo el siguiente sistema no lineal:

$$\begin{cases} y'' = 1/8 * (32 + 2t^3 - y * y') \\ y(1) = 17 \\ y(3) = 43/3 \end{cases}$$

que tiene como solución exacta $y(x) = x^2 + 16/x$

a) Calcular y(2)
b) Obtener una tabla con lo que vale y(t) siendo t = (1,3).

Solución:

a)

1.- Definir el problema como un problema de valores iniciales:

A y se le llama y_1 → $y_1 = y$
A y' se le llama y_2 → $y_2 = y'$
$y'' = 1/8 * (32 + 2t^3 - y * y')$

$$\begin{cases} y_1' = y_2 \\ y_2' = 1/8 * (32 + 2t^3 - y_1 * y_2 \\ y_1(1) = 17 \\ y_2(1) = x \end{cases}$$

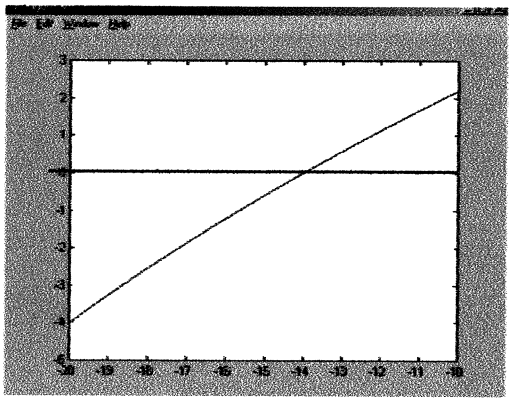
```
function z=F(t,y)
z(1)=y(2);
z(2)=1/8*(32+2*t.^3-y(1)*y(2));
```

2.- Definir la función psi

```
function z=psi(x)
y=runge_kutta('F',1,[17 x],3,50);
z=y(1)-43/3;
```

3.- Realizar un análisis gráfico de la función psi. Para ello utiliza la función dibujar que creé en el ejercicio anterior.

```
dibujar(-20,-10,'psi')
```



Al igual que en el ejercicio anterior, el problema sólo tiene una solución, que está aproximadamente entre –14 y –13.

4.- Ejecutar disparo para obtener y(2), dándole como aproximaciones -14 y -13

disparo_sec('F','psi' ,-14,-13,2) ⟶ **12.0000**

b)

Ejecuto el método disparo_tabla, teniendo cuidado de que el método use disparo_sec y no disparo_bis.

disparo_tabla('F','psi',1,3,-14,-13)

t	y(t)
1.0000	17.0000
1.2000	14.7733
1.4000	13.3886
1.6000	12.5600
1.8000	12.1289
2.0000	12.0000
2.2000	12.1127
2.4000	12.4267
2.6000	12.9138
2.8000	13.5543
3.0000	14.3333

3.- Use el algoritmo de disparo no lineal usando secante para aproximar el siguiente problema de contorno

$$\begin{cases} y'' = 2 * y^3 \\ y(1) = 1/4 \\ y(2) = 1/5 \end{cases}$$

definido en $1 \leq t \leq 2$. Obtener lo que vale el sistema para todos los t que lo definen y comparar el resultado con la solución real, que la proporciona $y(t) = 1 / (t + 3)$.

Solución:

1.- Definir el problema como un problema de valores iniciales:

A y se le llama y_1 → $y_1 = y$
A y' se le llama y_2 → $y_2 = y'$
$y'' = 2 * y^3$

$$\begin{cases} y_1' = y_2 \\ y_2' = 2 * y_1^3 \\ y_1(1) = 1/4 \\ y_2(1) = x \end{cases}$$

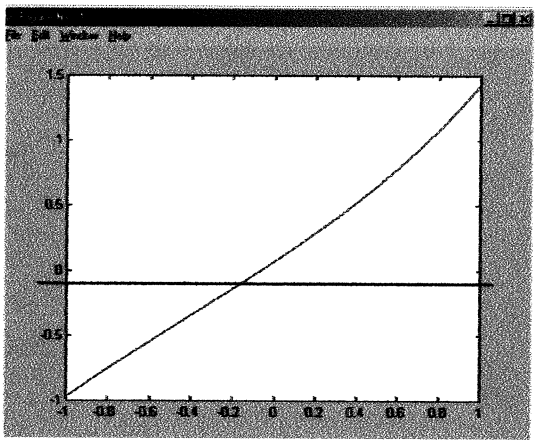
```
function z=F(t,y)
z(1)=y(2);
z(2)=2*y(1).^3;
```

2.- Definir la función psi

```
function z=psi(x)
y=runge_kutta('F',1,[1/4 x],2,100);
z=y(1)-1/5;
```

3.- Realizar un análisis gráfico de la función psi. Para ello utiliza la función dibujar que creé en el ejercicio anterior.

```
dibujar(-1,1,'psi')
```



Como puede verse la solución está muy cerca de 0.

4.- Adaptar disparo para este problema.

```
function z=disparo_sec(F,psi,x0,x1,t)
%x0,x1 son las dos aproximaciones para secante
sec=secante(psi,x0,x1,10^-10,20);
if (sec(2)==0) z=0;  %no tiene solución
else
   x=sec(1);
   y=runge_kutta(F,1,[1/4 x],t,100);
   z=y(1);
end
```

5.- Hacer que el método que proporciona la tabla llame a secante.

```
function tabla=disparo_tabla(F,psi,a,b,x0,x1)
tabla=zeros(10,2);
n=(b-a)/10;
i=1;
t=a;
while (t<b+n)
   tabla(i,1)=t;
   tabla(i,2)=disparo_sec(F,psi,x0,x1,t);
   t=t+n;
   i=i+1;
end
```

6.- Ejecutar el método para obtener el resultado.

Como x0 voy a pasarle -0.2 y como x1 voy a pasarle 0.

disparo_tabla('F','psi',1,2,-0.2,0)

t	y(t)
1.0000	0.2500
1.1000	0.2439
1.2000	0.2381
1.3000	0.2326
1.4000	0.2273
1.5000	0.2222
1.6000	0.2174
1.7000	0.2128
1.8000	0.2083
1.9000	0.2041
2.0000	0.2000

Método del disparo simple (utilizando bisección)

El problema que se tiene es el mismo, y se resuelve de la misma forma, la única diferencia está en que en vez de usar secante se usa bisección. Ahora el análisis gráfico sólo hay que realizarlo para saber por qué zona puede haber una raíz.

El código del método de bisección, adaptado para que indique si el método converge (devuelve en la segunda componente cero si no converge) es el siguiente:

```
function sol=bisecc(f,tol,a,b,nmax)
converge=0;
ya=feval(f,a);
yb=feval(f,b);
if ya*yb>0 break; end
n=1;
while n<nmax
   p=(a+b)/2;
   yp=feval(f,p);
   if yp==0
      a=p;
      b=p;
   elseif ya*yp<0
      b=p;
      yb=yp;
   else
      a=p;
      ya=yp;
   end
   if abs ((b-a)/2)<tol
      p=(a+b)/2;
      converge=1;
      break;
   else n=n+1;
   end
end
p=(a+b)/2;
yp=feval(f,p);
if (converge==0)
   n=nmax;
end
sol(1)=p;
if (n==nmax) sol(2)=0;
else
   sol(2)=n;
end
```

Ahora el código del método del disparo es distinto porque en vez de llamar a secante llama a bisección. El nuevo método es el siguiente:

```
function z=disparo_bis(F,psi,x0,x1,t)
%x0,x1 son las dos aproximaciones para secante
bis=bisecc('psi',10^-6,x0,x1,40);
if (bis(2)==0) z=0;  %no tiene solución
else
   x=bis(1);
   y=runge_kutta(F,-1,[-2 x],t,20);
   z=y(1);
end
```

EJERCICIOS:

1.- Resolver el siguiente sistema:

$$\begin{cases} y'' + 6t^2 + t^3 - 1 = 2y' + y + 6t \\ y(-1) = -2 \\ y(1) = 0 \end{cases}$$

c) Calcular y(3)
d) Obtener una tabla con lo que vale y(t) siendo t = (-1,1).

NOTA: Para probar los resultados tener en cuenta que $y(t) = t^3 - 1$

Solución:

a)

1.- Definir el problema como un problema de valores iniciales:

A y se le llama y_1 → $y_1 = y$
A y' se le llama y_2 → $y_2 = y'$
$y'' + 6t^2 + t^3 - 1 = 2y' + y + 6t$ → $y'' = 2y' + y + 6t - 6t^2 - t^3 + 1$

$$\begin{cases} y_1' = y_2 \\ y_2' = 2y_2 + y_1 + 6t - 6t^2 - t^3 + 1 \\ y_1(-1) = -2 \\ y_2(-1) = x \end{cases}$$

```
function z=F(t,y)
z(1)=y(2);
z(2)=2*y(2)+y(1)+6*t-6*t.^2-t.^3+1;
```

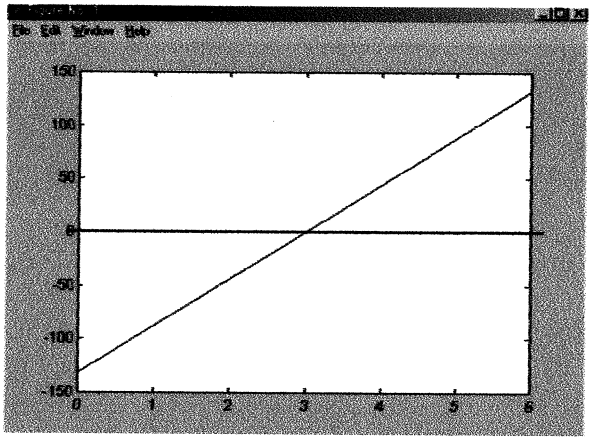
2.- Definir la función psi

```
function z=psi(x)
y=runge_kutta('F',-1,[-2 x],1,10);
z=y(1)-0;
```

3.- Realizar un análisis gráfico de la función psi

Para realizar el análisis gráfico creo una función que dibuje entre los puntos que se le pasen como parámetro, la función que se le indique.

```
function dibujar(x1,x2,f)
paso=x2-x1;
paso=paso/1000;
for i=1:1000
   a(i)=x1;
   b(i)=feval(f,x1);
   x1=x1+paso;
end
plot(a,b);
```



Según la gráfica la raíz está bien acotada entre 0 y 6, así que esos serán los valores que le pase al método del disparo.

4.- Ejecutar disparo para obtener el resultado, dándole como extremos del intervalo 0 y 6

disparo_bis('F','psi',0,6,3) ⟶ **25.9781**

b)

1.- Solo hay que hacer una pequeña modificación al método que mostraba la tabla antes. El cambio es que en vez de usar el método de disparo usando secante llame al método de disparo usando bisección.

```
function tabla=disparo_tabla(F,psi,a,b,x0,x1)
tabla=zeros(10,2);
n=(b-a)/10;
i=1;
t=a;
while (t<b+n)
   tabla(i,1)=t;
   tabla(i,2)=disparo_bis(F,psi,x0,x1,t);
   t=t+n;
   i=i+1;
end
```

2.- Ejecuto el método y obtengo el resultado.

t	y(t)
-1.0000	-2.0000
-0.8000	-1.5119
-0.6000	-1.2158
-0.4000	-1.0636
-0.2000	-1.0074
-0.0000	-0.9990
0.2000	-0.9904
0.4000	-0.9337
0.6000	-0.7811
0.8000	-0.4853
1.0000	0.0000

2.- Teniendo el siguiente sistema no lineal:

$$\begin{cases} y'' = 1/8 * (32 + 2t^3 - y * y') \\ y(1) = 17 \\ y(3) = 43/3 \end{cases}$$

que tiene como solución exacta $y(x) = x^2 + 16/x$

e) Calcular y(2)
f) Obtener una tabla con lo que vale y(t) siendo t = (1,3).

Solución:

a)

1.- Definir el problema como un problema de valores iniciales:

A y se le llama y_1 → $y_1 = y$
A y' se le llama y_2 → $y_2 = y'$
$y'' = 1/8 * (32 + 2t^3 - y * y')$

$$\begin{cases} y_1' = y_2 \\ y_2' = 1/8 * (32 + 2t^3 - y_1 * y_2 \\ y_1(1) = 17 \\ y_2(1) = x \end{cases}$$

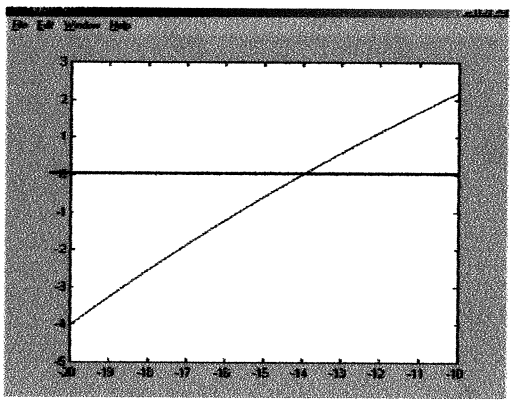
```
function z=F(t,y)
z(1)=y(2);
z(2)=1/8*(32+2*t.^3-y(1)*y(2));
```

2.- Definir la función psi

```
function z=psi(x)
y=runge_kutta('F',1,[17 x],3,50);
z=y(1)-43/3;
```

3.- Realizar un análisis gráfico de la función psi. Para ello utiliza la función dibujar que creé en el ejercicio anterior.

```
dibujar(-20,-10,'psi')
```



Según la gráfica, sólo hay una raíz, de tal forma que el problema sólo tendrá una solución.

4.- Ejecutar disparo para obtener y(2), dándole como aproximaciones extremos del intervalo los mismos que aparecen como extremos en el gráfico, -20, -10.

disparo_bis('F','psi',-20,-10,2) ⟶ **12.0000**

b)

Ejecuto el método disparo_tabla, teniendo cuidado de que el método use disparo_bis y no disparo_sec.

disparo_tabla('F','psi',1,3,-20,-10)

t	y(t)
1.0000	17.0000
1.2000	14.7733
1.4000	13.3886
1.6000	12.5600
1.8000	12.1289
2.0000	12.0000
2.2000	12.1127
2.4000	12.4267
2.6000	12.9138
2.8000	13.5543
3.0000	14.3333

3.- Aplique el método del disparo no lineal usando bisección para aproximar la solución al siguiente problema de contorno:

$$\begin{cases} y'' = y^3 - y * y' \\ y(1) = 1/2 \\ y(2) = 1/3 \end{cases}$$

definido en $1 \leq t \leq 2$. Obtener lo que vale el sistema para todos los t que lo definen y comparar el resultado con la solución real, que la proporciona $y(t) = 1 / (t + 1)$.

Solución:

1.- Definir el problema como un problema de valores iniciales:

A y se le llama y_1 → $y_1 = y$
A y' se le llama y_2 → $y_2 = y'$
$y'' = y^3 - y * y'$

$$\begin{cases} y_1' = y_2 \\ y_2' = y_1^3 - y_1 * y_2 \\ y_1(1) = 1/2 \\ y_2(1) = x \end{cases}$$

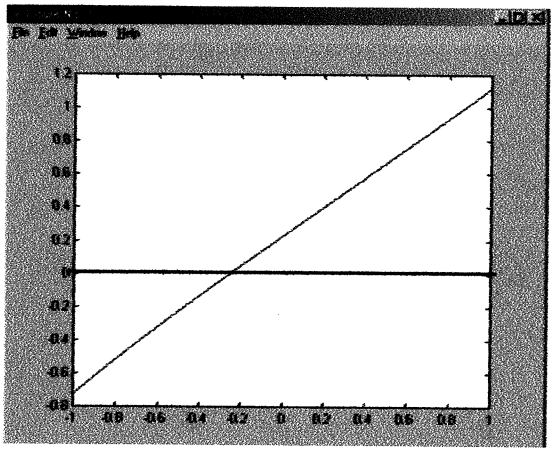
```
function z=F(t,y)
z(1)=y(2);
z(2)=y(1).^3-y(1)*y(2);
```

2.- Definir la función psi

```
function z=psi(x)
y=runge_kutta('F',1,[1/2 x],2,100);
z=y(1)-1/3;
```

3.- Realizar un análisis gráfico de la función psi. Para ello utiliza la función dibujar que creé en el ejercicio anterior.

```
dibujar(-1,1,'psi')
```



En este caso la solución está muy cerca de 0.

4.- Adaptar disparo para este problema.

```
function z=disparo_bis(F,psi,x0,x1,t)
%x0,x1 son las dos aproximaciones para secante
bis=bisecc('psi',10^-6,x0,x1,40);
if (bis(2)==0) z=0;  %no tiene solución
else
   x=bis(1);
   y=runge_kutta(F,1,[1/2 x],t,100);
   z=y(1);
end
```

5.- Hacer que el método que proporciona la tabla llame a secante.

```
function tabla=disparo_tabla(F,psi,a,b,x0,x1)
tabla=zeros(10,2);
n=(b-a)/10;
i=1;
t=a;
while (t<b+n)
   tabla(i,1)=t;
   tabla(i,2)= disparo_bis(F,psi,x0,x1,t);
   t=t+n;
   i=i+1;
end
```

6.- Ejecutar el método para obtener el resultado.

Como x0 voy a pasarle -1 y como x1 voy a pasarle 0.

disparo_tabla('F','psi',1,2,-1,0)

t	y(t)
1.0000	0.5000
1.1000	0.4762
1.2000	0.4545
1.3000	0.4348
1.4000	0.4167
1.5000	0.4000
1.6000	0.3846
1.7000	0.3704
1.8000	0.3571
1.9000	0.3448
2.0000	0.3333

Interpolación por Spline

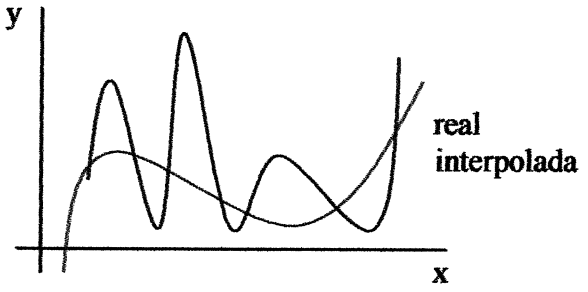
Se trata de obtener una función que ajuste los puntos de una tabla dada. Realiza lo mismo que la interpolación polinomial, pero de una forma mucho más exacta.

¿Cuándo es mejor la interpolación por Spline que la interpolación polinomial? Cuando la tabla de entrada es muy grande. Si se tienen muchos puntos, el polinomio interpolado tendrá un grado muy elevado, de tal forma que distará mucho de la función original.

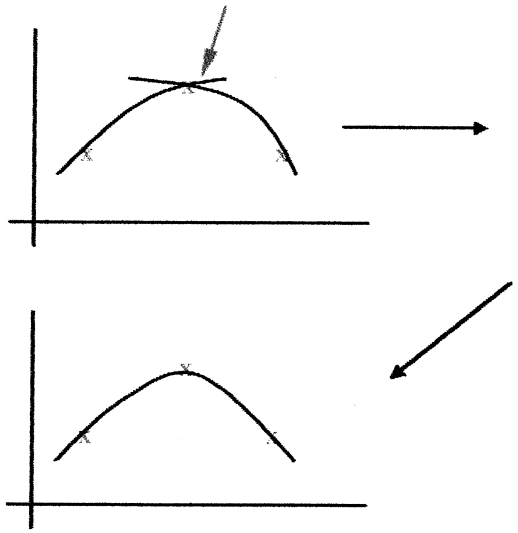
x	y = f(x)
x_1	y_1
x_2	y_2
...	...
x_M	y_M

↓

Muchos puntos



La solución a esto es construir una función polinomial a trozos, lo cual se hace uniendo los puntos dos a dos con polinomios cúbicos.



El problema es que en los puntos (que son las uniones de los polinomios) hay "esquinas", es decir, se crea una función en la que hay puntos que no son derivables. Para evitarlo hay que hacer coincidir en esos puntos las primeras y las segundas derivadas de los polinomios que se cruzan. A eso se le llama Spline.

Spline es una función definida en matlab, a la cual hay que pasarle la tabla (vector x, vector y) y el punto x en que se quiere saber lo que vale la función interpolada.

Ejemplo: Se tiene la siguiente tabla de la ecuación y = x2

x	y
-3	9
-1	1
0	0
1	1
3	3

Si se hace la llama:
spline ([-3 -1 0 1 3],[9 1 0 1 9],2)
El resultado es: **4**

EJERCICIOS:

1.- Se tienen las siguientes tablas de puntos (x,y), obtenidos de la función:

$$y = e^x + 2x * sen(x)$$

x	y
0.0000 (0)	1.0000
1.5708 (1/2π)	7.9521
3.1416 (π)	23.1407
4.7124 (3/2π)	101.8930
6.2832 (2π)	535.4917

x	y
0.0000 (0)	1.0000
0.6283 (1/5π)	2.6131
1.2566 (2/5π)	5.9039
1.8850 (3/5π)	10.1715
2.5133 (4/5π)	15.2998
3.1416 (π)	23.1407
3.7699 (6/5π)	38.9444
4.3982 (7/5π)	72.9409
5.0265 (8/5)	142.8450
5.6549 (9/5π)	279.0307
6.2832 (2π)	535.4917

x	y
0.0000 (0)	1.0000
0.3142 (1/10π)	1.5633
0.6283 (2/10π)	2.6131
0.9425 (3/10π)	4.0913
1.2566 (4/10π)	5.9039
1.5708 (5/10π)	7.9521
1.8850 (6/10π)	10.1715
2.1991 (7/10π)	12.5753
2.5133 (8/10π)	15.2998
2.8274 (9/10π)	18.6495
3.1416 (π)	23.1407
3.4558 (11/10π)	29.5463
3.7699 (12/10π)	38.9444
4.0841 (13/10π)	52.7785
4.3982 (14/10π)	72.9409
4.7124 (15/10π)	101.8930
5.0265 (16/10π)	142.8450
5.3407 (17/10π)	200.0188
5.6549 (18/10π)	279.0307
5.9690 (19/10π)	387.4355
6.2832 (2π)	535.4917

Para cada tabla mostrar un gráfico con la función original (verde), el polinomio interpolado de forma normal (azul), y el polinomio interpolado usando Spline (rojo). Calcular también la bondad de los polinomios obtenidos para obtener la exactitud de los ajustes.

a) Interpolar usando puntos igualmente espaciados.
b) Interpolar usando los nodos de chebyshev.

a)

1.- Definir la función original para poder dibujarla en verde

```
function y=func(x)
y=exp(x)+2.*x.*sin(x);
```

2- Crear una función que dibuje las tres funciones.

NOTA: Para la interpolación polinomial normal voy a usar Lagrange.

```
function bondad=nuevo_dibujo(x,y)
n=length(x);
bondad(1)=0;
bondad(2)=0;
parte=abs((x(1)-x(n))/50);
for i=1:51
   if i==1
      z(i)=x(1);
      zzz(i)=x(1);
   else
      z(i)=parte+z(i-1);
      zzz(i)=parte+zzz(i-1);
   end

   w(i)=lagrange(x,y,z(i));
   dif=abs(feval('func',z(i))-w(i));
   if (dif>bondad(1))
      bondad(1)=dif;
   end

   www(i)=spline(x,y,zzz(i));
   dif=abs(feval('func',zzz(i))-www(i));
   if (dif>bondad(2))
      bondad(2)=dif;
   end

end

zz=0:0.03:7;
ww=func(zz);

plot (zz,ww,'*g',z,w,'b',zzz,www,'.r');

%zz y ww son para la función original
%z y w son para la función interpolada con lagrange
%zzz y www son para la función interpolada con spline
%bondad(1) devuelve la bondad entre la función ...
%...interpolada con lagrange y la función original
%bondad(2) devuelve la bondad entre la función ...
%...interpolada con spline y la función original
```

bondad del polinomo de lagrange

bondad usando spline

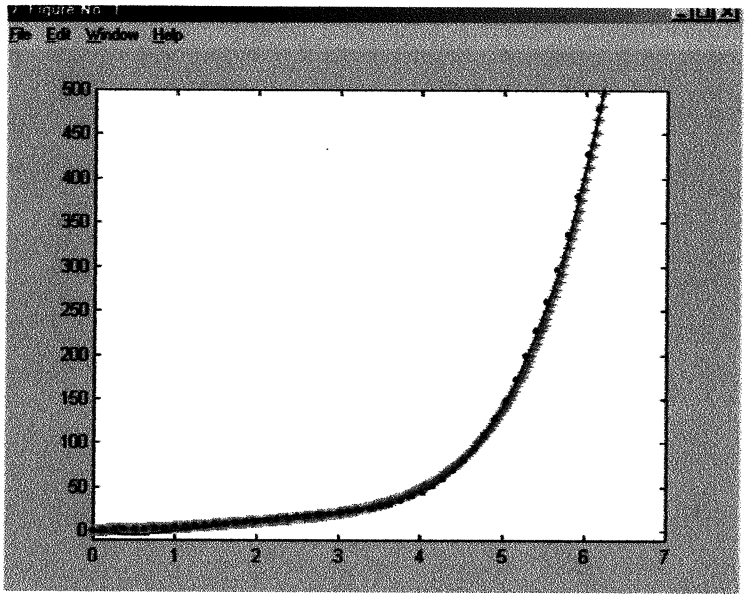
TABLA 1 (5 puntos)

a.- Definir los vectores x e y

```
x=[0 1/2*pi pi 3/2*pi 2*pi];   y=func(x)
```

b.- Crear la gráfica y obtener la bondad

```
nuevo_dibujo(x,y); axis ([0,7,-10,500]);
```



12.3920 ▼ lagrange

18.4300 ▼ spline

Viendo los valores de bondad es evidente que lagrange ajusta mejor que spline, aunque en el gráfico no se aprecie demasiado bien.

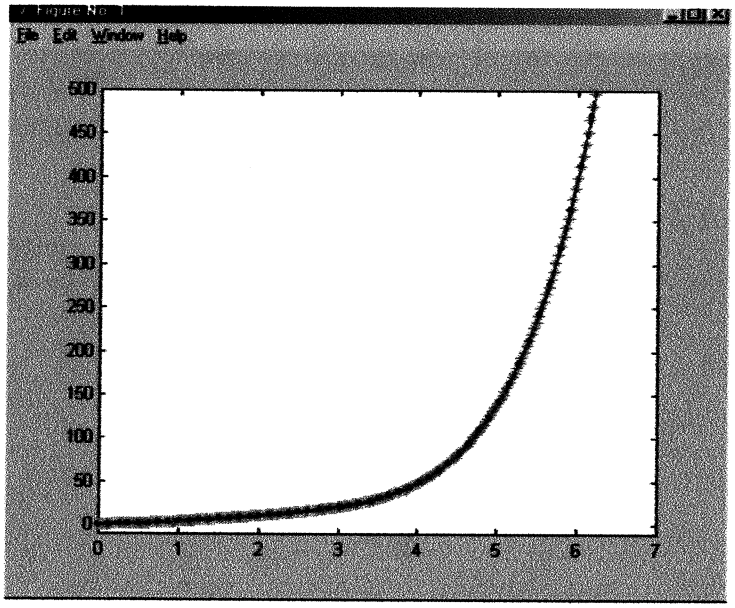
TABLA 2 (11 puntos)

a.- Definir los vectores x e y

x=[0 1/5*pi 2/5*pi 3/5*pi 4/5*pi pi 6/5*pi 7/5*pi 8/5*pi 9/5*pi 2*pi]
y=**func**(x)

b.- Crear la gráfica y obtener la bondad

nuevo_dibujo(x,y); axis ([0,7,-10,500]);



0.0026 lagrange

1.1760 spline

Aunque ahora la diferencia es más pequeña, de nuevo lagrange ajusta mejor que spline.

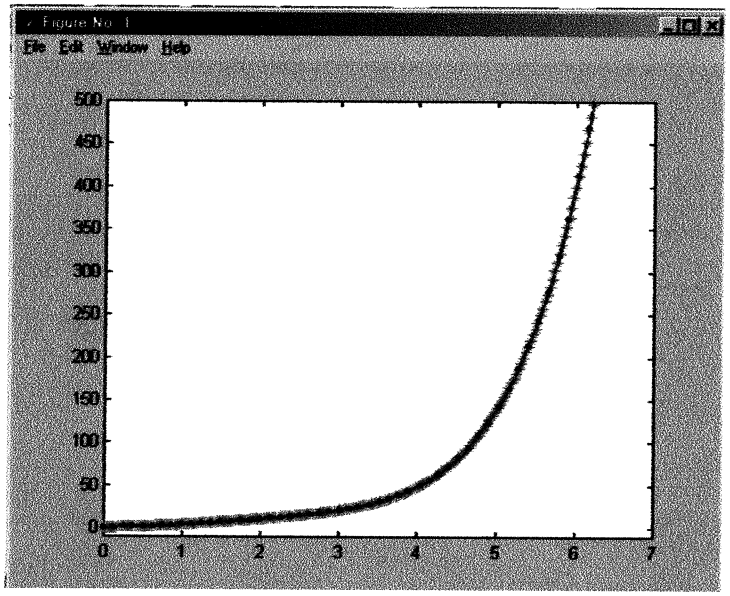
TABLA 3 (21 puntos)

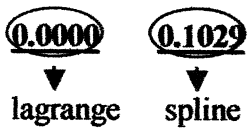
a.- Definir los vectores x e y

```
x=[0 1/10*pi 2/10*pi 3/10*pi 4/10*pi 5/10*pi 6/10*pi 7/10*pi 8/10*pi
   9/10*pi pi 11/10*pi 12/10*pi 13/10*pi 14/10*pi 15/10*pi 16/10*pi
   17/10*pi 18/10*pi 19/10*pi 2*pi]
y=func(x)
```

b.- Crear la gráfica y obtener la bondad

```
nuevo_dibujo(x,y); axis ([0,7,-10,500]);
```





De nuevo lagrange proporciona mejor bondad, sin embargo ahora spline tiene también una bondad muy pequeña.

De los resultados de las tres gráficas anteriores se puede deducir que spline se va haciendo más eficiente a medida que se van teniendo mas puntos en la tabla.

b)

Como ya está definida la función original, la función que dibuja los polinomios y la función que calcula los nodos de chebyshev, solo habría que utilizarlas.

El código del método de chebyshev es el siguiente:

```
function [x]=cheb(n,a,b) %n es n° de nodos
format long;
const=pi/(2*(n-1)+2);
for k=1:n
   d=(2*k-1)*const;
   x(n+1-k)=cos(d);
end
x=(((b-a)*x/2)+((b+a)/2));
```

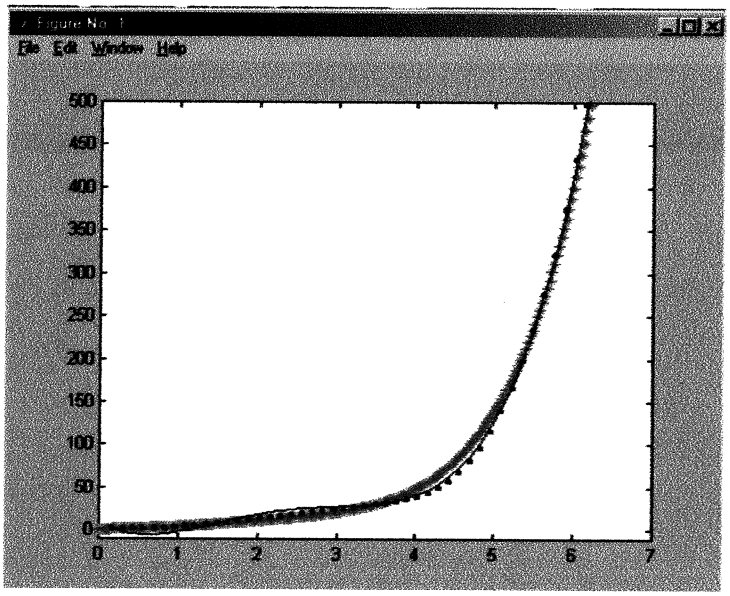
5 NODOS

a.- Definir los vectores x e y

x=cheb(5,0,7)
y=func(x)

b.- Crear la gráfica y obtener la bondad

nuevo_dibujo(x,y); axis ([0,7,-10,500]);



0.0000000003172 — lagrange

0.10287796973893 — spline

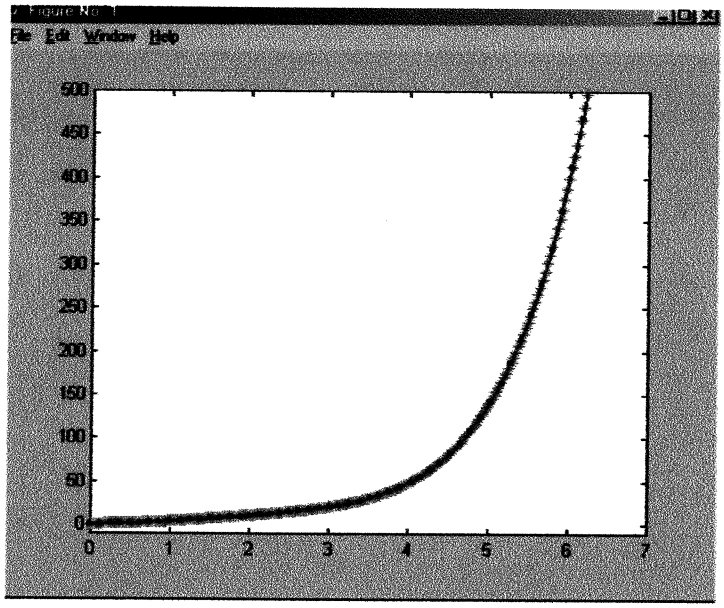
11 NODOS

a.- Definir los vectores x e y

```
x=cheb(11,0,7)
y=func(x)
```

b.- Crear la gráfica y obtener la bondad

```
nuevo_dibujo(x,y); axis ([0,7,-10,500]);
```



0.00000000003172 — lagrange

0.10287796973893 — spline

El resultado es el mismo que en el caso anterior.

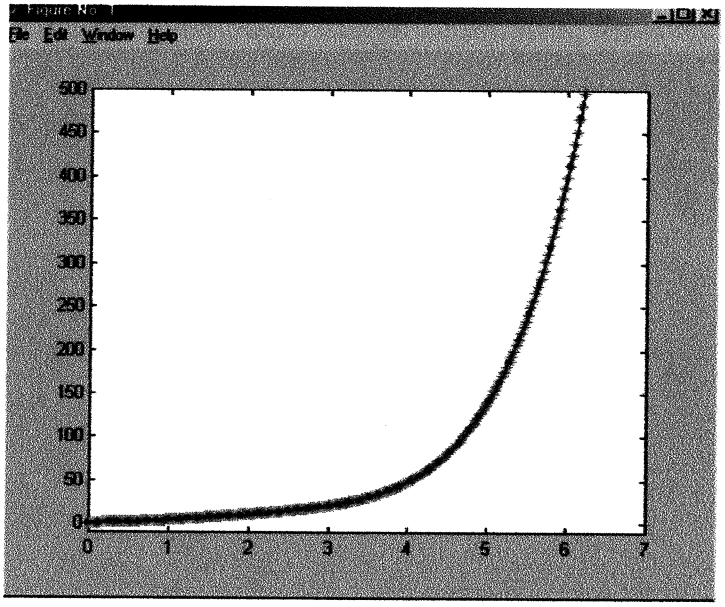
21 NODOS

a.- Definir los vectores x e y

```
x=cheb(21,0,7)
y=func(x)
```

b.- Crear la gráfica y obtener la bondad

```
nuevo_dibujo(x,y); axis ([0,7,-10,500]);
```



0.00000000003172 (lagrange)

0,10287796973893 (spline)

En los tres casos se obtiene la misma bondad, y en todos es mejor lagrange.

. . . . . .

Como puede verse, en este ejemplo no se ve la eficacia de spline, así que voy a realizar otro ejemplo con una función más compleja.

2.- Tomando la siguiente función:

$$y = \text{sen}(x) * \cos(x) + 2x * \text{sen}(x) - \cos(x) \qquad x \in [0, 6\pi]$$

a) Interpolar usando puntos igualmente espaciados.
b) Interpolar usando los nodos de chebyshev.

En los dos apartados usar interpolación polinomial e interpolación con spline. Mostrar en cada caso gráficas con la función original en verde, la interpolación normal en azul y la interpolación con spline en rojo. Calcular la bondad en cada caso.

Solución:

1.- Definir la función:

```
function y=func(x)
y=sin(x)*cos(x)+2*x*sin(x)-cos(x);
```

a)

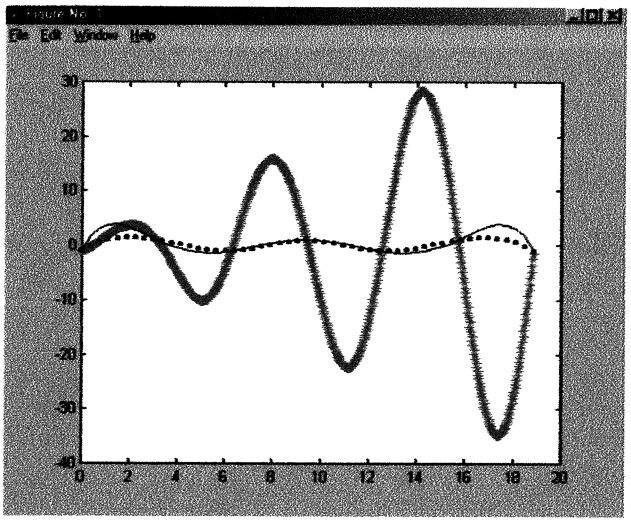
7 PUNTOS

2.- Definir los vectores x e y

```
x= [0 pi 2*pi 3*pi 4*pi 5*pi 6*pi]
y=func(x)
```

3.- Crear la gráfica y obtener la bondad

```
nuevo_dibujo(x,y)
```



38.5922 ▼ lagrange

36.0666 ▼ spline

Aunque las dos interpolaciones son muy malas es mejor la hecha con spline.

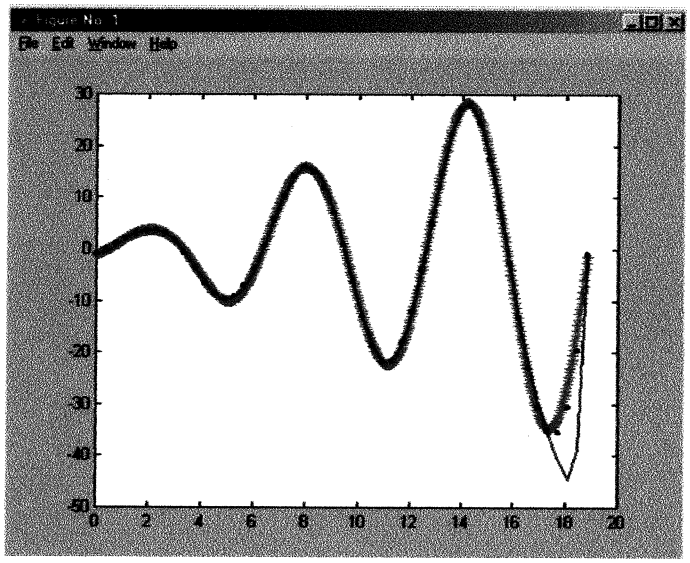
13 PUNTOS

2.- Definir los vectores x e y

x= [0 1/2*pi pi 3/2*pi 2*pi 5/2*pi 3*pi 7/2*pi 4*pi 9/2*pi 5*pi 11/2*pi 6*pi]
y=**func**(x)

3.- Crear la gráfica y obtener la bondad

nuevo_dibujo(x,y)



23.7729 lagrange

4.6040 spline

En el gráfico se puede ver que se ha mejorado, mucho y que ambas aproximan mejor, pero mirando la bondad se ve claramente que spline aproxima mucho mejor.

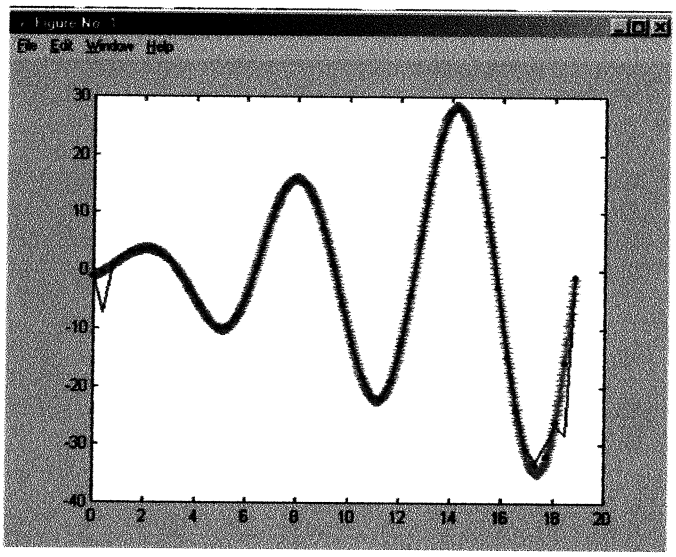
25 PUNTOS

2.- Definir los vectores x e y

```
x=[0 1/4*pi 2/4*pi 3/4*pi pi 5/4*pi 6/4*pi 7/4*pi 2*pi 9/4*pi 10/4*pi 11/4*pi
   3*pi 13/4*pi 14/4*pi 15/4*pi 4*pi 17/4*pi 18/4*pi 19/4*pi 5*pi 21/4*pi
   22/2*pi 23/4*pi 6*pi]
y=func(x)
```

3.- Crear la gráfica y obtener la bondad

```
nuevo_dibujo(x,y)
```



13.4290 ▼ lagrange

1.3772 ▼ spline

En este caso el error con spline es bastante pequeño, sin embargo la interpolación polinomial sigue dando un error muy elevado.

. . . . . .

Con este ejemplo si se puede ver claramente que si se dispone de una tabla con muchos puntos de una función compleja, la interpolación con splain es más eficiente que la interpolación polinomial.

b)

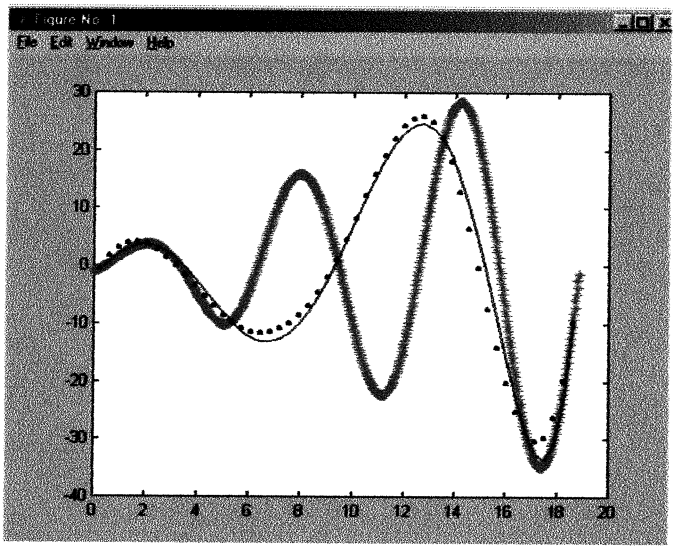
7 NODOS

2.- Definir los vectores x e y

```
x=cheb(7,0,6*pi)
y=func(x)
```

3.- Crear la gráfica y obtener la bondad

```
nuevo_dibujo(x,y)
```



40.8794 → lagrange

41.8779 → spline

La aproximación en ambos casos es muy mala.

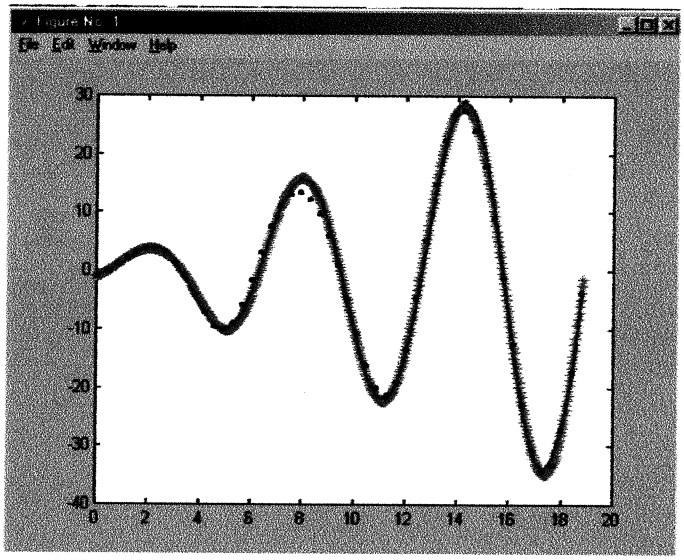
13 NODOS

2.- Definir los vectores x e y

```
x=cheb(13,0,6*pi)
y=func(x)
```

3.- Crear la gráfica y obtener la bondad

```
nuevo_dibujo(x,y)
```



1.6982 ▼ lagrange

2.8219 ▼ spline

Ambos polinomios interpolan bastante bien, sin embargo lagrange hace una aproximación más exacta.

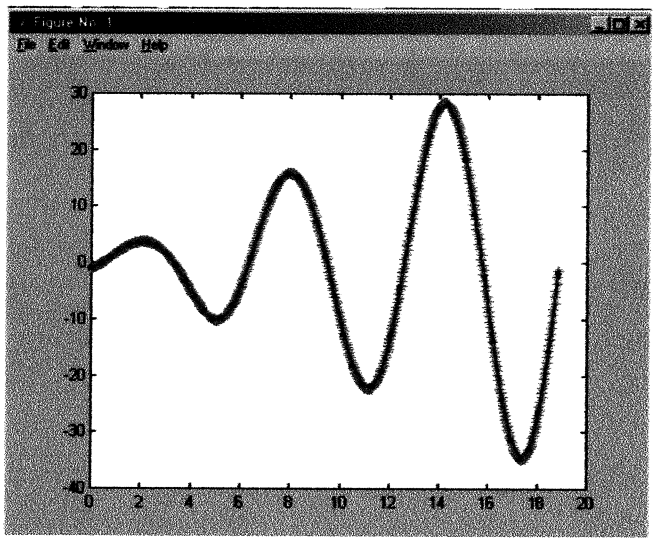
25 NODOS

2.- Definir los vectores x e y

```
x=cheb(25,0,6*pi)
y=func(x)
```

3.- Crear la gráfica y obtener la bondad

```
nuevo_dibujo(x,y)
```



0.0047 ▼ lagrange

0.1861 ▼ spline

La aproximación es muy buena en este caso, pero parece que spline lo hace un poco peor.

. . . . . .

Teniendo en cuenta los resultados obtenidos en este apartado parece ser que usando los nodos de chebyshev, a medida que se calculan mas nodos va siendo más eficiente la interpolación polinomial que la interpolación con spline.

3.- Usando la función $y = x^2 * \cos(x)$, comprobar la eficacia de la interpolación por Spline resolviendo los siguientes apartados:

a) Dibujar la función original frente a la función interpolada y obtener la bondad del ajuste utilizando para ello tablas de 6 puntos, 11 puntos, 21 puntos y 41 puntos.

b) Realizar lo mismo con la derivada de la función.

c) Lo mismo con la segunda derivada de la función original.

Solución:

a)

1.- Definir la función.

```
function y=func(x)
y=x.^2.*cos(x);
```

2.- Definir una nueva función que dibuje la función original frente a la interpolada y que calcule la bondad.

```
function bondad=dibujo_bondad_spline(x,y)
n=length(x);
bondad=0;
parte=abs((x(1)-x(n))/50);
for i=1:51
   if i==1
      zzz(i)=x(1);
   else
      zzz(i)=parte+zzz(i-1);
   end

   www(i)=spline(x,y,zzz(i));
   dif=abs(feval('func',zzz(i))-www(i));
   if (dif>bondad)
      bondad=dif;
   end

end

zz=0:0.01:2*pi;
ww=func(zz);

plot (zz,ww,'*g',zzz,www,'.r');

%zz y ww son para la función original
%zzz y www son para la función interpolada con spline
%bondad devuelve la bondad entre la función ...
%...interpolada con spline y la función original
```

3.- Obtener el resultado.

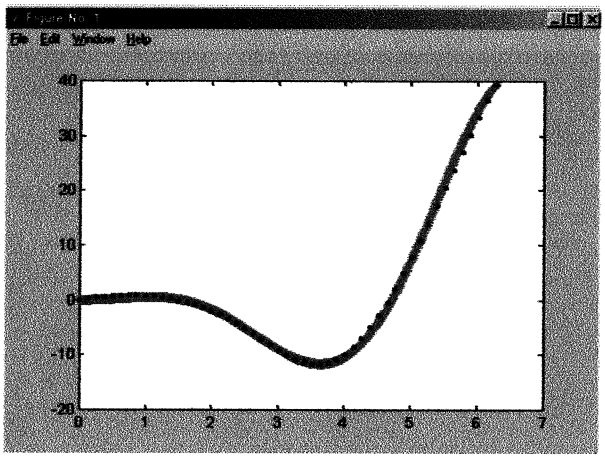
Con 6 puntos:

- Obtener el vector x y el vector y.

```
x=[0 2/5*pi 4/5*pi 6/5*pi 8/5*pi 2*pi];
y=func(x);
```

- Visualizar la gráfica y obtener la bondad.

```
dibujo_bondad_spline(x,y)
```



2.3322

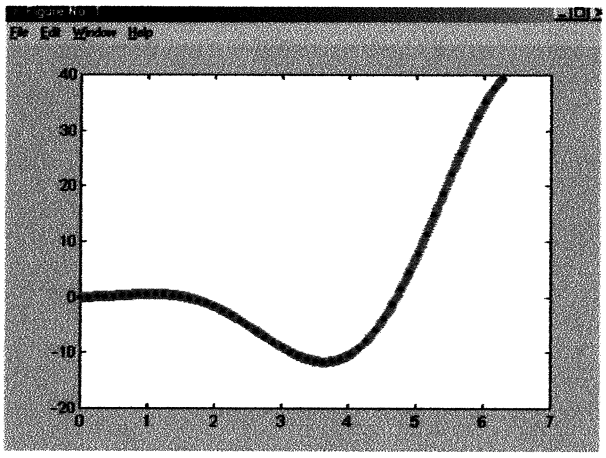
Con 11 puntos:

- Obtener el vector x y el vector y.

```
x=[0 1/5*pi 2/5*pi 3/5*pi 4/5*pi pi 6/5*pi 7/5*pi 8/5*pi 9/5*pi 2*pi];
y=func(x);
```

- Visualizar la gráfica y obtener la bondad.

```
dibujo_bondad_spline(x,y)
```



0.0503

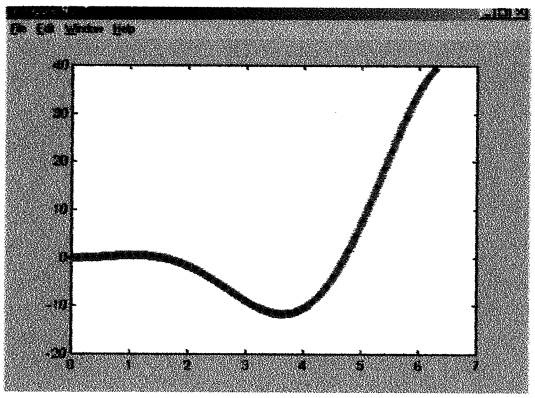
Con 21 puntos:

- Obtener el vector x y el vector y.

```
x=[0 1/10*pi 2/10*pi 3/10*pi 4/10*pi 5/10*pi 6/10*pi 7/10*pi 8/10*pi 9/10*pi pi
   11/10*pi 12/10*pi 13/10*pi 14/10*pi 15/10*pi 16/10*pi 17/10*pi 18/10*pi
   19/10*pi 2*pi]
y=func(x);
```

- Visualizar la gráfica y obtener la bondad.

dibujo_bondad_spline(x,y)



0.0028

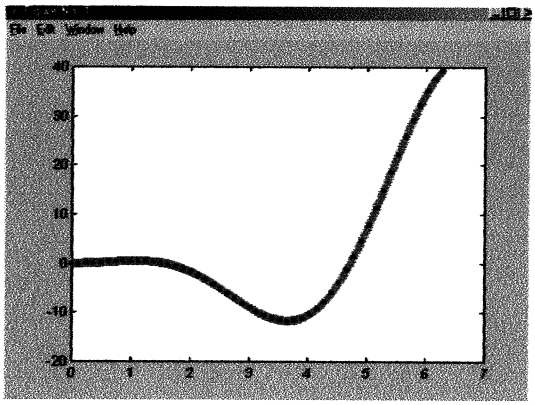
Con 41 puntos:

- Obtener el vector x y el vector y.

```
x=[0 1/20*pi 2/20*pi 3/20*pi 4/20*pi 5/20*pi 6/20*pi 7/20*pi 8/20*pi 9/20*pi
   10/20*pi 11/20*pi 12/20*pi 13/20*pi 14/20*pi 15/20*pi 16/20*pi 17/20*pi
   18/20*pi 19/20*pi pi 21/20*pi 22/20*pi 23/20*pi 24/20*pi 25/20*pi 26/20*pi
   27/20*pi 28/20*pi 29/20*pi 30/20*pi 31/20*pi 32/20*pi 33/20*pi 34/20*pi
   35/20*pi 36/20*pi 37/20*pi 38/20*pi 39/20*pi 2*pi];
y=func(x);
```

- Visualizar la gráfica y obtener la bondad.

dibujo_bondad_spline(x,y)



0.0001

b)

1.- Definir la derivada de la función.

```
function y=func_der(x)
y=2.*x.*cos(x)-x.^2.*sin(x);
```

2.- Definir una función que calcule el spline en un punto dada una tabla.

```
function z=func_spline(x,y,p0)
z=spline(x,y,p0);
```

3.- Definir una nueva función que dibuje la derivada de la función original frente a la interpolada de es la función y que calcule la bondad.

```
function bondad=dibujo_bondad_spline_der(x,y)
n=length(x);
bondad=0;
parte=abs((x(1)-x(n))/50);
for i=1:51
   if i==1
      zzz(i)=x(1);
   else
      zzz(i)=parte+zzz(i-1);
   end
   www(i)=derivada_nueva('func_spline',x,y,0.01,10^-6,10,zzz(i));
   %www(i) almacena la derivada de una función spline
   dif=abs(feval('func_der',zzz(i))-www(i));
   if (dif>bondad)
      bondad=dif;
   end
end
zz=0:0.01:2*pi;
ww=func_der(zz);
plot (zz,ww,'*g',zzz,www,'.r');
```

NOTA: Como puede verse, el método utiliza otro método que es derivada_nueva, que lo que hace es calcular la derivada de la función spline. Su código es el siguiente:

```
function sol=derivada_nueva(f,x,y,h,tol,max,x0)
r=zeros(max,max);
r(1,1)=(feval(f,x,y,(x0+h))-feval(f,x,y,(x0-h)))/(2*h);
sol=r(1,1);
for i=1:max
   h=h/2;
   r(i+1,1)=(feval(f,x,y,(x0+h))-feval(f,x,y,(x0-h)))/(2*h);
   sol=r(i+1,1);
   for k=1:i
      r(i+1,k+1)=r(i+1,k)+(((r(i+1,k))-(r(i,k)))/((4^k)-1));
      sol=r(i+1,k+1);
   end
   err=abs(r(i,i)-r(i+1,i+1));
   if (err<=tol)
      break;
   end
end
```

3.- Obtener el resultado.

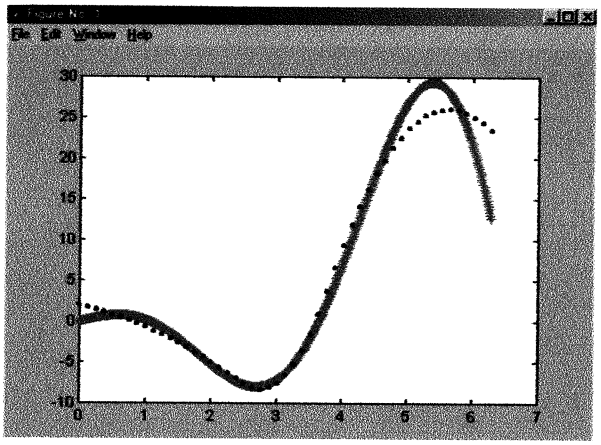
Con 6 puntos:

- Obtener el vector x y el vector y.

```
x=[0 2/5*pi 4/5*pi 6/5*pi 8/5*pi 2*pi];
y=func(x);
```

- Visualizar la gráfica y obtener la bondad.

dibujo_bondad_spline_der(x,y)



10.8302

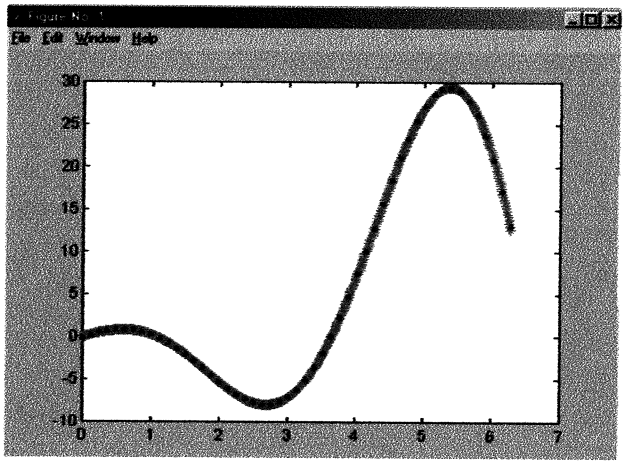
Con 11 puntos:

- Obtener el vector x y el vector y.

```
x=[0 1/5*pi 2/5*pi 3/5*pi 4/5*pi pi 6/5*pi 7/5*pi 8/5*pi 9/5*pi 2*pi];
y=func(x);
```

- Visualizar la gráfica y obtener la bondad.

dibujo_bondad_spline_der(x,y)



0.4056

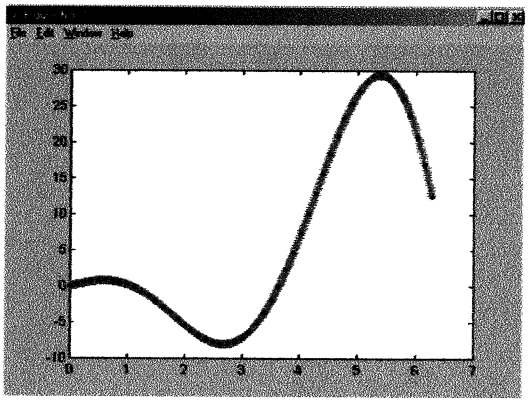
Con 21 puntos:

- Obtener el vector x y el vector y.

```
x=[0 1/10*pi 2/10*pi 3/10*pi 4/10*pi 5/10*pi 6/10*pi 7/10*pi 8/10*pi 9/10*pi pi
   11/10*pi 12/10*pi 13/10*pi 14/10*pi 15/10*pi 16/10*pi 17/10*pi 18/10*pi
   19/10*pi 2*pi];
y=func(x);
```

- Visualizar la gráfica y obtener la bondad.

dibujo_bondad_spline_der(x,y)



0.0587

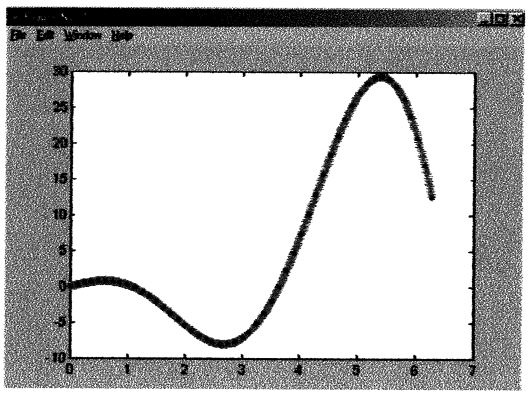
Con 41 puntos:

- Obtener el vector x y el vector y.

```
x=[0 1/20*pi 2/20*pi 3/20*pi 4/20*pi 5/20*pi 6/20*pi 7/20*pi 8/20*pi 9/20*pi
   10/20*pi 11/20*pi 12/20*pi 13/20*pi 14/20*pi 15/20*pi 16/20*pi 17/20*pi
   18/20*pi 19/20*pi pi 21/20*pi 22/20*pi 23/20*pi 24/20*pi 25/20*pi 26/20*pi
   27/20*pi 28/20*pi 29/20*pi 30/20*pi 31/20*pi 32/20*pi 33/20*pi 34/20*pi
   35/20*pi 36/20*pi 37/20*pi 38/20*pi 39/20*pi 2*pi];
y=func(x);
```

- Visualizar la gráfica y obtener la bondad.

dibujo_bondad_spline_der(x,y)



0.0124

c)

1.- Definir la derivada segunda de la función.

```
function y=func_der2(x)
y=2.*cos(x)-4.*x.*sin(x)-x.^2.*cos(x);
```

2.- Definir una nueva función que dibuje la derivada segunda de la función original frente a la interpolada de la función y que calcule la bondad.

```
function bondad=dibujo_bondad_spline_der2(x,y)
n=length(x);
bondad=0;
parte=abs((x(1)-x(n))/50);
for i=1:51
   if i==1
      zzz(i)=x(1);
   else
      zzz(i)=parte+zzz(i-1);
   end
   www(i)=derivada2_nueva('func_spline',x,y,0.01,10^-6,10,zzz(i));
   %www(i) almacena la derivada de una función spline
   dif=abs(feval('func_der2',zzz(i))-www(i));
   if (dif>bondad)
      bondad=dif;
   end
end
zz=0:0.01:2*pi;
ww=func_der2(zz);
plot (zz,ww,'*g',zzz,www,'.r');
```

NOTA: Como puede verse, el método utiliza otro método que es derivada2_nueva, que lo que hace es calcular la derivada segunda de la función spline. Su código es el siguiente:

```
function sol=derivada2_nueva(f,x,y,h,tol,max,x0)
r=zeros(max,max);
r(1,1)=(feval(f,x,y,(x0+h))-2*feval(f,x,y,x0)+feval(f,x,y,(x0-h)))
       /(h^2);
sol=r(1,1);
for i=1:max
   h=h/2;
   r(i+1,1)=(feval(f,x,y,(x0+h))-2*feval(f,x,y,x0)+feval(f,x,y,
            (x0-h)))/(h^2);
   sol=r(i+1,1);
   for k=1:i
      r(i+1,k+1)=r(i+1,k)+(((r(i+1,k))-(r(i,k)))/((4^k)-1));
      sol=r(i+1,k+1);
   end
   err=abs(r(i,i)-r(i+1,i+1));
   if (err<=tol)
      break;
   end
end
```

3.- Obtener el resultado.

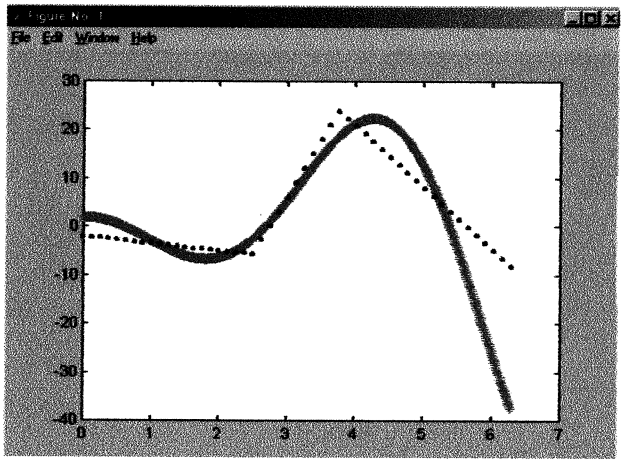
Con 6 puntos:

- Obtener el vector x y el vector y.

```
x=[0 2/5*pi 4/5*pi 6/5*pi 8/5*pi 2*pi];
y=func(x);
```

- Visualizar la gráfica y obtener la bondad.

```
dibujo_bondad_spline_der2(x,y)
```



29.2512

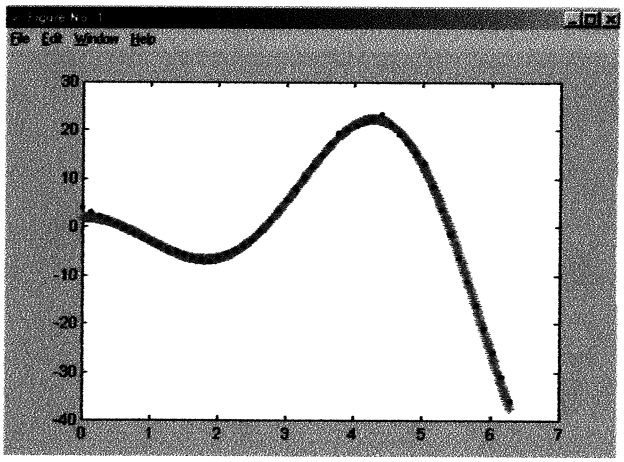
Con 11 puntos:

- Obtener el vector x y el vector y.

```
x=[0 1/5*pi 2/5*pi 3/5*pi 4/5*pi pi 6/5*pi 7/5*pi 8/5*pi 9/5*pi 2*pi];
y=func (x);
```

- Visualizar la gráfica y obtener la bondad.

```
dibujo_bondad_spline_der2(x,y)
```



1.9842

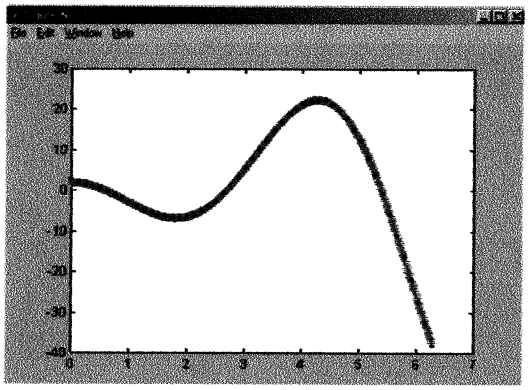
Con 21 puntos:

- Obtener el vector x y el vector y.

```
x=[0 1/10*pi 2/10*pi 3/10*pi 4/10*pi 5/10*pi 6/10*pi 7/10*pi 8/10*pi 9/10*pi pi
   11/10*pi 12/10*pi 13/10*pi 14/10*pi 15/10*pi 16/10*pi 17/10*pi 18/10*pi
   19/10*pi 2*pi];
y=func(x);
```

- Visualizar la gráfica y obtener la bondad.

dibujo_bondad_spline_der2(x,y)



0.7466

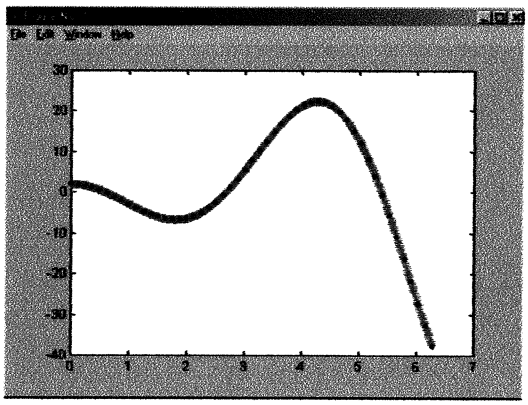
Con 41 puntos:

- Obtener el vector x y el vector y.

```
x=[0 1/20*pi 2/20*pi 3/20*pi 4/20*pi 5/20*pi 6/20*pi 7/20*pi 8/20*pi 9/20*pi
   10/20*pi 11/20*pi 12/20*pi 13/20*pi 14/20*pi 15/20*pi 16/20*pi 17/20*pi
   18/20*pi 19/20*pi pi 21/20*pi 22/20*pi 23/20*pi 24/20*pi 25/20*pi 26/20*pi
   27/20*pi 28/20*pi 29/20*pi 30/20*pi 31/20*pi 32/20*pi 33/20*pi 34/20*pi
   35/20*pi 36/20*pi 37/20*pi 38/20*pi 39/20*pi 2*pi];
y=func(x);
```

- Visualizar la gráfica y obtener la bondad.

dibujo_bondad_spline_der2(x,y)



0.3258

4.- Usando la función y = sen(x) * cos(x) + 2cos(x) con $0 \leq x \leq 9$:

a) Dibujar la función original frente a la función interpolada y obtener la bondad del ajuste utilizando para ello tablas de 10 puntos, 19 puntos y 37 puntos.

b) Usando las mismas cantidades de puntos dibujar una gráfica que muestre la derivada de la función original frente a una derivada aproximada usando Spline.

c) Realizar lo mismo con la segunda derivada de la función original.

Solución:

a)

1.- Definir la función.

```
function y=func(x)
y=sin(x).*cos(x)+2.*cos(x);
```

2.- Obtener el resultado.

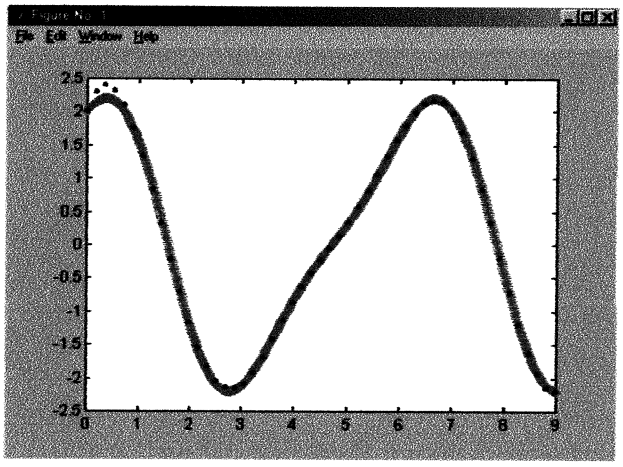
<u>Con 10 puntos:</u>

- Obtener el vector x y el vector y.

```
x=[0 1 2 3 4 5 6 7 8 9];
y=func(x);
```

- Visualizar la gráfica y obtener la bondad.

dibujo_bondad_spline(x,y)



<u>0.1999</u>

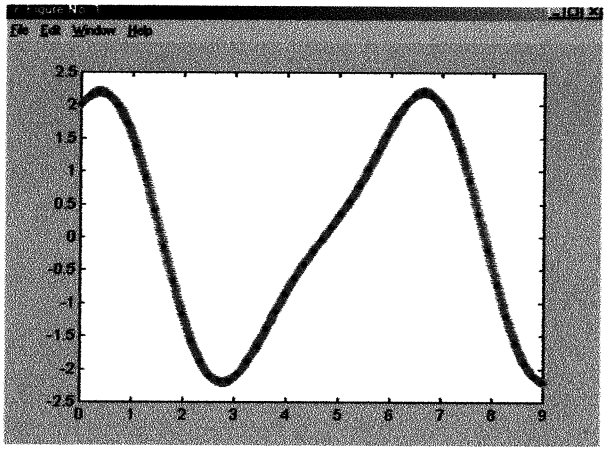
Con 19 puntos:

- Obtener el vector x y el vector y.

```
x=[0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5 5.5 6 6.5 7 7.5 8 8.5 9];
y=func(x);
```

- Visualizar la gráfica y obtener la bondad.

```
dibujo_bondad_spline(x,y)
```



0.0153

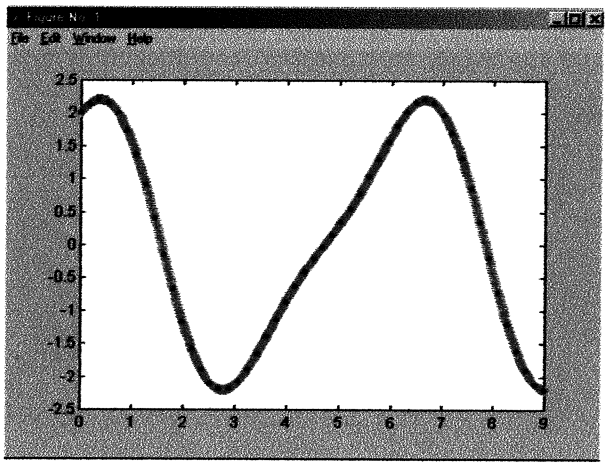
Con 37 puntos:

- Obtener el vector x y el vector y.

```
x=[0 0.25 0.5 0.75 1 1.25 1.5 1.75 2 2.25 2.5 2.75 3 3.25 3.5 3.75 4 4.25 4.5
   4.75 5 5.25 5.5 5.75 6 6.25 6.5 6.75 7 7.25 7.5 7.75 8 8.25 8.5 8.75 9]
y=func(x);
```

- Visualizar la gráfica y obtener la bondad.

```
dibujo_bondad_spline(x,y)
```



0.0005

b)

1.- Definir la derivada de la función función.

```
function y=func_der(x)
y=cos(x).*cos(x)-sin(x).*sin(x)-2*sin(x);
```

2.- Obtener el resultado.

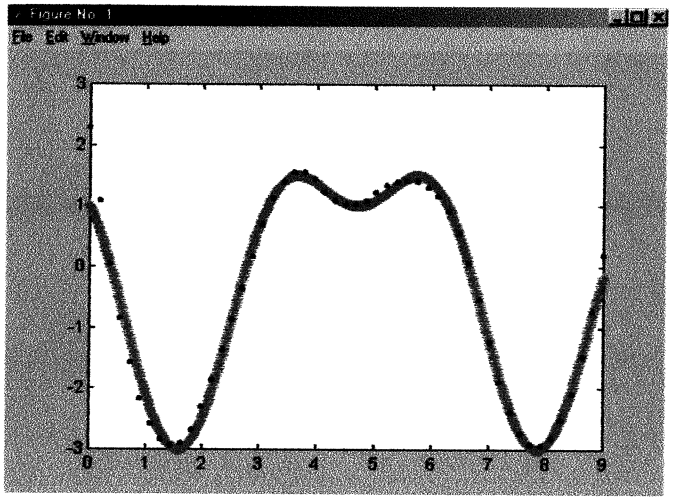
Con 10 puntos:

- Obtener el vector x y el vector y.

```
x=[0 1 2 3 4 5 6 7 8 9];
y=func(x);
```

- Visualizar la gráfica y obtener la bondad.

dibujo_bondad_spline_der(x,y)



1.2952

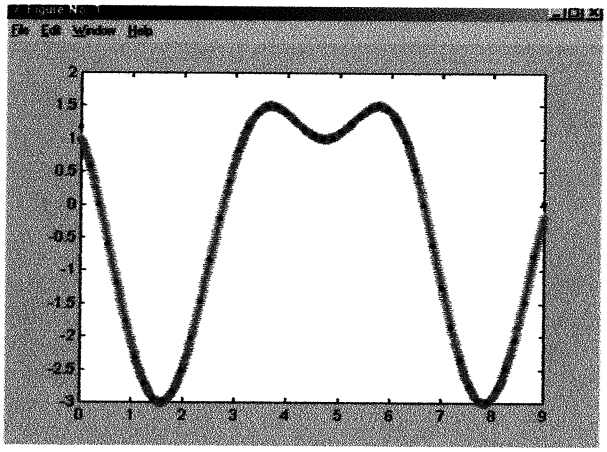
Con 19 puntos:

- Obtener el vector x y el vector y.

```
x=[0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5 5.5 6 6.5 7 7.5 8 8.5 9];
y=func(x);
```

- Visualizar la gráfica y obtener la bondad.

```
dibujo_bondad_spline_der(x,y)
```



0.1955

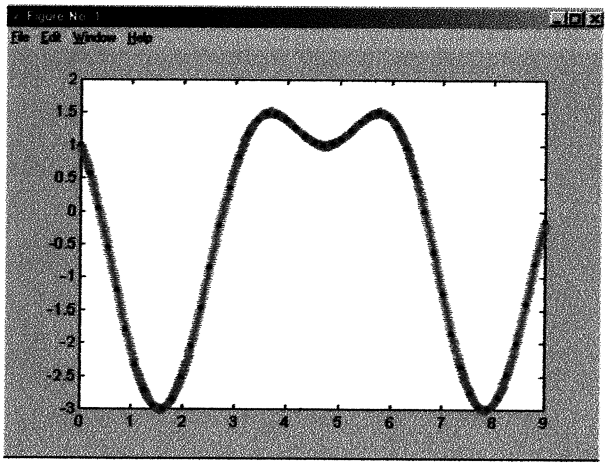
Con 37 puntos:

- Obtener el vector x y el vector y.

```
x=[0 0.25 0.5 0.75 1 1.25 1.5 1.75 2 2.25 2.5 2.75 3 3.25 3.5 3.75 4 4.25 4.5
   4.75 5 5.25 5.5 5.75 6 6.25 6.5 6.75 7 7.25 7.5 7.75 8 8.25 8.5 8.75 9]
y=func(x);
```

- Visualizar la gráfica y obtener la bondad.

```
dibujo_bondad_spline_der(x,y)
```



0.0261

c)

1.- Crear una nueva función que defina la derivada segunda de la función original.

```
function y=func_der2(x)
y=-2.*sin(x).*cos(x)-2.*cos(x).*sin(x)-2.*cos(x);
```

2.- Obtener el resultado.

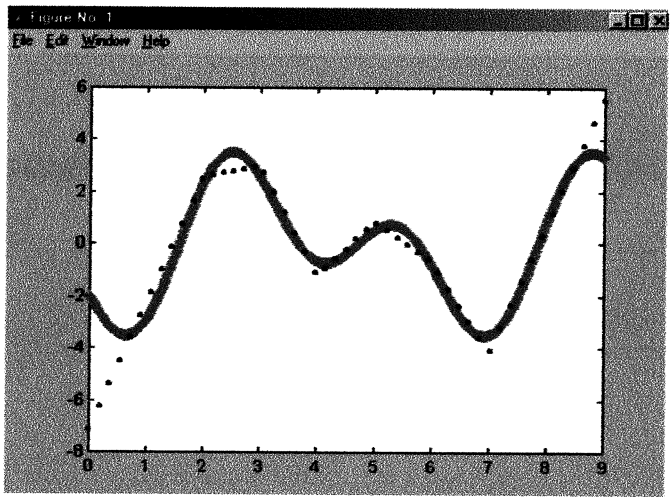
Con 10 puntos:

- Obtener el vector x y el vector y.

x=[0 1 2 3 4 5 6 7 8 9];
y=**func**(x);

- Visualizar la gráfica y obtener la bondad.

dibujo_bondad_spline_der2(x,y)



5.1393

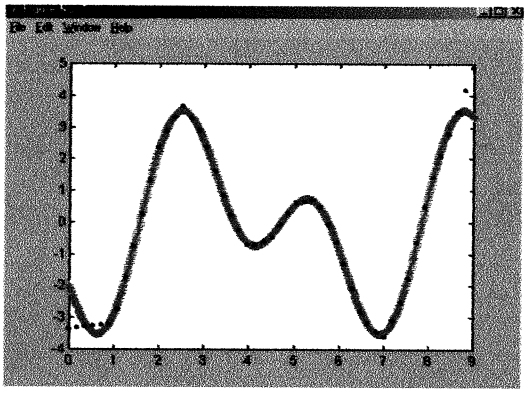
Con 19 puntos:

- Obtener el vector x y el vector y.

```
x=[0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5 5.5 6 6.5 7 7.5 8 8.5 9];
y=func(x);
```

- Visualizar la gráfica y obtener la bondad.

dibujo_bondad_spline_der2(x,y)



1.5482

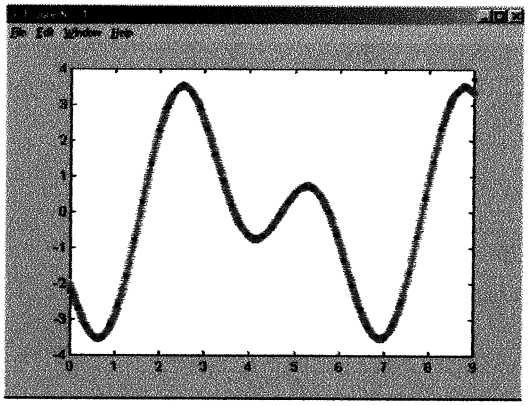
Con 37 puntos:

- Obtener el vector x y el vector y.

```
x=[0 0.25 0.5 0.75 1 1.25 1.5 1.75 2 2.25 2.5 2.75 3 3.25 3.5 3.75 4 4.25 4.5
   4.75 5 5.25 5.5 5.75 6 6.25 6.5 6.75 7 7.25 7.5 7.75 8 8.25 8.5 8.75 9]
y=func(x);
```

- Visualizar la gráfica y obtener la bondad.

dibujo_bondad_spline_der2(x,y)



0.4052

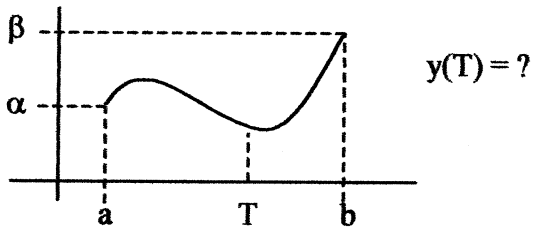
Como ha podido verse con todos los ejemplos, los Spline ajustan muy bien las funciones, pero ajustan un poco peor las derivadas y las segundas derivadas, sin embargo, al aumentar el número de puntos de la tabla de entrada se ha podido ver que se consigue un ajuste bastante bueno en todos los casos.

Método de Quasi Newton

El problema que se tiene es el mismo que en los dos métodos anteriores:

$$\begin{cases} y'' = f(t,y,y') \\ y(a) = \alpha \\ y(b) = \beta \end{cases}$$

Dado lo que vale la función y en dos puntos (extremos) y su ecuación diferencial de segundo orden, queremos saber lo que vale y en un punto determinado (T) que se encontrará entre a y b. Gráficamente es lo siguiente:



Para poder resolverlo, lo primero que hay que hacer es pasar el problema de contorno a problema de valores iniciales, para lo cual se pasa la ecuación de segundo orden a sistema de primer orden.

$$y'' = f(t,y,y') \qquad \begin{matrix} y_1 = y \\ y_2 = y' \end{matrix} \qquad \begin{cases} y_1' = y_2 \\ y_2' = f(t,y_1,y_2) \end{cases} \Rightarrow F$$

$$y(a) = \alpha \qquad y_1(a) = \alpha$$
$$y'(a) = x \qquad y_2(a) = x$$

Ahora se debe calcular un valor para x tal que $y_1(b) = \beta$, que es lo mismo que decir :

$$y_1(b;x) - \beta = 0 \quad \Rightarrow \quad \psi(x) = \text{Psi}(x)$$

Pero ahora x no se va a calcular por secante o bisección, como se ha hecho en los casos anteriores, sino que se va a aproximar con un p_0 (el cual se obtendrá de una gráfica hecha de Psi (x)) y el p siguiente (p_1) se obtendrá siguiendo la siguiente fórmula:

$$p_1 = p_0 - \frac{\psi(p_0)}{\text{derivada}\,(\psi(p_0))}$$

donde derivada es el siguiente método:

```
function sol=derivada(f,h,tol,max,x0)
r=zeros(max,max);
r(1,1)=(feval(f,(x0+h))-feval(f,(x0-h)))/(2*h);
sol=r(1,1);
for i=1:max
   h=h/2;
   r(i+1,1)=(feval(f,(x0+h))-feval(f,(x0-h)))/(2*h);
   sol=r(i+1,1);
   for k=1:i
      r(i+1,k+1)=r(i+1,k)+(((r(i+1,k))-(r(i,k)))/((4^k)-1));
      sol=r(i+1,k+1);
   end
   err=abs(r(i,i)-r(i+1,i+1));
   if (err<=tol)
      break;
   end
end
```

La función que calcula el valor de x es la siguiente:

```
function p0=quasi(psi,p0,tol,nmax)
for n=1:nmax
   p1=p0-(feval(psi,p0)/derivada(psi,0.01,10^-6,nmax,p0));
   if (abs(p1-p0)<tol)
      break;
   end
   p0=p1;
end
```

Por último, el código del método Quasi-Newton es el siguiente:

```
function z=quasi_newton(F,psi,a,alfa1,p0,t,tol,nmax)
x=quasi(psi,p0,tol,nmax);
y=runge_kutta(F,a,[alfa1 x],t,nmax);
z=y(1);
```

. . . . . .

El problema es que usando la fórmula

$$p_1 = p_0 - \frac{\psi(p_0)}{\text{derivada}\,(\psi(p_0))}$$

se necesita hacer muchas llamadas a Runge_kutta, porque derivada hace extrapolación, y se ralentiza mucho el problema, además de que se pierde exactitud con tantos cálculos.

Una forma de solucionarlo es crear una tabla alrededor de p_0 e interpolarla, de forma que se trabaje con la interpolación en vez de con la función. Ahora, cada vez que se necesite un valor de la función se recurrirá al polinomio interpolado, de tal forma que sólo habrá que ejecutar Runge_kutta una vez, para realizar la interpolación.

Esto se realiza creando una nueva función, que sustituirá a Quasi y su código es el siguiente:

```
function p0=quasi2(p0,tol,nmax,x,y)
for n=1:nmax
   p1=p0-
(feval('psi_spline',x,y,p0)/derivada_nueva('psi_spline',
      x,y,0.01,10^-6,nmax,p0));
   if (abs(p1-p0)<tol)
      break;
   end
   p0=p1;
end
```

Como puede verse, el método utiliza una función, que es la que crea el polinomio interpolado por Spline. La función tiene el siguiente código:

```
function z=psi_spline(x,y,p0)
z=spline(x,y,p0);
```

Y se utiliza también un nuevo método, que en realidad es el método de la derivada pero modificado para que trabaje con psi_spline.

```
function sol=derivada_nueva(f,x,y,h,tol,max,x0)
r=zeros(max,max);
r(1,1)=(feval(f,x,y,(x0+h))-feval(f,x,y,(x0-h)))/(2*h);
sol=r(1,1);
for i=1:max
   h=h/2;
   r(i+1,1)=(feval(f,x,y,(x0+h))-feval(f,x,y,(x0-h)))/(2*h);
   sol=r(i+1,1);
   for k=1:i
      r(i+1,k+1)=r(i+1,k)+(((r(i+1,k))-(r(i,k)))/((4^k)-1));
      sol=r(i+1,k+1);
   end
   err=abs(r(i,i)-r(i+1,i+1));
   if (err<=tol)
      break;
   end
end
```

Ahora, Quasi-Newton utiliza Quasi2 en vez de Quasi y hay que pasarle la tabla para que realice la interpolación. Queda de la siguiente forma:

```
function z=quasi_newton2(F,a,alfa1,p0,t,tol,nmax,x,y)
x=quasi2(p0,tol,nmax,x,y);
y=runge_kutta(F,a,[alfa1 x],t,nmax);
z=y(1);
```

EJERCICIOS:

1.- Resolver el siguiente sistema:

$$\begin{cases} y'' + 6t^2 + t^3 - 1 = 2y' + y + 6t \\ y(-1) = -2 \\ y(1) = 0 \end{cases}$$

Calcular y(2), y(3), y(5) e y(7).

NOTA: Para probar los resultados tener en cuenta que $y(t) = t^3 - 1$

Solución:

1.- Definir el problema como un problema de valores iniciales:

A y se le llama y_1 → $y_1 = y$
A y' se le llama y_2 → $y_2 = y'$
$y'' + 6t^2 + t^3 - 1 = 2y' + y + 6t$ → $y'' = 2y' + y + 6t - 6t^2 - t^3 + 1$

$$\begin{cases} y_1' = y_2 \\ y_2' = 2y_2 + y_1 + 6t - 6t^2 - t^3 + 1 \\ y_1(-1) = -2 \\ y_2(-1) = x \end{cases}$$

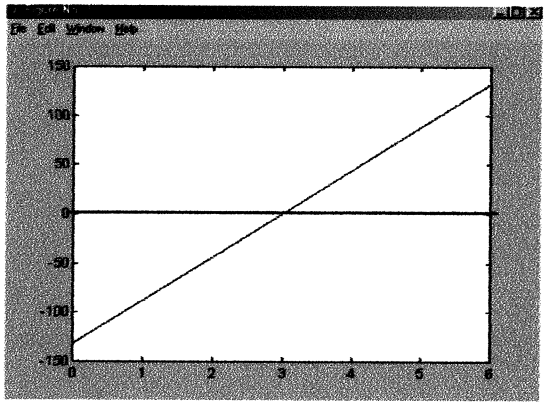
```
function z=F(t,y)
z(1)=y(2);
z(2)=2*y(2)+y(1)+6*t-6*t.^2-t.^3+1;
```

2.- Definir la función psi

```
function z=psi(x)
y=runge_kutta('F',-1,[-2 x],1,100);
z=y(1)-0;
```

3.- Realizar un análisis gráfico de la función Psi. Para ello utilizo la función dibujar.

```
dibujar(0,6,'psi')
```



Forma 1:

4.- Ejecutar el método para obtener los resultados. (El método utiliza Quasi.)

```
function z=quasi_newton(F,psi,a,alfa1,p0,t,tol,nmax)
x=quasi(psi,p0,tol,nmax);
y=runge_kutta(F,a,[alfa1 x],t,nmax);
z=y(1);
```

Como puede verse en la gráfica, la raíz está aproximadamente en x = 3, por lo que voy a dar 3 como p0.

quasi_newton('F','psi',-1,-2,3,2,10^-4,1000) → **6.9999**
quasi_newton('F','psi',-1,-2,3,3,10^-4,1000) → **25.9999**
quasi_newton('F','psi',-1,-2,3,5,10^-4,1000) → **123.9998**
quasi_newton('F','psi',-1,-2,3,7,10^-4,1000) → **341.9108**

Forma 2:

4.- Obtener una tabla alrededor de la raíz de psi (3).

```
x=[1.5 2 2.5 3 3.5 4 4.5];
y(1)=psi(x(1));
y(2)=psi(x(2));
y(3)=psi(x(3));
y(4)=psi(x(4));
y(5)=psi(x(5));
y(6)=psi(x(6));
y(7)=psi(x(7));
```

5.- Ejecutar el método para obtener los resultados. (El método utiliza Quasi2.)

```
function z=quasi_newton2(F,a,alfa1,p0,t,tol,nmax,x,y)
x=quasi2(p0,tol,nmax,x,y);
y=runge_kutta(F,a,[alfa1 x],t,nmax);
z=y(1);
```

p0 = 3

quasi_newton2('F',-1,-2,3,2,10^-4,1000,x,y) → **7.0000**
quasi_newton2('F',-1,-2,3,3,10^-4,1000,x,y) → **26.0000**
quasi_newton2('F',-1,-2,3,5,10^-4,1000,x,y) → **123.9998**
quasi_newton2('F',-1,-2,3,7,10^-4,1000,x,y) → **341.9108**

2.- Teniendo el siguiente sistema no lineal:

$$\begin{cases} y'' = 1/8 * (32 + 2t^3 - y * y') \\ y(1) = 17 \\ y(3) = 43/3 \end{cases}$$

que tiene como solución exacta $y(x) = x^2 + 16/x$

Calcular y(1), y(2), y(3).

Solución:

1.- Definir el problema como un problema de valores iniciales:

A y se le llama y_1 → $y_1 = y$
A y' se le llama y_2 → $y_2 = y'$
$y'' = 1/8 * (32 + 2t^3 - y * y')$

$$\begin{cases} y_1' = y_2 \\ y_2' = 1/8 * (32 + 2t^3 - y_1 * y_2 \\ y_1(1) = 17 \\ y_2(1) = x \end{cases}$$

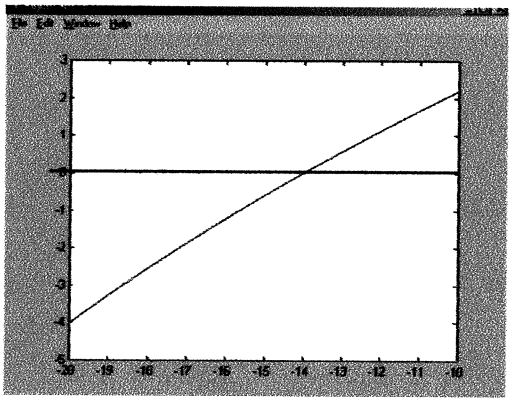
```
function z=F(t,y)
z(1)=y(2);
z(2)=1/8*(32+2*t.^3-y(1)*y(2));
```

2.- Definir la función psi

```
function z=psi(x)
y=runge_kutta('F',1,[17 x],3,100);
z=y(1)-43/3;
```

3.- Realizar un análisis gráfico de la función psi. Para ello utiliza la función dibujar que creé en el ejercicio anterior.

```
dibujar(-20,-10,'psi')
```



Según la gráfica, sólo hay una raíz, la cual se encuentra aproximadamente en –14, luego voy a darle al método un p0 de –14.

Forma 1:

4.- Ejecutar el método para obtener los resultados. (El método utiliza Quasi.)

```
function z=quasi_newton(F,psi,a,alfa1,p0,t,tol,nmax)
x=quasi(psi,p0,tol,nmax);
y=runge_kutta(F,a,[alfa1 x],t,nmax);
z=y(1);
```

quasi_newton('F','psi',1,17,-14,1,10^-4,1000) → **17.0000**
quasi_newton('F','psi',1,17,-14,2,10^-4,1000) → **12.0000**
quasi_newton('F','psi',1,17,-14,3,10^-4,1000) → **14.3333**

Forma 2:

4.- Obtener una tabla alrededor de –14, que es la raíz de psi.

```
x=[-15.6 -15.2 -14.8 -14.4 -14 -13.6 -13.2 -12.8 -12.4];
y(1)=psi(x(1));
y(2)=psi(x(2));
y(3)=psi(x(3));
y(4)=psi(x(4));
y(5)=psi(x(5));
y(6)=psi(x(6));
y(7)=psi(x(7));
y(8)=psi(x(8));
y(9)=psi(x(9));
```

5.- Ejecutar el método para obtener los resultados. (El método utiliza Quasi2.)

```
function z=quasi_newton2(F,a,alfa1,p0,t,tol,nmax,x,y)
x=quasi2(p0,tol,nmax,x,y);
y=runge_kutta(F,a,[alfa1 x],t,nmax);
z=y(1);
```

quasi_newton2('F',1,17,-14,1,10^-4,1000,x,y) → **17.0000**
quasi_newton2('F',1,17,-14,2,10^-4,1000,x,y) → **12.0000**
quasi_newton2('F',1,17,-14,3,10^-4,1000,x,y) → **14.3333**

3.- Teniendo el siguiente sistema no lineal:

$$\begin{cases} y'' = 2y^3 - 6y - 2t^3 \\ y(1) = 2 \\ y(2) = 5/2 \end{cases}$$

que tiene como solución exacta $y(t) = t + t^{-1}$

Calcular y(1.3), y(1.6), y(1.9).

Solución:

1.- Definir el problema como un problema de valores iniciales:

A y se le llama y_1 → $y_1 = y$
A y' se le llama y_2 → $y_2 = y'$
$y'' = 2y^3 - 6y - 2t^3$

$$\begin{cases} y_1' = y_2 \\ y_2' = 2y_1^3 - 6y_1 - 2t^3 \\ y_1(1) = 2 \\ y_2(1) = x \end{cases}$$

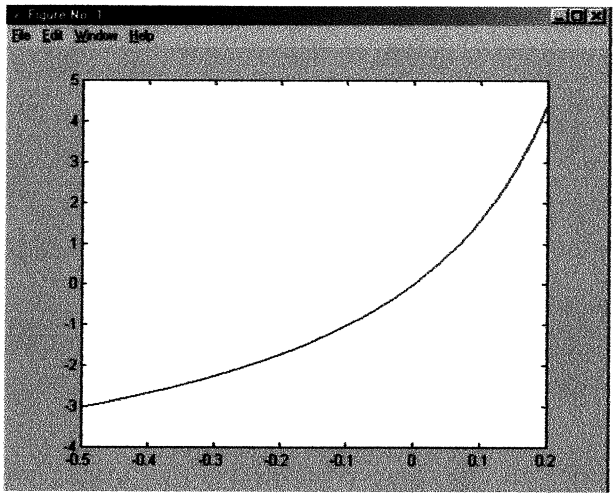
```
function z=F(t,y)
z(1)=y(2);
z(2)=2*y(1).^3-6*y(1)-2*t.^3;
```

2.- Definir la función psi

```
function z=psi(x)
y=runge_kutta('F',1,[2 x],2,100);
z=y(1)-5/2;
```

3.- Realizar un análisis gráfico de la función psi. Para ello utiliza la función dibujar que creé en el ejercicio anterior.

```
dibujar(-0.5,0.2,'psi')
```



Según la gráfica, sólo hay una raíz, la cual se encuentra aproximadamente en 0

Forma 1:

4.- Ejecutar el método para obtener los resultados. (El método utiliza Quasi.)

```
function z=quasi_newton(F,psi,a,alfa1,p0,t,tol,nmax)
x=quasi(psi,p0,tol,nmax);
y=runge_kutta(F,a,[alfa1 x],t,nmax);
z=y(1);
```

quasi_newton('F','psi',1,2,0,1.3,10^-4,1000) → **2.0692**
quasi_newton('F','psi',1,2,0,1.6,10^-4,1000) → **2.2250**
quasi_newton('F','psi',1,2,0,1.9,10^-4,1000) → **2.4263**

Forma 2:

4.- Obtener una tabla alrededor de –14, que es la raíz de psi.

```
x=[-0.2 -0.15 -0.1 -0.05 0 0.05 0.1 0.15 0.2];
y(1)=psi(x(1));
y(2)=psi(x(2));
y(3)=psi(x(3));
y(4)=psi(x(4));
y(5)=psi(x(5));
y(6)=psi(x(6));
y(7)=psi(x(7));
y(8)=psi(x(8));
y(9)=psi(x(9));
```

5.- Ejecutar el método para obtener los resultados. (El método utiliza Quasi2.)

```
function z=quasi_newton2(F,a,alfa1,p0,t,tol,nmax,x,y)
x=quasi2(p0,tol,nmax,x,y);
y=runge_kutta(F,a,[alfa1 x],t,nmax);
z=y(1);
```

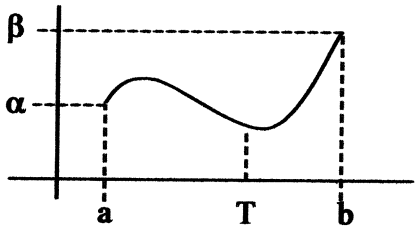
quasi_newton2('F',1,2,0,1.3,10^-4,1000,x,y) → **2.0692**
quasi_newton2('F',1,2,0,1.6,10^-4,1000,x,y) → **2.2250**
quasi_newton2('F',1,2,0,1.9,10^-4,1000,x,y) → **2.4263**

Método de Newton

De nuevo se tiene un problema del tipo:

$$\begin{cases} y'' = f(t,y,y') \\ y(a) = \alpha \\ y(b) = \beta \end{cases}$$

En el cual se sabe lo que vale una función en dos extremos y se quiere saber lo que valdrá en un punto intermedio. Gráficamente es lo siguiente:



Para resolverlo, como siempre, hay que pasar el problema de contorno a problema de valores iniciales, para lo cual se pasa la ecuación de segundo orden a sistema de primer orden.

$$y'' = f(t,y,y') \quad \begin{matrix} y_1 = y \\ \longrightarrow \\ y_2 = y' \end{matrix} \quad \begin{cases} y_1' = y_2 \\ y_2' = f(t,y_1,y_2) \end{cases} \Rightarrow F$$

$$y(a) = \alpha \qquad y_1(a) = \alpha$$
$$y'(a) = x \qquad y_2(a) = x$$

Ahora se plantea la ecuación:

$$y_1(b;x) - \beta = 0$$

$$\psi(x) = \text{Psi}(x)$$

Aplicando ahora el método Newton-Raphson, se toma un x_0 inicial (basándonos en el gráfico que previamente se debe haber hecho de Psi) con el cual se calcula x resolviendo la siguiente ecuación:

$$x_1 = x_0 - \frac{\psi(x_0)}{\boxed{\psi'(x_0)}} \quad !$$

El problema está en la forma de calcular la derivada de Psi en x_0.

- **Cálculo de $\psi'(x_0)$**

$\psi'(x_0) = \partial y/\partial x\ (b;x)$ ⟶ Vamos a llamar a esto z(b), de tal forma que si en vez de evaluarlo en un instante concreto (b) lo evaluamos en cualquier t obtenemos:

$$z(t) = \partial y/\partial x\ (t;x)$$

La función inicial es:

$$y''(t;x) = f(t;y(t;x);y'(t;x))$$

Derivando esta ecuación respecto de x se obtiene:

$$(\partial y/\partial x\ (t;x))'' = f_y\ (t,y,y') * \partial y/\partial x\ (t;x) + f_{y'}\ (t,y,y')* (\partial y/\partial x\ (t;x))'$$

donde f_y y $f_{y'}$ son la " derivada de f respecto de y " y la " derivada de f respecto de y' " respectivamente.

Como puede verse, $(\partial y/\partial x\ (t;x))''$ es z''(t). Si ahora llamamos a(t) a $f_y(t,y,y')$ y b(t) a $f_{y'}(t,y,y')$ se obtiene la siguiente ecuación:

$$z''(t) = a(t) * z(t) + b(t) * z'(t)$$

Si ahora se deriva y(a;x) e y'(a;x) se obtiene lo siguiente:

$$\begin{matrix} y(a;x) = \alpha \\ y'(a;x) = x \end{matrix} \longrightarrow \begin{matrix} \partial y/\partial x\ (a;x) = 0 \\ (\partial y/\partial x\ (a;x))' = 1 \end{matrix} \rightarrow \begin{matrix} z(a) \\ z'(a) \end{matrix}$$

Con todo lo que se acaba de hacer se obtiene un problema de valores iniciales,

$$\begin{cases} z''(t) = a(t) * z(t) + b(t) * z'(t) \\ z(a) = 0 \\ z'(a) = 1 \end{cases}$$

el cual se puede resolver fácilmente con Runge_kutta, que dará como resultado z(t) y z'(t). Si tomamos t = b obtendremos z(b) que es $\psi'(x_0)$.

EJERCICIOS:

1.- Resolver el siguiente sistema:

$$\begin{cases} y'' + 6t^2 + t^3 - 1 = 2y' + y + 6t \\ y(-1) = -2 \\ y(1) = 0 \end{cases}$$

Calcular y(2), y(3), y(5) e y(7).

Solución:

1.- Definir el problema como un problema de valores iniciales:

A y se le llama y_1 → $y_1 = y$
A y' se le llama y_2 → $y_2 = y'$
$y'' + 6t^2 + t^3 - 1 = 2y' + y + 6t$ → $y'' = 2y' + y + 6t - 6t^2 - t^3 + 1$

$$\begin{cases} y_1' = y_2 \\ y_2' = 2y_2 + y_1 + 6t - 6t^2 - t^3 + 1 \\ y_1(-1) = -2 \\ y_2(-1) = x \end{cases}$$

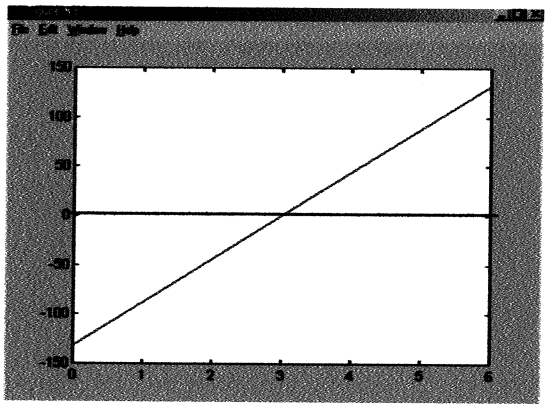
```
function z=F(t,y)
z(1)=y(2);
z(2)=2*y(2)+y(1)+6*t-6*t.^2-t.^3+1;
```

2.- Definir la función psi

```
function z=psi(x)
y=runge_kutta('F',-1,[-2 x],1,100);
z=y(1)-0;
```

3.- Realizar un análisis gráfico de la función Psi. Para ello utilizo la función dibujar.

```
dibujar(0,6,'psi')
```



4.- Crear una tabla cuyos valores estén próximos a la raíz de Psi(x), en este caso a 3.

```
x=[1.5 2 2.5 3 3.5 4 4.5];
y(1)=psi(x(1));
y(2)=psi(x(2));
y(3)=psi(x(3));
y(4)=psi(x(4));
y(5)=psi(x(5));
y(6)=psi(x(6));
y(7)=psi(x(7));
```

5.- Crear una función que realice una interpolación por Spline de la tabla obtenida anteriormente.

```
function z=psi_spline(x,y,p0)
z=spline(x,y,p0);
```

6.- Definir el segundo problema de valores iniciales que se debe resolver.

$$\begin{cases} z_2(t) = a(t) * z_1(t) + b(t) * z_2(t) \\ z_1(-1) = 0 \\ z_2(-1) = 1 \end{cases}$$

```
function w=G(t,z)
w=func_a(t).*z(1)+func_b(t).*z(2);
```

7.- Definir las funciones func_a(t) y func_b(t).

func_a(t) es la derivada de la función respecto de y ⟶ func_a(t) = 1.
func_b(t) es la derivada de la función respecto de y' ⟶ func_b(t) = 2.

```
function w=func_a(t)
w=1;

function w=func_b(t)
w=2;
```

8.- Crear el método y ejecutarlo para obtener el resultado.

El código final del método es el siguiente:

```
function y=newton_contorno(G,F,a,b,alfa1,p0,t,x,y,tol,nmax)
for n=1:nmax
   r=runge_kutta(G,a,[0 1],b,nmax);
   p1=p0-(feval('psi_spline',x,y,p0)/r(1));
   if (abs(p1-p0)<tol)
      break;
   end
   p0=p1;
end
y=runge_kutta(F,a,[alfa1 p0],t,nmax);
y=y(1);
```

Calcula x

Resuelve el primer problema de valores iniciales, es decir, obtiene el resultado.

Llamadas:

newton_contorno('G','F',-1,1,-2,3,2,x,y,10^-4,1000) → **7.0000**
newton_contorno('G','F',-1,1,-2,3,3,x,y,10^-4,1000) → **26.0000**
newton_contorno('G','F',-1,1,-2,3,5,x,y,10^-4,1000) → **123.9998**
newton_contorno('G','F',-1,1,-2,3,7,x,y,10^-4,1000) → **341.9108**

2.- Teniendo el siguiente sistema no lineal:

$$\begin{cases} y'' = 1/8 * (32 + 2t^3 - y * y') \\ y(1) = 17 \\ y(3) = 43/3 \end{cases}$$

que tiene como solución exacta $y(x) = x^2 + 16/x$

Calcular y(1), y(2), y(3).

Solución:

NOTA: Al ser el problema no lineal se van a necesitar ciertos cambios a las funciones del ejercicio anterior para que sea eficiente, así que voy a hacer el ejercicio completo con todas las funciones y todos los cambios.

1.- Definir el problema como un problema de valores iniciales:

A y se le llama y_1 → $y_1 = y$
A y' se le llama y_2 → $y_2 = y'$
$y'' = 1/8 * (32 + 2t^3 - y * y')$

$$\begin{cases} y_1' = y_2 \\ y_2' = 1/8 * (32 + 2t^3 - y_1 * y_2 \\ y_1(1) = 17 \\ y_2(1) = x \end{cases}$$

```
function z=F(t,y)
z(1)=y(2);
z(2)=1/8*(32+2*t.^3-y(1)*y(2));
```

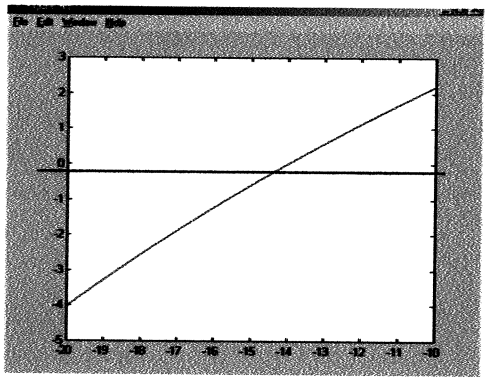
2.- Definir la función psi

Porque a Runge-Kutta hay que añadirle una entrada (una tabla). Se le pasa cualquier valor porque F no lo va a utilizar.

```
function z=psi(x)
y=runge_kutta('F',1,[17 x],3,100,0);
z=y(1)-43/3;
```

3.- Realizar un análisis gráfico de la función psi para ver dónde se encuentra la raíz. Para ello hay que utilizar la función dibujar que se ha creado en ejercicios anteriores.

```
dibujar(-20,-10,'psi')
```



4.- Crear una función que haga que psi pero con la segunda componente de Runge-kutta (y').

```
function z=psi_der(x)
y=runge_kutta('F',1,[17 x],3,100,18);
z=y(2)-43/3;
```

Porque a Runge-Kutta hay que añadirle una entrada (una tabla). Se le pasa cualquier valor porque F no lo va a utilizar.

5.- Obtener una tabla alrededor de –14, que es la raíz. Para ello es necesario crear una función que la calcule, ya que se va a hacer uso de esta tabla en varias ocasiones.

```
function tab=crear_tabla_RK(x0)
%Crea una tabla de 9 eltos.
tab=zeros(9,3);
i=4;
x=x0-0.5;
while i>0
   tab(i,1)=x;
   tab(i,2)=psi(x);
   tab(i,3)=psi_der(x);
   i=i-1;
   x=x-0.5;
end
tab(5,1)=x0;
tab(5,2)=psi(x0);
tab(5,3)=psi_der(x0);
i=6;
x=x0+0.5;
while i<10
   tab(i,1)=x;
   tab(i,2)=psi(x);
   tab(i,3)=psi_der(x);
   i=i+1;
   x=x+0.5;
end
```

6.- Crear una función que realice una interpolación por Spline de la tabla obtenida anteriormente.

```
function z=psi_spline(x,y,p0)
z=spline(x,y,p0);
```

7.- Construir el segundo problema de valores iniciales y crear una función que lo defina.

$$\begin{cases} z_2(t) = a(t) * z_1(t) + b(t) * z_2(t) \\ z_1(1) = 0 \\ z_2(1) = 1 \end{cases}$$

```
function w=G(t,z,tab)
y1=feval('psi_spline',tab(:,1),tab(:,2),t);
y2=feval('psi_spline',tab(:,1),tab(:,3),t);
w=-1/8*y1.*z(1)-1/8*y2.*z(2);
```

8.- Modificar Runge-Kutta para que pase a G la tabla.

```
function w0=runge_kutta (f,a,alfa,T,n,tabla)
h=(T-a)/n;
t=a;
w0=alfa;
for i=1:n
   f1=feval(f,t,w0,tabla);
   f2=feval(f,t+h/2,w0+f1*h/2,tabla);
   f3=feval(f,t+h/2,w0+f2*h/2,tabla);
   f4=feval(f,t+h,w0+h*f3,tabla);
   w1=w0+(h*(f1+2*f2+2*f3+f4)/6);
   t=t+h;
   w0=w1;
end
```

9.- Crear el método y ejecutarlo para obtener el resultado.

```
function y=newton_contorno(G,F,a,b,alfa1,p0,t,tol,nmax)
for n=1:10
   tabla=crear_tabla_RK(p0);
   x=tabla(:,1);
   y=tabla(:,2);
   r=runge_kutta(G,a,[0 1],b,nmax,tabla);
   p1=p0-(feval('psi_spline',x,y,p0)/r(1));
   if (abs(p1-p0)<tol)
      break;
   end
   p0=p1;
end
y=runge_kutta(F,a,[alfa1 p0],t,nmax,18);
y=y(1);
```

Porque a Runge-Kutta hay que añadirle una entrada (una tabla). Se le pasa cualquier valor porque F no lo va a utilizar.

newton_contorno('G','F',1,3,17,-14,1,10^-4,100) → **17.0000**
newton_contorno('G','F',1,3,17,-14,2,10^-4,100) → **12.0000**
newton_contorno('G','F',1,3,17,-14,3,10^-4,100) → **14.3333**

3.- Aplique el método del disparo no lineal usando bisección para aproximar la solución al siguiente problema de contorno:

$$\begin{cases} y'' = y^3 - y * y' \\ y(1) = 1/2 \\ y(2) = 1/3 \end{cases}$$

definido en $1 \leq t \leq 2$. Obtener lo que vale el sistema para todos los t que lo definen y comparar el resultado con la solución real, que la proporciona $y(t) = 1 / (t + 1)$.

Calcular y(1.1), y(1.4), y(1.7).

Solución:

1.- Definir el problema como un problema de valores iniciales:

A y se le llama y_1 → $y_1 = y$
A y' se le llama y_2 → $y_2 = y'$
$y'' = y^3 - y * y'$

$$\begin{cases} y_1' = y_2 \\ y_2' = y_1^3 - y_1 * y_2 \\ y_1(1) = 1/2 \\ y_2(1) = x \end{cases}$$

```
function z=F(t,y)
z(1)=y(2);
z(2)=y(1).^3-y(1)*y(2);
```

2.- Definir la función psi

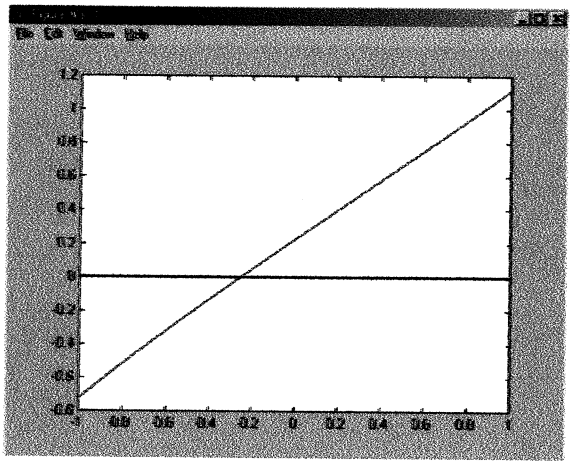
```
function z=psi(x)
y=runge_kutta('F',1,[1/2 x],2,100,18);
z=y(1)-1/3;
```

3.- Definir la función psi_der

```
function z=psi_der(x)
y=runge_kutta('F',1,[1/2 x],2,100,18);
z=y(2)-1/3;
```

4.- Realizar un análisis gráfico de la función psi para ver dónde se encuentra la raíz. Para ello hay que utilizar la función dibujar que se ha creado en ejercicios anteriores.

```
dibujar(-1,1,'psi')
```



5.- Construir el segundo problema de valores iniciales y crear una función que lo defina.

$$\begin{cases} z_2(t) = a(t) * z_1(t) + b(t) * z_2(t) \\ z_1(1) = 0 \\ z_2(1) = 1 \end{cases}$$

```
function w=G(t,z,tab)
y1=feval('psi_spline',tab(:,1),tab(:,2),t);
y2=feval('psi_spline',tab(:,1),tab(:,3),t);
w=(3*y1.^2-y2).*z(1)-y1.*z(2);
```

6.- No es necesario crear psi_spline, crear_tabla, Runge_Kutta ni Newton_contorno porque tienen el mismo código.

7.- Ejecutar el método para obtener el resultado.

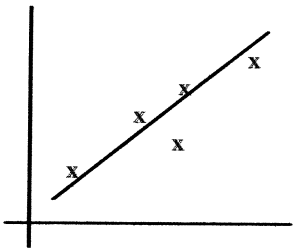
```
newton_contorno('G','F',1,2,1/2,-0.2,1.1,10^-4,100)
newton_contorno('G','F',1,2,1/2,-0.2,1.4,10^-4,100)
newton_contorno('G','F',1,2,1/2,-0.2,1.7,10^-4,100)
```

→ **0.4762**
→ **0.4167**
→ **0.3704**

AJUSTE POR MÍNIMOS CUADRADOS

Dada una nube de puntos (tabla), obtener la función que mejor se ajuste a ellos.

x	y
x0	y0
x1	y1
...	...
xm	ym



$y = ax + b$

El criterio que se sigue para obtener la función que mejor se ajusta es que el error (la suma de los cuadrados de la distancia que hay entre la función y los puntos) sea mínimo, es decir:

Error = $E(f(x)) = (f(x0) - y0)^2 + (f(x1) - y1)^2 + ... + (f(xm) - ym)^2$ donde $E(f(x))$ debe ser mínimo, siendo $f(x)$ la función obtenida.

Si la función a obtener es la siguiente:

$$y = F_n(x) = a_0 + a_1x + a_2x^2 + ... + a_nx^n$$

(siendo n el grado del polinomio), el error será:

$$E = \sum_{i=0}^{m} [F_n(x_i) - y_i]^2$$

Para conseguir que el error sea mínimo hay que hacer que la derivada sea 0.

$$\partial E / \partial a_j = \sum_{i=0}^{m} 2 * [F_n(x_i) - y_j] * x_i^j = 0 \qquad j = 0, ... , n$$

Esto es:

$$(F_n(x_0) - y_0) * x_0^j + (F_n(x_1) - y_1) * x_1^j + ... + (F_n(x_m) - y_m) * x_m^j = 0$$

$$\begin{cases} j=0 & a_0(1+1+...+1)+a_1(x_0+x_1+...+x_m)+a_2(x_0^2+x_1^2+...+x_m^2)+ ... +a_n(x_0^n+x_1^n+...+x_m^n) = y_0+y_1+...+y_m \\ j=1 & a_0(x_0+x_1+...+x_m)+a_1(x_0^2+x_1^2+...+x_m^2)+a_2(x_0^3+x_1^3+...+x_m^3)+ ... +a_n(x_0^n+x_1^n+...+x_m^n) = y_0+y_1+...+y_m \\ j=i & a_0[x_0^{i+j}+x_1^{i+j}+...+x_m^{i+j}] \end{cases}$$

En forma matricial sería:

$$\begin{array}{l} \\ j=0 \\ j=1 \\ \\ j=i \end{array} \quad \begin{array}{c} A \\ \begin{pmatrix} \\ \\ A(i,j) \\ \\ \end{pmatrix} \end{array} * \begin{array}{c} x \\ \begin{pmatrix} a_0 \\ a_1 \\ \\ \\ a_n \end{pmatrix} \end{array} = \begin{array}{c} b \\ \begin{pmatrix} y_0+y_1+\ldots+y_m \\ y_0 * x_0 + y_1 * x_1 +\ldots+y_m * x_m \\ \\ y_0 * x_0^{j} + y_1 * x_1^{j} +\ldots+y_m * x_m^{j} \\ \end{pmatrix} \end{array}$$

$$x = A \setminus b$$

Siendo A:

$$\begin{pmatrix} 1+1+\ldots+1 & x_0+x_1+\ldots+x_m & x_0^2+x_1^2+\ldots+x_m^2 & \ldots \\ x_0+x_1+\ldots+x_m & x_0^2+x_1^2+\ldots+x_m^2 & \ldots & \\ x_0^2+x_1^2+\ldots+x_m^2 & \ldots & & \\ \ldots & & & \end{pmatrix}$$

Los elementos interiores de la matriz A se repiten. Para que el algoritmo sea más rápido es suficiente con calcular los elementos de la primera fila y de la última columna, ya que con ellos se puede rellenar toda la matriz.

El código del método es el siguiente:

```
function x=min_cuadrados(x,y,grado)
grado=grado+1;
nx=length(x);

matrizA=zeros(grado,grado);
fila1=zeros(1,grado);
fila1(1)=nx;
for i=2:grado
   for j=1:nx
      fila1(i)=fila1(i)+x(j).^(i-1);
   end
end
n=grado-1;
column=zeros(grado,1);
for i=1:grado
   for j=1:nx
      column(i)=column(i)+x(j).^n;
   end
   n=n+1;
end
matrizA(1,:)=fila1;
matrizA(:,grado)=column;
fila=fila1;
for i=2:grado
   for j=1:grado-1
      matrizA(i,j)=fila(j+1);
   end
   fila=matrizA(i,:);
end

matrizB=zeros(grado,1);
for j=1:nx
   matrizB(1)=matrizB(1)+y(j);
end
for i=2:grado
   for j=1:nx
      matrizB(i)=matrizB(i)+y(j)*x(j).^(i-1);
   end
end

x=matrizA\matrizB;
```

EJERCICIOS:

1.- Con los puntos (0,0), (1,2), (2,-1) calcular la recta de regresión que mejor se ajusta. Comprobar el resultado sabiendo que debe coincidir con los coeficientes que proporciona polyfit(x,y,1)

Solución:

1.- Obtener el resultado

```
min_cuadrados([0 1 2],[0 2 -1],1)

    0.8333
   -0.5000
```

El resultado indica que la recta es:

$$y = -0.5x + 0.83$$

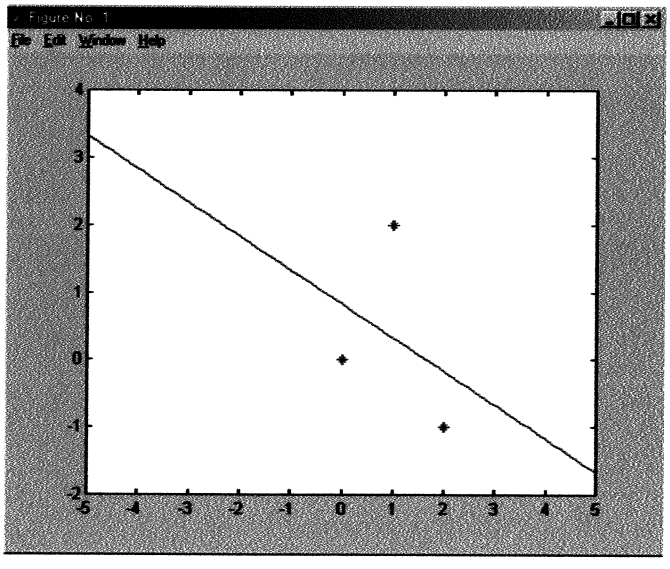
2.- Comprobar con polyfit:

polyfit([0 1 2],[0 2 -1],2) ⟶ -0.5000 0.8333

3.- Comprobar con grafica:

```
function y=P(x)
y = -0.5*x + 0.83
```

```
x1=-5:0.1:5; y1=P(x1); x2=[0 1 2]; y2=[0 2 -1]; plot(x1,y1,x2,y2,'*r')
```



2.- Con los puntos (-2,1), (-1,2), (0,0), (1,2), (2,-1) calcular la parábola que mejor se ajusta. Comprobar el resultado sabiendo que debe coincidir con los coeficientes que proporciona polyfit(x,y,2)

Solución:

1.- Obtener el resultado

```
min_cuadrados([-2 -1 0 1 2],[1 2 0 2 -1],2)
```

1.3714
-0.4000
-0.2857

Según esto la parábola es:

$$y = -0.2857x^2 - 0.4x + 1.3714$$

2.- Comprobar con polyfit:

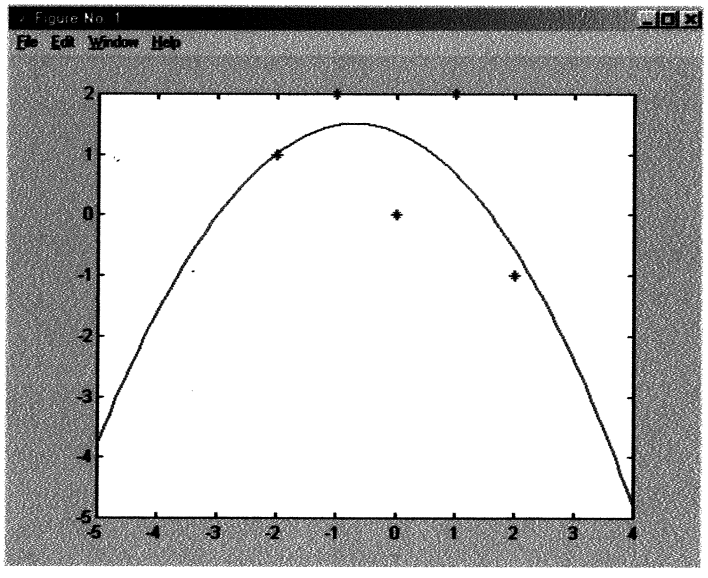
```
polyfit([-2 -1 0 1 2],[1 2 0 2 -1],2)
```

-0.2857 -0.4000 1.3714

3.- Comprobar con grafica:

```
function y=P(x)
y=-0.2857*x.^2-0.4*x+1.3714;
```

```
x1=-5:0.1:4; y1=P(x1); x2=[-2 -1 0 1 2]; y2=[1 2 0 2 -1];
plot(x1,y1,x2,y2,'*r')
```



3.- Con la siguiente tabla de puntos:

x	y
1	1.3
2	3.5
3	4.2
4	5
5	7
6	8.8
7	10.1
8	12.5
9	13.0
10	15.6

calcular la recta que mejor se ajusta. Comprobar el resultado sabiendo que debe coincidir con los coeficientes que proporciona polyfit(x,y,1)

Solución:

1.- Obtener el resultado

min_cuadrados([1 2 3 4 5 6 7 8 9 10],[1.3 3.5 4.2 5 7 8.8 10.1 12.5 13.0 15.6],1)

-0.3600
1.5382

Según esto la recta es:

$y = 1.5382x - 0.36$

2.- Comprobar con polyfit:

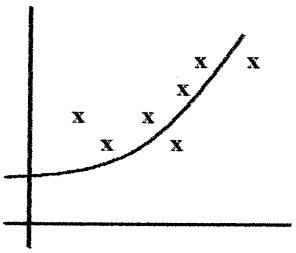
polyfit ([1 2 3 4 5 6 7 8 9 10],[1.3 3.5 4.2 5 7 8.8 10.1 12.5 13.0 15.6],1)

1.5382 -0.3600

Ajuste de mínimos cuadrados de forma no lineal usando Newton (Ajustando con función exponencial)

Se trata de ajustar una serie de puntos sabiendo el tipo de función que se debe obtener como resultado. Para estudiar este método vamos a ajustar los puntos con una curva exponencial (se puede usar cualquier tipo de función).

x	y
x0	y0
x1	y1
...	...
xm	ym



$y = ae^{bx}$

Se debe calcular el valor de a y b (inicialmente se calcularán a_0 y b_0 de forma aproximada usando una maya) para que la curva que se obtenga sea la que mejor se ajuste a los puntos dados según el método de mínimos cuadrados. Esto es que el error (la suma de los cuadrados de la distancia de la función a los puntos) sea mínimo.

$$Error(a,b) = \sum (ae^{bi} - y_i)^2 \quad \leftarrow \quad \text{mínimo}$$

Para que el error sea mínimo la derivada del error debe ser cero, luego:

$$\partial E/\partial a = \sum (2 * (ae^{bxi} - y_i) * e^{bxi}) = 0 = \sum ((ae^{bxi} - y_i) * e^{bxi}) = f(a,b)$$
$$\partial E/\partial b = \sum (2 * (ae^{bxi} - y_i) * ax_i * e^{bxi}) = 0 = \sum ((ae^{bxi} - y_i) * ax_i * e^{bxi}) = g(a,b)$$

Ahora hay que calcular las derivadas parciales de f y g para obtener la siguiente matriz:

$$DF(a,b) = \begin{pmatrix} \partial f/\partial a & \partial f/\partial b \\ \partial g/\partial a & \partial g/\partial b \end{pmatrix}$$

Siendo las derivadas parciales:

$$\partial f/\partial a = \sum (e^{2bxi}) = 0$$
$$\partial f/\partial b = \sum (2x_i * a * e^{2bxi} - x_i * y_i * e^{bxi}) = 0$$
$$\partial g/\partial a = \sum (2x_i * a * e^{2bxi} - x_i * y_i * e^{bxi}) = 0$$
$$\partial g/\partial b = \sum (2a^2 * x_i^2 * e^{2bxi} - a * y_i * x_i^2 * e^{bxi}) = 0$$

Una vez se tiene la matriz hay que aplicar el método de Newton para obtener un resultado definitivo.

El código del método es el siguiente:

```
function z=min_cuadrados_exp(x,y,tol,nmax)
inic=obt_min(x,y);
z=newton_raphson([inic(1);inic(2)],tol,nmax,x,y);
```

Que como puede verse utiliza dos funciones que son:

```
function z=obt_min(punto,h)
n=length(x);
E=100000;
a=0; b=0;
for j=1:200
   for h=1:200
      for i=1:n
         En=(a*exp(1).^(b*x(i))-y(i)).^2;
      end
      if En<E
         E=En;
         an=a; bn=b;
      end
      a=a+0.25;
   end
   a=0;
   b=b+0.25;
end
z=[an bn];
```

para obtener una aproximación inicial para a y b y:

```
function punto=newton_raphson(punto,tol,nmax,x,y)
   p=punto;
   for i=1:nmax
      if (i>1)
         while (norm(p-punto)>tol)
            punto=p;
            v=(-DF(p,x,y)\fun(p,x,y)');
            p=punto+v;
         end
      else
         v=-(DF(p,x,y)\fun(p,x,y)');
         p=punto+v;
      end
   end
   punto=p;
```

También podría haberse puesto:
(norm(fun(punto))>tol)
o bien
((norm(x-punto)/norm(x))>tol)

que utiliza a su vez las funciones:

```
function g=DF(punto,x,y)
z=zeros(2,2);
z(1,1)=fa(punto,x,y);
z(1,2)=fb(punto,x,y);
z(2,1)=ga(punto,x,y);
z(2,2)=gb(punto,x,y);

function z=fun(punto,x,y)
z=zeros(1,2);
z(1)=f(punto,x,y);
z(2)=g(punto,x,y);
```

```
function z=f(punto,x,y)
n=length(x);
a=punto(1);
b=punto(2);
z=0;
for i=1:n
   z=z+a*exp(1).^(2*b*x(i))-y(i)*exp(1).^(b*x(i));
end

function z=g(punto,x,y)
n=length(x);
a=punto(1);
b=punto(2);
z=0;
for i=1:n
   z=z+a.^2*x(i)*exp(1).^(2*b*x(i))-a*y(i)*x(i)*exp(1).^(b*x(i));
end

function z=fa(punto,x,y)
n=length(x);
a=punto(1);
b=punto(2);
z=0;
for i=1:n
   z=z+exp(1).^(2*b*x(i));
end

function z=fb(punto,x,y)
n=length(x);
a=punto(1);
b=punto(2);
z=0;
for i=1:n
   z=z+2*x(i)*a*exp(1).^(2*b*x(i))-x(i)*y(i)*exp(1).^(b*x(i));
end

function z=ga(punto,x,y)
n=length(x);
a=punto(1);
b=punto(2);
z=0;
for i=1:n
   z=z+2*a*x(i)*exp(1).^(2*b*x(i))-y(i)*x(i)*exp(1).^(b*x(i));
end

function z=gb(punto,x,y)
n=length(x);
a=punto(1);
b=punto(2);
z=0;
for i=1:n
   z= z+2*a.^2*x(i).^2*exp(1).^(2*b*x(i))-
      a*y(i)*x(i).^2*exp(1).^(b*x(i));
end
```

EJERCICIOS:

1.- Con la siguiente tabla de puntos:

x	y
1	5.10
1.25	5.79
1.50	6.53

calcular la función exponencial que mejor se ajusta.

Solución:

1.- Obtener el resultado

```
min_cuadrados_exp([1 1.25 1.50],[5.10 5.79 6.53],10^-3,20)
```

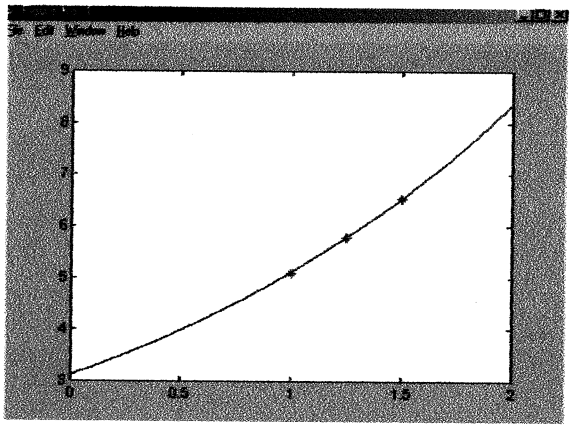
3.1186
0.4932

Luego la curva obtenida es: $y = 3.1186 * e^{0.4932x}$

2.- Comprobación gráfica

```
function y=grafica(x)
y=3.1186*exp(1).^(0.4932*x);
```

```
x1=0:0.1:2; y1=grafica(x1); x2=[1 1.25 1.50]; y2=[5.10 5.79 6.53];
plot(x1,y1,x2,y2,'*r');
```



3.- Cálculo del error.

```
function E=calc_error(x,y,f)
n=length(x);
for i=1:n
   func=feval(f,x(i));
   E=(func-y(i)).^2;
end
```

```
calc_error([1 1.25 1.50],[5.10 5.79 6.53],'grafica')
```

0.000025

2.- Con la siguiente tabla de puntos:

x	y
0	1
0.25	1.284
0.5	1.6487
0.75	2.117
1	2.7183

calcular la función exponencial que mejor se ajusta.

Solución:

1.- Obtener el resultado

```
min_cuadrados_exp ([0 0.25 0.5 0.75 1],[1 1.2840 1.6487 2.1170 2.7183],10^-3,20)
```

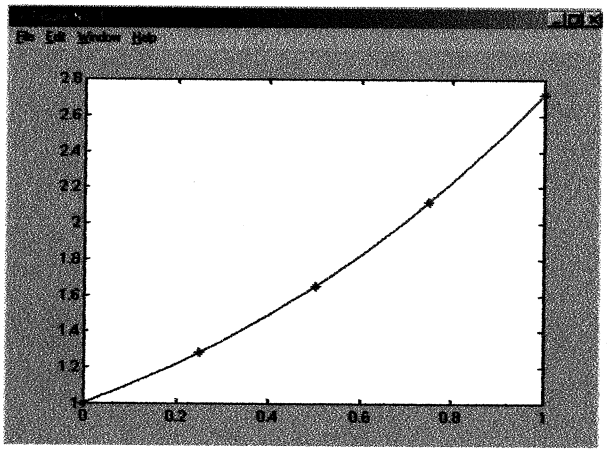
1.0000
1.0000

La curva obtenida es: $y = e^x$

2.- Comprobación gráfica

```
function y=grafica(x)
y=exp(1).^x;
```

```
x1=0:0.1:1; y1=grafica(x1); x2=[0 0.25 0.5 0.75 1];
y2=[1 1.2840 1.6487 2.1170 2.7183]; plot(x1,y1,x2,y2,'*r');
```



3.- Cálculo del error.

```
calc_error([0 0.25 0.5 0.75 1],[1 1.2840 1.6487 2.1170 2.7183],'grafica')
```

0.00000000033

Ajuste de mínimos cuadrados de forma no lineal usando Newton (Ajustando con función exponencial) [Aproximación 1]

Vamos a tratar de hacer lo mismo pero sin disponer de las derivadas parciales. Esto implicará que el resultado sea menos exacto.

Como en el caso anterior vamos a intentar obtener una función que se ajuste a los puntos dados y que tenga la forma:

$$y = ae^{bx}$$

Para que la función se ajuste bien a los puntos el error debe ser mínimo (siguiendo la regla de los mínimos cuadrados). Esto es:

$$Error(a,b) = \sum (ae^{bi} - y_i)^2 \quad \longleftarrow \quad \text{mínimo}$$

Para que el error sea mínimo la derivada del error debe ser cero, luego:

$$\partial E/\partial a = \Sigma\,(2 * (ae^{bxi} - y_i) * e^{bxi}) = 0 = f(a,b)$$
$$\partial E/\partial b = \Sigma\,(2 * (ae^{bxi} - y_i) * ax_i * e^{bxi}) = 0 = g(a,b)$$

El método va a manejar los puntos que se le pasan como entrada y las funciones f y g. Con estos datos el método debe obtener una aproximación inicial para a y b, la cual utilizará para poner en marcha el nuevo método de Newton, el cual creará una aproximación a las derivadas parciales de f y g sin tener control del error.

Las derivadas las calculará de la siguiente forma:

$$\partial f/\partial a = (f(a_0 + h, b_0) - f(a_0 - h, b_0)) / (2 * h)$$
$$\partial f/\partial b = (f(a_0, b_0 + h) - f(a_0, b_0 - h)) / (2 * h)$$
$$\partial g/\partial a = (g(a_0 + h, b_0) - g(a_0 - h, b_0)) / (2 * h)$$
$$\partial g/\partial b = (g(a_0, b_0 + h) - g(a_0, b_0 - h)) / (2 * h)$$

$$\Delta F(a,b) = \begin{pmatrix} \partial f/\partial a & \partial f/\partial b \\ \partial g/\partial a & \partial g/\partial b \end{pmatrix}$$

Los códigos del método y de las funciones auxiliares son:

- Para obtener las aproximaciones iniciales:

```
function z=obt_min(punto,h)
n=length(x);
E=100000;
a=0;
b=0;
for j=1:200
   for h=1:200
      i=1;
      while (i<n)
         En=(a*exp(1).^(b*x(i))-y(i)).^2;
         i=i+1;
      end
      if En<E
         E=En;
         an=a;
         bn=b;
      end
      a=a+0.25;
   end
   a=0;
   b=b+0.25;
end
z=[an bn];
```

- Para obtener un valor final para a y b.

```
function punto=newton_raphson_quasi(punto,tol,nmax,h,x,y)
   p=punto;
   for i=1:nmax
      if (i>1)
         while (norm(p-punto)>tol)
            punto=p;
            v=(-var_F(p,h,x,y)\fun(p,x,y)');
            p=punto+v;
         end
      else
         v=-(var_F(p,h,x,y)\fun(p,x,y)');
         p=punto+v;
      end
   end
   punto=p;
```

También podría haberse puesto:
(norm(fun(punto))>tol)
o bien
((norm(x-punto)/norm(x))>tol)

- Auxiliares para Newton

```
function g=var_F(punto,h,x,y)
z=zeros(2,2);
a=punto(1);
b=punto(2);
z(1,1)=(f([a+h b],x,y)-f([a-h b],x,y))/(2*h);
z(1,2)=(f([a b+h],x,y)-f([a b-h],x,y))/(2*h);
z(2,1)=(g([a+h b],x,y)-g([a-h b],x,y))/(2*h);
z(2,2)=(g([a b+h],x,y)-g([a b-h],x,y))/(2*h);
```

```
function z=fun(punto)
z=zeros(1,2);
z(1)=f(punto);
z(2)=g(punto);

function z=f(punto,x,y)
n=length(x);
a=punto(1);
b=punto(2);
z=0;
for i=1:n
   z=z+a*exp(1).^(2*b*x(i))-y(i)*exp(1).^(b*x(i));
end

function z=g(punto,x,y)
n=length(x);
a=punto(1);
b=punto(2);
z=0;
for i=1:n
   z= z+a.^2*x(i)*exp(1).^(2*b*x(i))-
      a*y(i)*x(i)*exp(1).^(b*x(i));
end
```

- Código del método

```
function z=min_cuadrados_exp_quasi(x,y,tol,nmax,h)
inic=obt_min(x,y);
z=newton_raphson_quasi([inic(1);inic(2)],tol,nmax,h,x,y);
```

EJERCICIOS:

1.- Con la siguiente tabla de puntos:

x	y
1	5.10
1.25	5.79
1.50	6.53

calcular la función exponencial que mejor se ajusta.

Solución:

1.- Obtener el resultado

min_cuadrados_exp_quasi([1 1.25 1.50],[5.10 5.79 6.53],10^-3,20,10^-5)

3.1186
0.4932

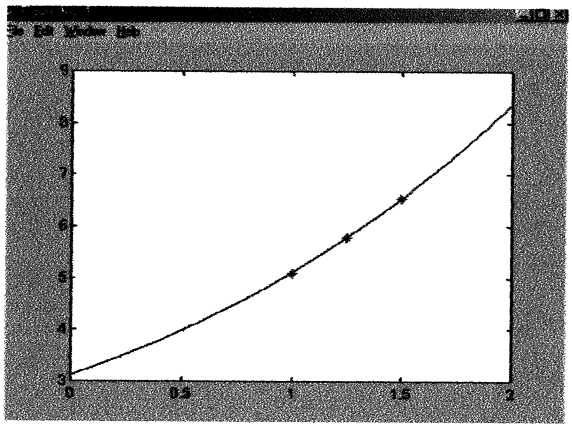
Luego la curva obtenida es:

$$y = 3.1186 * e^{0.4932x}$$

2.- Comprobación gráfica

```
function y=grafica(x)
y=3.1186*exp(1).^(0.4932*x);
```

x1=0:0.1:2; y1=grafica(x1); x2=[1 1.25 1.50]; y2=[5.10 5.79 6.53]; plot(x1,y1,x2,y2,'*r');



3.- Cálculo del error.

calc_error([1 1.25 1.50],[5.10 5.79 6.53],'grafica') ⟶ **0.000025**

2.- Con la siguiente tabla de puntos:

x	y
2	3.7
3	4
4	4.7
5	6.5

obtener la función exponencial que mejor se ajuste a ellos.

Solución:

1.- Obtener el resultado

min_cuadrados_exp_quasi([2 3 4 5],[3.7 4 4.7 6.5],10^-3,20,10^-5)

2.2428
0.2051

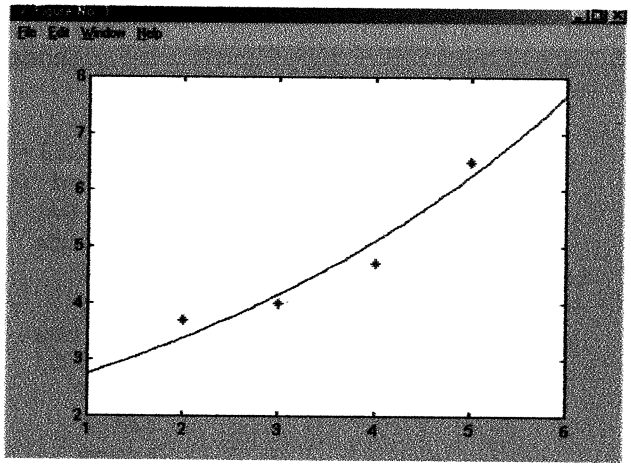
Luego la curva obtenida es:

$$y = 2.2428 * e^{0.2051x}$$

2.- Comprobación gráfica

```
function y=grafica(x)
y=2.2428*exp(1).^(0.2051*x);
```

x1=1:0.1:6; y1=grafica(x1); x2=[2 3 4 5]; y2=[3.7 4 4.7 6.5];
plot(x1,y1,x2,y2,'*r');



3.- Cálculo del error.

calc_error([2 3 4 5],[3.7 4 4.7 6.5],'grafica') → **0.0605**

- Para obtener un valor final para a y b.

```
function punto=newton_raphson_quasi2(punto,tol,nmax,h,x,y)
   p=punto;
   for i=1:nmax
      if (i>1)
         while (norm(p-punto)>tol)
             punto=p;
             v=(-var_F2(p,h,x,y)\fun(p,x,y)');
            p=punto+v;
         end
      else
       v=-(var_F2(p,h,x,y)\fun(p,x,y)');
         p=punto+v;
       end
   end
   punto=p;
```

También podría haberse puesto:
(norm(fun(punto))>tol)
o bien
((norm(x-punto)/norm(x))>tol)

- Auxiliares para Newton

```
function g=var_F2(punto,h,x,y)
z=zeros(2,2);
z(1,1)=derivada_a('f',h,10^-6,30,punto,x,y);
z(1,2)=derivada_b('f',h,10^-6,30,punto,x,y);
z(2,1)=derivada_a('g',h,10^-6,30,punto,x,y);
z(2,2)=derivada_b('g',h,10^-6,30,punto,x,y);

function z=fun(punto,x,y)
z=zeros(1,2);
z(1)=f(punto,x,y);
z(2)=g(punto,x,y);

function z=f(punto,x,y)
n=length(x);
a=punto(1);
b=punto(2);
z=0;
for i=1:n
   z=z+a*exp(1).^(2*b*x(i))-y(i)*exp(1).^(b*x(i));
end

function z=g(punto,x,y)
n=length(x);
a=punto(1);
b=punto(2);
z=0;
for i=1:n
   z=z+a.^2*x(i)*exp(1).^(2*b*x(i))-
     a*y(i)*x(i)*exp(1).^(b*x(i));
end
```

EJERCICIOS:

1.- Con la siguiente tabla de puntos:

x	y
1	5.10
1.25	5.79
1.50	6.53

calcular la función exponencial que mejor se ajusta.

Solución:

1.- Obtener el resultado

min_cuadrados_exp_quasi2([1 1.25 1.50],[5.10 5.79 6.53],10^-3,20,10^-5)

3.1186
0.4932

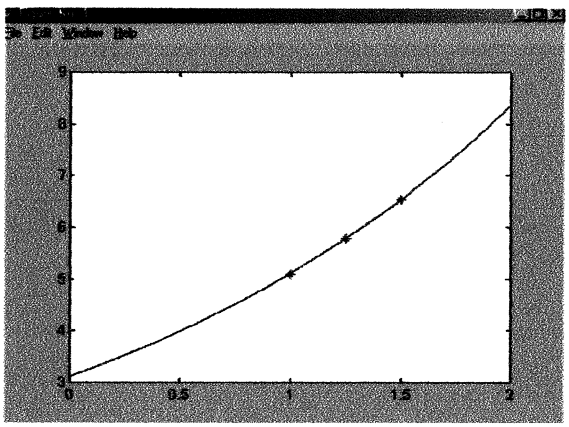
Luego la curva obtenida es:

$$y = 3.1186 * e^{0.4932x}$$

2.- Comprobación gráfica

```
function y=grafica(x)
y=3.1186*exp(1).^(0.4932*x);
```

x1=0:0.1:2; y1=grafica(x1); x2=[1 1.25 1.50]; y2=[5.10 5.79 6.53]; plot(x1,y1,x2,y2,'*r');



3.- Cálculo del error.

calc_error([1 1.25 1.50],[5.10 5.79 6.53],'grafica') ⟶ **0.000025**

2.- Ajustar la siguiente tabla de puntos con una función exponencial.

x	y
0	0.5
2	3.69
4	27.25
6	201.7
8	1470.4

Solución:

1.- Obtener el resultado

```
min_cuadrados_exp_quasi2([0 2 4 6 8],[0.5 3.69 27.25 201.7 1470.4],10^-3,20,10^-5)
```

0.5158
0.9944

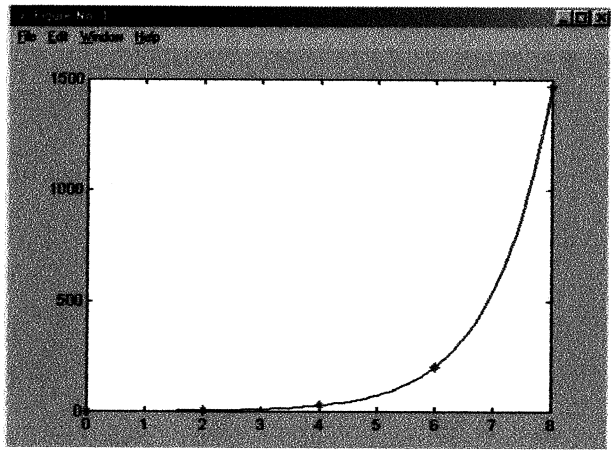
La curva obtenida es:

$$y = 0.5158 * e^{0.9944x}$$

2.- Comprobación gráfica

```
function y=grafica(x)
y=0.5158*exp(1).^(0.9944*x);
```

```
x1=0:0.1:8; y1=grafica(x1); x2=[0 2 4 6 8];
y2=[0.5 3.69 27.25 201.7 1470.4]; plot(x1,y1,x2,y2,'*r');
```



3.- Cálculo del error.

```
calc_error([0 2 4 6 8],[0.5 3.69 27.25 201.7 1470.4],'grafica')
```

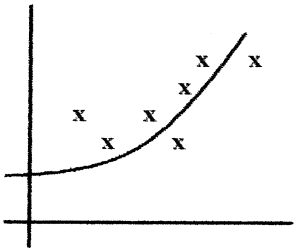
0.0343

Linealización

Se trata de transformar un problema de ajuste no lineal en un problema de ajuste lineal.

La situación es la siguiente:

x	y
x0	y0
x1	y1
...	...
xm	ym



Ahora se trata de transformar la función $y = ae^{bx}$ en una función lineal.

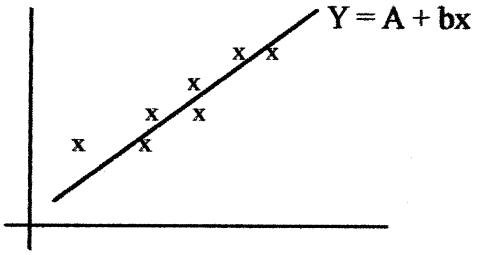
$$y = ae^{bx} \xrightarrow{\text{tomando logaritmo neperiano}} \ln(y) = \ln(ae^{bx}) \longrightarrow \ln(y) = \ln(a) + \ln(e^{bx})$$

$$\longrightarrow \ln(y) = \ln(a) + bx * \ln(e) \longrightarrow \ln(y) = \ln(a) + bx$$

$$\ln(e) \rightarrow 1 \qquad \ln(y) \rightarrow Y \qquad \ln(a) \rightarrow A$$

La nueva función es: $Y = A + bx$, cuya tabla de valores es la siguiente:

x	y	ln(y)
x0	y0	ln(y0)
x1	y1	ln(y1)
...	...	...
xm	ym	ln(ym)



Ahora hay que calcular A y b de forma lineal para que la recta se ajuste a los nuevos puntos con un error mínimo. Una vez se sabe lo que valen A y b se deshacen los cambios para obtener los coeficientes de la curva que ajusta los puntos iniciales.

$$a = e^A$$
$$b = b$$

El problema es que la solución que se obtenga para a y b no es exacta, es una aproximación, así que lo que se puede hacer es usar este método para que Newton obtenga su solución inicial (esto es cambiar la función obt_min por la función de linealización).

El código del método para transformar un problema de ajuste no lineal en uno lineal es el siguiente:

```
function z=linealizar(x,y,grado)
n=length(y);
lny=zeros(1,n);
for i=1:n
   lny(i)=log(y(i)); %log es logaritmo neperiano
end
lineal=min_cuadrados(x,lny,grado);
A=lineal(1);
b=lineal(2);
a=exp(1).^A;
z(1)=a;
z(2)=b;
```

que como puede verse hace uso del método que calcula la recta que mejor se ajusta a una serie de puntos usando mínimos cuadrados de forma lineal.

EJERCICIOS:

1.- Con la siguiente tabla de valores:

x	y
1	5.10
1.25	5.79
1.50	6.53

calcular la función exponencial que mejor se ajuste a ellos usando el método de Newton_Raphson, pero ahora tomando como aproximación inicial la solución proporcionada por el método de linealización.

Solución:

1.- Definir las funciones f y g

```
function z=f(punto,x,y)
n=length(x);
a=punto(1);
b=punto(2);
z=0;
for i=1:n
   z=z+a*exp(1).^(2*b*x(i))-y(i)*exp(1).^(b*x(i));
end

function z=g(punto,x,y)
n=length(x);
a=punto(1);
b=punto(2);
z=0;
for i=1:n
   z= z+a.^2*x(i)*exp(1).^(2*b*x(i))-
      a*y(i)*x(i)*exp(1).^(b*x(i));
end
```

2.- Definir las derivadas parciales de f y g.

```
function z=fa(punto,x,y)
n=length(x);
a=punto(1);
b=punto(2);
z=0;
for i=1:n
   z=z+exp(1).^(2*b*x(i));
end

function z=fb(punto,x,y)
n=length(x);
a=punto(1);
b=punto(2);
z=0;
for i=1:n
   z=z+2*x(i)*a*exp(1).^(2*b*x(i))-x(i)*y(i)*exp(1).^(b*x(i));
end
```

```
function z=ga(punto,x,y)
n=length(x);
a=punto(1);
b=punto(2);
z=0;
for i=1:n
   z=z+2*a*x(i)*exp(1).^(2*b*x(i))-y(i)*x(i)*exp(1).^(b*x(i));
end

function z=gb(punto,x,y)
n=length(x);
a=punto(1);
b=punto(2);
z=0;
for i=1:n
   z= z+2*a.^2*x(i).^2*exp(1).^(2*b*x(i))-
      a*y(i)*x(i).^2*exp(1).^(b*x(i));
end
```

3.- Obtener la aproximación inicial:

linealizar([1 1.25 1.50],[5.10 5.79 6.53],1) ⟶ **3.1143** **0.4943**

4.- Obtener el resultado:

newton_raphson([3.1143;0.4943],10^-3,20,[1 1.25 1.50],[5.10 5.79 6.53])

3.1186
0.4932

Luego la curva obtenida es:

$$y = 3.1186 * e^{0.4932x}$$

2.- Con la siguiente tabla de valores:

x	y
-3	0.003
-1	0.06
0	0.3
1	1.3
3	27
4	121
5	542

obtener la función exponencial que mejor se ajuste a ellos usando el método de Newton_Raphson, tomando como aproximación inicial la solución proporcionada por el método de linealización.

Solución:

1.- Obtener la aproximación inicial:

linealizar([-3 -1 0 1 3 4 5],[0.003 0.06 0.3 1.3 27 121 542],1)

0.2842 1.5143

2.- Obtener el resultado:

newton_raphson([0.2842;1.5143],10^-3,20,[-3 -1 0 1 3 4 5],[0.003 0.06 0.3 1.3 27 121 542])

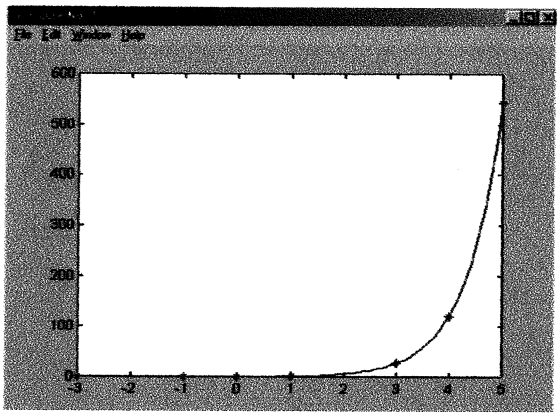
0.3005
1.4995

La función resultante es:

$$y = 0.3005 * e^{1.4995x}$$

3.- Comprobar con gráfica

```
function y=grafica(x)
y=0.3005*exp(1).^(1.4995*x);

x1=-3:0.1:5; y1=grafica(x1); x2=[-3 -1 0 1 3 4 5];
y2=[0.003 0.06 0.3 1.3 27 121 542]; plot(x1,y1,x2,y2,'*r');
```



4.- Cálculo del error.

```
calc_error([-3 -1 0 1 3 4 5],[0.003 0.06 0.3 1.3 27 121 542],'grafica')
```

0.0016

3.- Con la siguiente tabla de valores:

x	y
3	9
3.7	7.5
4	7
5	7
6.5	5.3
8	4.7
9	3.5
12	3
12.5	3

obtener la función exponencial que mejor se ajuste a ellos usando el método de Newton_Raphson, tomando como aproximación inicial la solución proporcionada por el método de linealización.

Solución:

1.- Obtener la aproximación inicial:

linealizar([3 3.7 4 5 6.5 8 9 12 12.5], [9 7 7.5 7 5.3 4.7 3.5 3 3],1)

11.6480 -0.1150

2.- Obtener el resultado:

newton_raphson([11.6480;-0.1150],10^-3,20,[3 3.7 4 5 6.5 8 9 12 12.5],[9 7 7.5 7 5.3 4.7 3.5 3 3])

12.2066
-0.1224

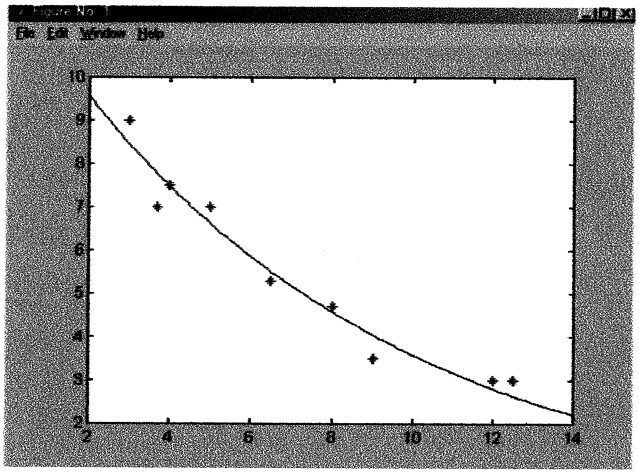
La función resultante es:

$$y = 12.2066 * e^{-0.1224x}$$

3.- Comprobar con gráfica

```
function y=grafica(x)
y=12.2066*exp(1).^(-0.1224*x);
```

x1=2:0.1:14; y1=grafica(x1); x2=[3 3.7 4 5 6.5 8 9 12 12.5];
y2=[9 7 7.5 7 5.3 4.7 3.5 3 3]; plot(x1,y1,x2,y2,'*r');



4.- Cálculo del error.

```
calc_error([3 3.7 4 5 6.5 8 9 12 12.5],[9 7 7.5 7 5.3 4.7 3.5 3 3],'grafica')
```

0.1273

MÉTODOS DE COLOCACIÓN

Métodos de colocación

Dado $v \in Y$ queremos encontrar un $u \in X$ tal que $Lu = v$, siendo L un operador que transforma ciertos elementos del conjunto X en elementos del conjunto Y.

$$L : X \longrightarrow Y$$

Para resolver este problema necesitamos conocer:

- N elementos independientes del conjunto X, los cuales los vamos a representar como:

$$< \phi_1, \phi_2, \dots, \phi_N > \in X$$

Estos elementos nos permitirán obtener u de la siguiente forma:

$$u \approx \sum_{i=1}^{N} c_i * \phi_i \qquad \text{con } c_i \text{ desconocidos}$$

- N nodos o términos, los cuales se van a representar como:

$$\{ t_1, t_2, \dots, t_N \}$$

Con todo esto se obtiene v, el cual está formado por N ecuaciones, teniendo las ecuaciones la siguiente forma:

$$v(t_j) = L(\sum_{i=1}^{N} c_i * \phi_i) * (t_j) \qquad \text{Siendo } i, j = 1, 2, \dots, N$$

Con este método se pueden resolver problemas lineales y problemas no lineales.

- Si el problema es lineal se obtendrá: $AC = b \rightarrow C = A \setminus b$
- Si el problema es no lineal el resultado será: $F(C) = 0 \longrightarrow C = \begin{bmatrix} C_1 \\ C_N \end{bmatrix}$

En ambos casos se obtendrá un y resultante con la siguiente forma:

$$y \approx \sum_{i=1}^{N} c_i * \phi_i$$

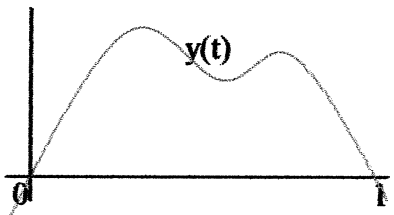
que es la función que se está buscando.

La ventaja de este método es que permite resolver cualquier tipo de problema, pero su inconveniente es que no se controla el error que hay entre la función obtenida y la función real. El error depende de los elementos del conjunto X y de los nodos que se elijan.

Aplicación del método anterior a problemas de contorno lineales

Se parte de un problema del tipo:

$$\begin{cases} y'' = p(t) * y' + q(t) * y + r(t) \\ y(0) = 0 \\ y(1) = 0 \end{cases}$$



el cual siempre se ha resuelto por disparo o por diferencias finitas. Ahora se trata de transformarlo para resolverlo por colocación.

Inicialmente se definen:

$$Ly = y'' - p(t) * y' - q(t) * y \qquad Ly = r$$
$$x = \{ y \in C^2 [0\ 1] : \quad y(0) = y(1) = 0 \}$$
$$y = \{ r \in C^0 [0\ 1] \}$$

Dada $r \in Y$, encontrar $y \in X$ tal que $Ly = r$

Ahora se plantean N ecuaciones:

$$Ly(t_1) = r(t_1)$$
$$\dots$$
$$Ly(t_j) = r(t_j)$$
$$\dots$$
$$Ly(t_N) = r(t_N)$$

Siendo:

$$Ly(t_j) = \sum_{i=1}^{N} c_i * \phi_i''(t_j) - p(t_j) * \sum_{i=1}^{N} c_i * \phi_i'(t_j) - q(t_j) * \sum_{i=1}^{N} c_i * \phi_i(t_j) = r(t_j)$$

De forma matricial es lo siguiente:

$$j \blacktriangleright \begin{pmatrix} & A & \\ a_{j1} & a_{ji} & a_{jN} \\ & & \end{pmatrix} * \begin{pmatrix} C \\ c_1 \\ c_i \\ c_N \end{pmatrix} = \begin{pmatrix} b \\ b_1 \\ b_j \\ b_N \end{pmatrix}$$

$$\boxed{\begin{array}{l} a_{ji} = \phi_i''(t_j) - p(t_j) * \phi_i'(t_j) - q(t_j) * \phi_i(t_j) \\ b_j = r(t_j) \end{array}}$$

El método calcula lo que valen aji y bj en cada momento para rellenar con ellos las matrices A y b; una vez están rellenas el método calcula el vector C de la forma C = A \ b. Cuando se tiene C se tiene la función que se buscaba:

$$y \approx c_1 * \phi_1 + c_2 * \phi_2 + \dots + c_N * \phi_N$$

El código del método es el siguiente:

```
function C=colocacion(p,q,r)
t=rellenar_nodos;
N=length(t);
A=zeros(N,N);
B=zeros(N,1);
for j=1:N
   ecuac=matriz_ecuac(t(j));
   der_ecuac=matriz_der_ecuac(t(j));
   der2_ecuac=matriz_der2_ecuac(t(j));
   for i=1:N
      A(j,i)=der2_ecuac(i)-feval(p,t(j))*der_ecuac(i)-
            feval(q,t(j))*ecuac(i);
   end
   b(j,1)=feval(r,t(j));
end
C=A\b;
```

Como puede verse, el método hace uso de una serie de funciones, cuyo código cambiará dependiendo del ejercicio que se esté realizando. Rellenar_nodos contiene los nodos o términos que se hayan elegido, matriz_ecuac es para las ecuaciones independientes elegidas del conjunto X, matriz_der_ecuac almacena las primeras derivadas de las ecuaciones de matriz_ecuac y por último matriz_der2_ecuac contiene las derivadas segundas de las ecuaciones de matriz_ecuac.

EJERCICIOS:

1.- Con el siguiente problema de contorno:

$$\begin{cases} y''(t) = t^2 * y' - 4\pi^2 * y - 2\pi t^2 * \cos(2\pi t) \\ y(0) = 0 \\ y(1) = 0 \end{cases}$$

Obtener una función que aproxime y(t) usando el método de colocación con:

a) tres ecuaciones y tres nodos
b) cuatro ecuaciones y cuatro nodos
c) cinco ecuaciones y cinco nodos

Comprobar el resultado con la solución real sabiendo que y(t) = sen(2πt)

Solución:

a)

1.- Elegir los tres nodos:

Los nodos pueden ser cualesquiera, yo los voy a elegir usando el método de Chebyshev cuyo código es el siguiente:

```
function [x]=cheb(n,a,b)
const=pi/(2*(n-1)+2);
for k=1:n
   d=(2*k-1)*const;
   x(n+1-k)=cos(d);
end
x=(((b-a)*x/2)+((b+a)/2));


function y=rellenar_nodos
c=cheb(3,0,1);
y=c';
```

2.- Elegir las tres ecuaciones:

Se puede elegir cualquier ecuación que se anule en x = 0 y en x = 1, como por ejemplo:

$$\phi_1(t) = t * (1 - t)$$
$$\phi_2(t) = t^2 * (1 - t)$$
$$\phi_3(t) = (t - t^2)^2$$

```
function y=matriz_ecuac(t)
y(1)=t*(1-t);
y(2)=t.^2*(1-t);
y(3)=(t-t.^2).^2;
```

3.- Definir las derivadas (primera y segunda)

```
function y=matriz_der_ecuac(t)
y(1)=1-2*t;
y(2)=2*t-3*t.^2;
y(3)=2*t+4*t.^3-6*t.^2;

function y=matriz_der2_ecuac(t)
y(1)=-2;
y(2)=2-6*t;
y(3)=12*t.^2-6+2;
```

4.- Definir las funciones p, q y r sabiendo que:

$$p = t^2$$
$$q = -4\pi^2$$
$$r = -2\pi t^2 * \cos(2\pi t)$$

```
function y=p(t)
y=t.^2;

function y=q(t)
y=-4*pi.^2;

function y=r(t)
y=-2*pi*t.^2*cos(2*pi*t);
```

5.- Ejecutar el método:

colocacion('p','q','r')

2.5392
-4.1449
-1.6095

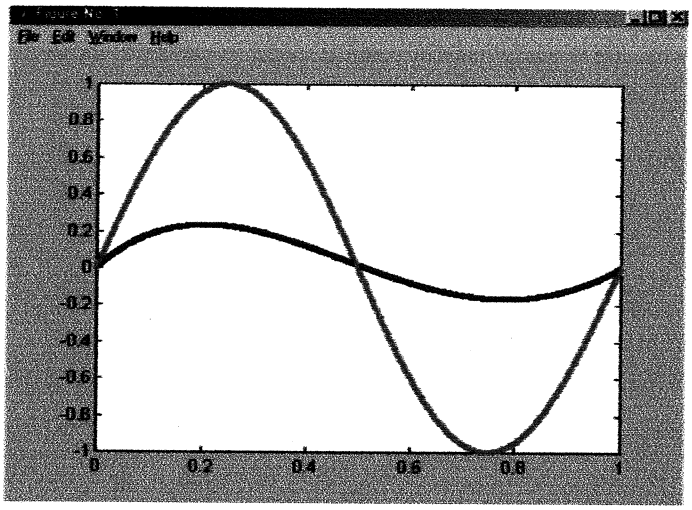
Con este resultado se contruye la función y(t) la cual es:

$$y(t) = 2.5392t * (1-t) - 4.1449t^2 * (1-t) - 1.6095 * (t-t^2)^2$$

```
function y=grafica(t)
y=2.5392.*(t.*(1-t))-4.1449.*(t.^2.*(1-t))-1.6095.*((t-t.^2).^2);
```

6.- Comprobación gráfica:

x=0:0.001:1; y1=grafica(x); y2=sin(2*pi*x); plot(x,y1,'.r',x,y2,'.g')



real
colocación

7.- Cálculo del error:

Se ve en la gráfica que el resultado no es muy bueno, que la aproximación dista bastante de la gráfica original pero voy a obtener su error para tener un resultado numérico. Para ello creo una nueva función que busque el máximo de diferencia entre las dos funciones.

```
function error=error_coloc(a,b,f,g)
error=0;
parte=abs((a-b)/100);
x=a;
for i=1:101
   real(i)=feval(f,x);
   nueva(i)=feval(g,x);
   dif=abs(real(i)-nueva(i));
   if (dif>error)
      error=dif;
   end
   x=x+parte;
end
```

error_coloc(0,1,'f','grafica') ⟶ **0.8369**

Siendo f la función original:

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

b)

1.- Elegir los cuatro nodos:

```
function y=rellenar_nodos
c=cheb(4,0,1);
y=c';
```

2.- Elegir la nueva ecuación:

$$\phi_4(t) = t^3 * (1-t)^2$$

```
function y=matriz_ecuac(t)
y(1)=t*(1-t);
y(2)=t.^2*(1-t);
y(3)=(t-t.^2).^2;
y(4)=t.^3*(1-t).^2;
```

3.- Añadir las derivadas de la nueva ecuación

```
function y=matriz_der_ecuac(t)
y(1)=1-2*t;
y(2)=2*t-3*t.^2;
y(3)=2*t+4*t.^3-6*t.^2;
y(4)=3*t.^2+5*t.^4-8*t.^3;

function y=matriz_der2_ecuac(t)
y(1)=-2;
y(2)=2-6*t;
y(3)=12*t.^2-6+2;
y(4)=6*t+20*t.^3-24*t.^2;
```

4.- Ejecutar el método:

colocacion('p','q','r')

0.2297
0.4056
-0.0110
-3.3823

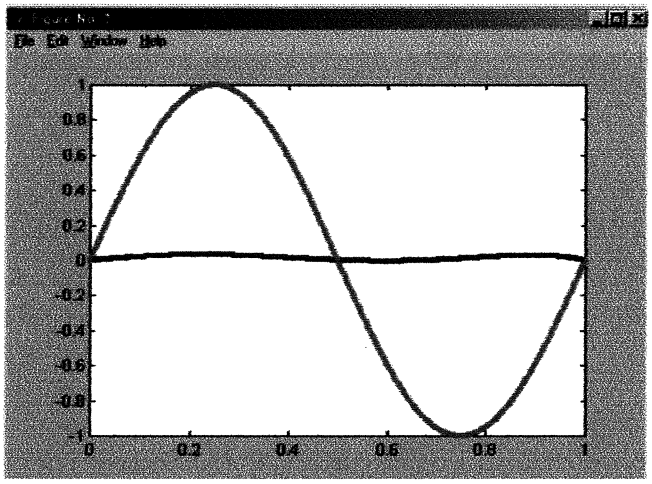
Con este resultado se contruye la nueva función y(t) la cual es:

$$y(t) = 0.2297t * (1-t) + 0.456t^2 * (1-t) - 0.0110 * (t-t^2)^2 - 3.3823t^3 * (1-t)^2$$

```
function y=grafica(t)
y=0.2297.*(t.*(1-t))+0.4056.*(t.^2.*(1-t))-0.0110.*((t-t.^2).^2)-
3.3823.*(t.^3.*(1-t).^2);
```

5.- Comprobación gráfica:

x=0:0.001:1; y1=grafica(x); y2=sin(2*pi*x); plot(x,y1,'.r',x,y2,'.g')



real
colocación

6.- Cálculo del error:

error_coloc(0,1,'f','grafica') ⟶ **1.0105**

Como puede verse el error es más grande que en el apartado anterior, quizas se deba a que la nueva ecuación incluida no sea muy adecuada.

c)

1.- Elegir los cuatro nodos:

```
function y=rellenar_nodos
c=cheb(5,0,1);
y=c';
```

2.- Elegir la nueva ecuación:

$$\phi_4(t) = t * (1-t)^3$$

```
function y=matriz_ecuac(t)
y(1)=t*(1-t);
y(2)=t.^2*(1-t);
y(3)=(t-t.^2).^2;
y(4)=t.^3*(1-t).^2;
y(5)=t*(1-t).^3;
```

3.- Añadir las derivadas de la nueva ecuación

```
function y=matriz_der_ecuac(t)
y(1)=1-2*t;
y(2)=2*t-3*t.^2;
y(3)=2*t+4*t.^3-6*t.^2;
y(4)=3*t.^2+5*t.^4-8*t.^3;
y(5)=1-6*t+9*t.^2-4*t.^3;

function y=matriz_der2_ecuac(t)
y(1)=-2;
y(2)=2-6*t;
y(3)=12*t.^2-6+2;
y(4)=6*t+20*t.^3-24*t.^2;
y(5)=-6+18*t-12*t.^2;
```

4.- Ejecutar el método:

colocacion('p','q','r')

34.4862
-42.9084
-0.2829
-50.9543
-26.2570

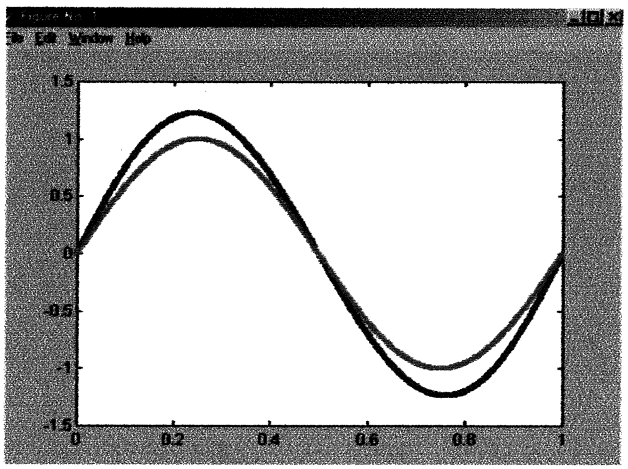
Con este resultado se contruye la nueva función y(t) la cual es:

$$y(t) = 34.4562t * (1 - t) - 42.9084t^2 * (1 - t) - 0.2829 * (t - t^2)^2 - 50.9543t^3 * (1 - t)^2 - 26.2570t * (1 - t)^3$$

```
function y=grafica(t)
y=34.4562.*(t.*(1-t))-42.9084.*(t.^2.*(1-t))-0.2829.*((t-t.^2).^2)-
50.9543.*(t.^3.*(1-t).^2)-26.2570.*(t.*(1-t).^3);
```

5.- Comprobación gráfica:

x=0:0.001:1; y1=grafica(x); y2=sin(2*pi*x); plot(x,y1,'.r',x,y2,'.g')



real
colocación

6.- Cálculo del error:

error_coloc(0,1,'f','grafica') ⟶ **0.2471**

Ahora sí se obtiene un error más pequeño, pero aún debería mejorarse incluyendo más ecuaciones y más nodos porque la gráfica no está todavía bien definida.

2.- Teniendo el siguiente problema de contorno:

$$\begin{cases} y''(t) = 2y' + y + 6t - 6t^2 - t^3 + 1 \\ y(-1) = 2 \\ y(1) = 0 \end{cases}$$

a) Obtener una función que aproxime y(t) usando el método de colocación con la cantidad de nodos y ecuaciones que se crean oportunos.
b) Comprobar el resultado con la solución real sabiendo que $y(t) = t^3 - 1$

Solución:

a)

1.- Definir las funciones p, q y r sabiendo que:

$$p = 2$$
$$q = 1$$
$$r = 6t - 6t^2 - t^3 + 1$$

```
function y=p(t)
y=2;

function y=q(t)
y=1;

function y=r(t)
y=6*t-6*t.^2-t.^3+1;
```

2.- Elegir los nodos:

Igual que en el ejercicio anterior voy a usar el método de chebyshev para elegir los nodos, puesto que este método proporciona nodos que son más óptimos que los nodos equidistantes. Voy a comenzar usando tres nodos.

```
function y=rellenar_nodos
c=cheb(3,-1,1);
y=c';
```

Se toma –1 y 1 porque el problema de contorno indica que y(-1) = 2 y que y(1) = 0

3.- Elegir ecuaciones:

Como he tomado tres nodos debo elegir tres ecuaciones, las cuales deben valer 2 en –1 [y(-1) = 2] y 0 en 1 [y(1) = 0].

$$\phi_1(t) = 1 - t$$
$$\phi_2(t) = -t * (1 - t)$$
$$\phi_3(t) = t^2 * (1 - t)$$

```
function y=matriz_ecuac(t)
y(1)=(1-t);
y(2)=-t*(1-t);
y(3)=t.^2*(1-t);
```

4.- Definir las derivadas (primera y segunda)

```
function y=matriz_der_ecuac(t)
y(1)=-1;
y(2)=-1+2*t;
y(3)=2*t-3*t.^2;

function y=matriz_der2_ecuac(t)
y(1)=0;
y(2)=2;
y(3)=2-6*t;
```

5.- Ejecutar el método:

colocacion('p','q','r')

-1.0000
1.0000
-1.0000

Con este resultado se contruye la función y(t) la cual es:

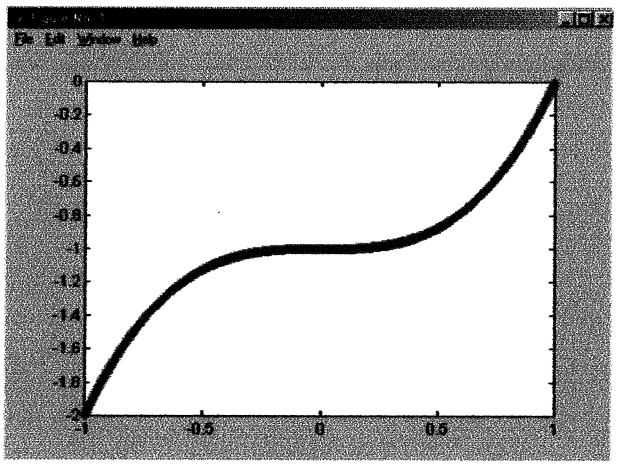
$$y(t) = -1 * (1-t) - 1t * (1-t) - 1t^2 * (1-t)^2$$

```
function y=grafica(t)
y=-1.*(1-t)+1.*(-t.*(1-t))-1.*(t.^2.*(1-t));
```

b)

1.- Comprobación gráfica:

x=-1:0.001:1; y1=grafica(x); y2=t.^3-1; plot(x,y2,'.g',x,y1,'.r')



real
colocación

2.- Cálculo del error:

Es evidente que el resultado es muy bueno pero me voy a asegurar obteniendo el error.

error_coloc(-1,1,'f','grafica') ⟶ $\underline{2.2204 * 10^{-16}}$

Siendo ahora f la función:

```
function y=f(t)
y=t.^3-1;
```

3.- Conclusión:

Como en el enunciado del problema no se especifica el error que se debe obtener lo voy a dejar aquí ya que me parece que el resultado es muy bueno y que no es necesario incluirle más ecuaciones al método.

3.- Con el problema de contorno:

$$\begin{cases} y''(t) = 3y' + 2/t^2 * y - 6t + 48/t^2 \\ y(1) = 17 \\ y(2) = 12 \end{cases}$$

obtener una función que aproxime y(t) usando el método de colocación con la cantidad de nodos y ecuaciones que se crean oportunos y comprobar el resultado con la solución real sabiendo que es $y(t) = t^2 + 16/t$

Solución:

1.- Definir las funciones p, q y r sabiendo que:

$p = 3$
$q = 2/t^2$
$r = -6t + 48/t^2$

```
function y=p(t)
y=3;

function y=q(t)
y=2/t.^2;

function y=r(t)
y=-6*t+48/t.^2;
```

2.- Elegir los nodos:

De nuevo voy a usar el método de chebyshev para elegir los nodos. En principio voy a tomar un único nodo que después iré incrementando.

```
function y=rellenar_nodos
c=cheb(1,1,2);
y=c';
```

Se toma 1 y 2 porque el problema de contorno indica que y(1) = 17 y que y(2) = 12

3.- Elegir ecuaciones:

Debo elegir una ecuación que valga 17 con x = 1 y que valga 12 con x = 2.

$$\phi_1(t) = t^3 + 16/t^2$$

```
function y=matriz_ecuac(t)
y(1)=t.^3+16/t.^2;
```

4.- Definir las derivadas (primera y segunda)

```
function y=matriz_der_ecuac(t)
y(1)=3*t.^2-32/t.^3;

function y=matriz_der2_ecuac(t)
y(1)=6*t+96/t.^4;
```

5.- Ejecutar el método:

colocacion('p','q','r')

0.4596

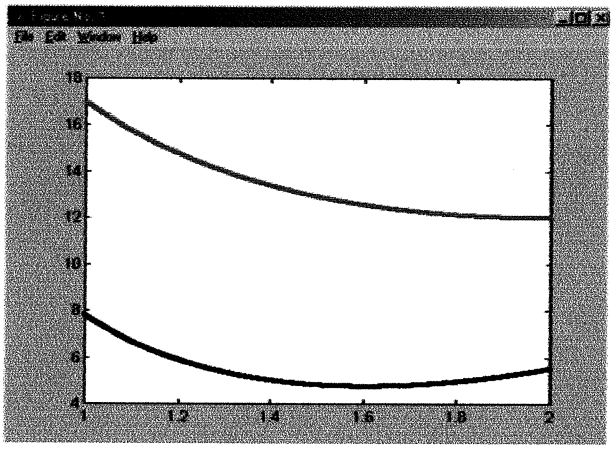
La función y(t):

$$y(t) = 0.4596 * (t^3 + 16 / t^2)$$

```
function y=grafica(t)
y=0.4596.*(t.^3+16./t.^2);
```

6.- Comprobación gráfica:

x=1:0.001:2; y1=grafica(x); y2=x.^2+16./x; plot(x,y2,'.g',x,y1,'.r')



real
colocación

7.- Cálculo del error:

Es evidente que el resultado es muy bueno pero me voy a asegurar obteniendo el error.

error_coloc(1,2,'f','grafica') ⟶ **9.1868**

Siendo ahora f la función:

```
function y=f(t)
y=t.^2+16/t;
```

Como el error es muy elevado voy a incluir una nueva ecuación para ver si mejora.

8.- Elegir nuevo nodo, nueva ecuación y definir sus derivadas:

```
function y=rellenar_nodos
c=cheb(2,1,2);
y=c';
```

$\phi_2(t) = 10 / t + 7$

```
function y=matriz_ecuac(t)
y(1)=t.^3+16/t.^2;
y(2)=10/t+7;

function y=matriz_der_ecuac(t)
y(1)=3*t.^2-32/t.^3;
y(2)=-10/t.^2;

function y=matriz_der2_ecuac(t)
y(1)=6*t+96/t.^4;
y(2)=20/t.^3;
```

9.- Volver a ejecutar el método:

colocacion('p','q','r')

0.2189
0.7534

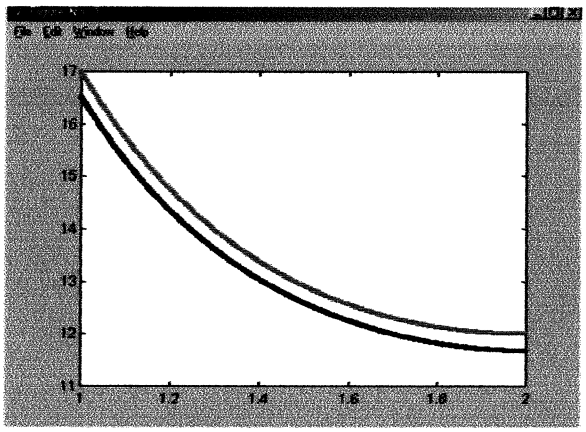
La función y(t):

$$y(t) = 0.2189 * (t^3 + 16 / t^2) + 0.7534 * (10 / t + 7)$$

```
function y=grafica(t)
y=0.2189.*(t.^3+16./t.^2)+0.7534.*(10./t+7);
```

10.- Comprobación gráfica:

x=1:0.001:2; y1=grafica(x); y2=x.^2+16./x; plot(x,y2,'.g',x,y1,'.r')



real
colocación

11.- Cálculo del error:

error_coloc(1,2,'f','grafica') ⟶ **0.4709**

Como puede verse el error ha disminuido notablemente, pero voy a incluir una nueva ecuación para ver qué pasa.

12.- Elegir nuevo nodo, nueva ecuación y definir sus derivadas:

```
function y=rellenar_nodos
c=cheb(3,1,2);
y=c';
```

$\phi_3(t) = 2t + 1 + 14 / t$

```
function y=matriz_ecuac(t)
y(1)=t.^3+16/t.^2;
y(2)=10/t+7;
y(3)=2*t+1+14/t;

function y=matriz_der_ecuac(t)
y(1)=3*t.^2-32/t.^3;
y(2)=-10/t.^2;
y(3)=2-14/t.^2;

function y=matriz_der2_ecuac(t)
y(1)=6*t+96/t.^4;
y(2)=20/t.^3;
y(3)=28/t.^3;
```

13.- Ejecutar el método:

colocacion('p','q','r')

-0.6586
-5.8577
7.9244

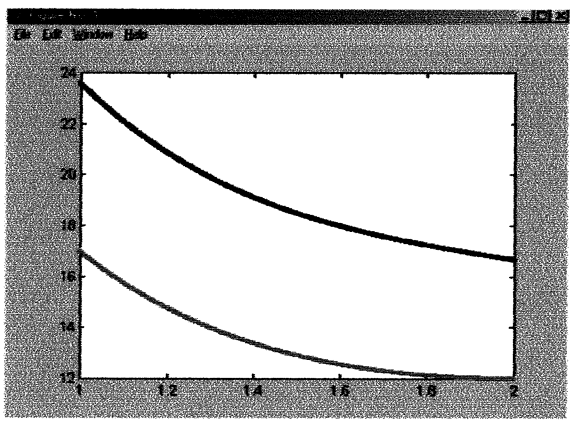
La función y(t):

$y(t) = -0.6586 * (t^3 + 16 / t^2) - 5.8577 * (10 / t + 7) + 7.9244 * (2t + 1 + 14 / t)$

```
function y=grafica(t)
y=0.2189.*(t.^3+16./t.^2)+0.7534.*(10./t+7);
```

14.- Comprobación gráfica:

x=1:0.001:2; y1=grafica(x); y2=x.^2+16./x; plot(x,y2,'.g',x,y1,'.r')



real
colocación

15.- Cálculo del error:

error_coloc(1,2,'f','grafica') ⟶ **6.5977**

El resultado ha empeorado al añadir esta ecuación, así que voy a añadir una más.

16.- Elegir nuevo nodo, nueva ecuación y definir sus derivadas:

```
function y=rellenar_nodos
c=cheb(4,1,2);
y=c';
```

$$\phi_4(t) = 3t + 8/t^3 + 2/t + 4$$

```
function y=matriz_ecuac(t)
y(1)=t.^3+16/t.^2;
y(2)=10/t+7;
y(3)=2*t+1+14/t;
y(4)=3*t+8/t.^3+2/t+4;

function y=matriz_der_ecuac(t)
y(1)=3*t.^2-32/t.^3;
y(2)=-10/t.^2;
y(3)=2-14/t.^2;
y(4)=3-24/t.^4-2/t.^2;

function y=matriz_der2_ecuac(t)
y(1)=6*t+96/t.^4;
y(2)=20/t.^3;
y(3)=28/t.^3;
y(4)=96/t.^5+4/t.^3;
```

17.- Ejecutar el método:

colocacion('p','q','r')

0.2049
-0.0125
0.9058
-0.1247

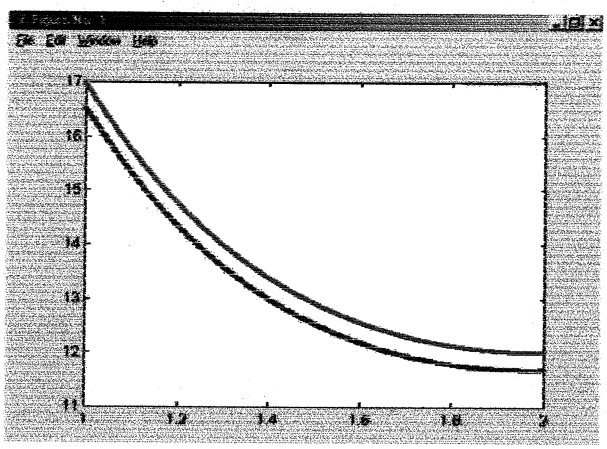
La función y(t):

$$y(t) = 0.2049 * (t^3 + 16/t^2) - 0.0125 * (10/t + 7) + 0.9058 * (2t + 1 + 14/t) - 0.1247 * (3t + 8/t^3 + 2/t + 4)$$

```
function y=grafica(t)
y=0.2049.*(t.^3+16./t.^2)-0.0125.*(10./t+7)+0.9058.*(2.*t+1+14./t)-
0.1247.*(3.*t+8./t.^3+2./t+4);
```

18.- Comprobación gráfica:

x=1:0.001:2; y1=grafica(x); y2=x.^2+16./x; plot(x,y2,'.g',x,y1,'.r')



real
colocación

19.- Cálculo del error:

error_colec(1,2,'','grafica') ⟶ **0.4505**

De nuevo se tiene un error pequeño, incluso un poco más pequeño que el error obtenido con las dos primeras ecuaciones, lo cual quiere decir que las ecuaciones y los nodos elegidos son adecuados. Si se quisiera obtener un error aún más pequeño se debería seguir añadiendo ecuaciones y nodos hasta obtener el error deseado.

GRAFICOS EN TRES DIMENSIONES

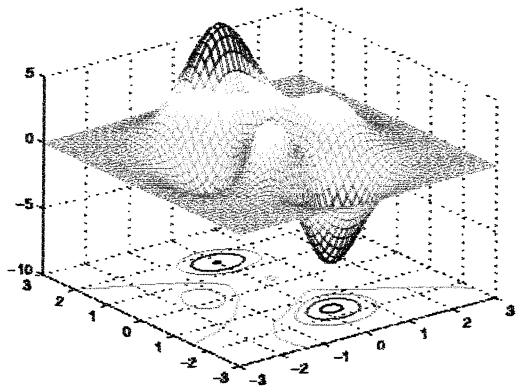
Hasta ahora se han realizado diversos gráficos para resolver los distintos problemas y estos gráficos siempre han sido en dos dimensiones. Pero a partir de ahora se van a necesitar gráficos en tres dimensiones, por lo que es conveniente hacer un pequeño estudio de las distintas funciones y opciones que ofrece MATLAB para realizar gráficos 3D.

- Función meshc

  - Código de ejemplo:

```
[X,Y] = meshgrid(-3:.125:3);
Z = peaks(X,Y);
meshc(X,Y,Z);
axis([-3 3 -3 3 -10 5])
```

  - Gráfico resultante:

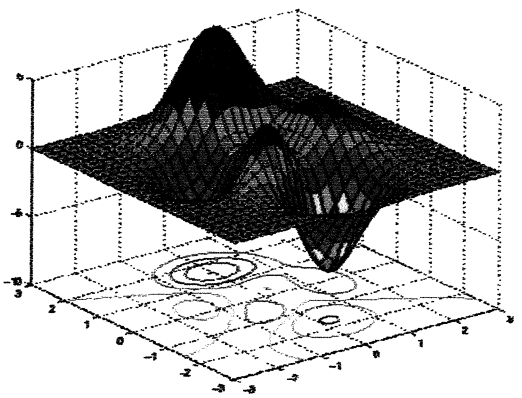


- Función surfc

  - Código de ejemplo:

```
[X,Y,Z] = peaks(30);
surfc(X,Y,Z)
colormap hsv        <-- Para elegir el estilo
axis([-3 3 -3 3 -10 5])
```

  - Gráfico resultante:



- Función surfl

  - Código de ejemplo:

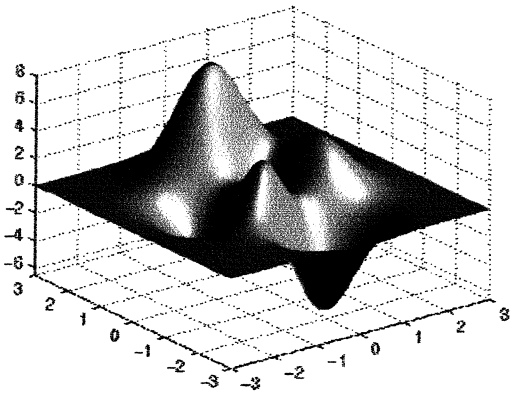
```
[x,y] = meshgrid(-3:1/8:3);
z = peaks(x,y);
surfl(x,y,z);
shading interp
colormap(gray);
axis([-3  3  -3  3  -8  8])
```

Para que no aparezcan líneas sino que sea una superficie lisa.

Colormap permite elegir el color del gráfico. Ejemplos de colores:

- Autumn: Tonos naranjas.
- Cool: Tonos morados.
- Copper: Tonos marrones.
- Gray: Tonos grises.
- Hot: Rojos y amarillos.
- Pink: Tonos rosados.
- Summer: Tonos verdes.
- Winter: Azules y verdes.

  - Gráfico resultante:

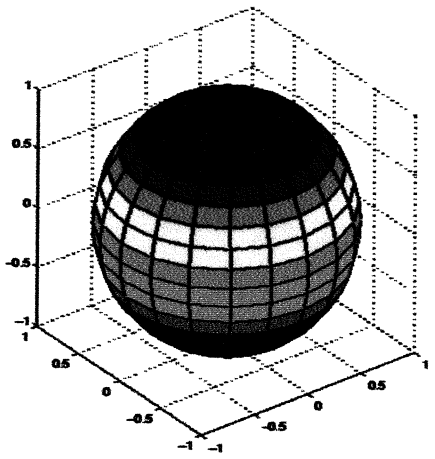


- **Función sphere: Permite dibujar una espera.**

  - **Código de ejemplo:**

```
sphere;
axis equal;
```

  - **Gráfico resultante:**

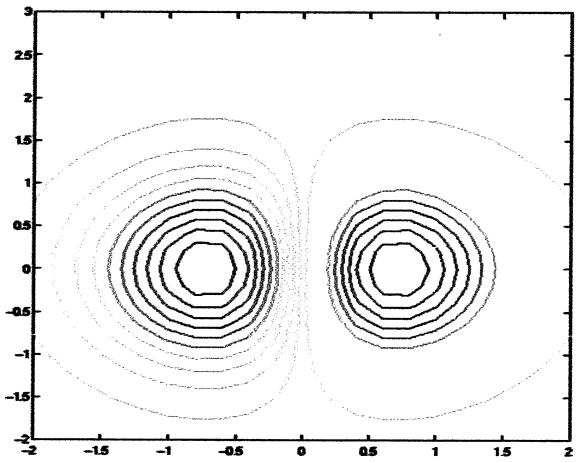


- **Función contour: Permite dibujar el contorno de una superficie.**

  - **Código de ejemplo:**

```
axis [X,Y] = meshgrid(-2:.2:2,-2:.2:3);
Z = X.*exp(-X.^2-Y.^2);
contour(X,Y,Z,20)
```

  - **Gráfico resultante:**

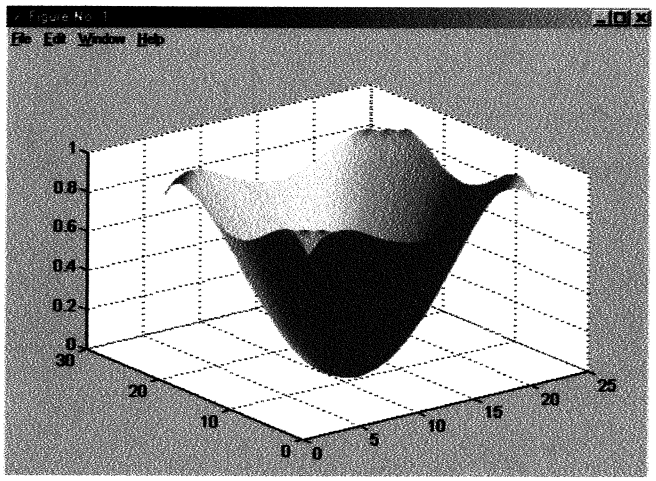


EJERCICIOS:

1.- Dibujar la función $z = \text{sen}(x^2 + y^2)$ siendo: $-1 \leq x \leq 1$ y $-1 \leq y \leq 1$

Solución:

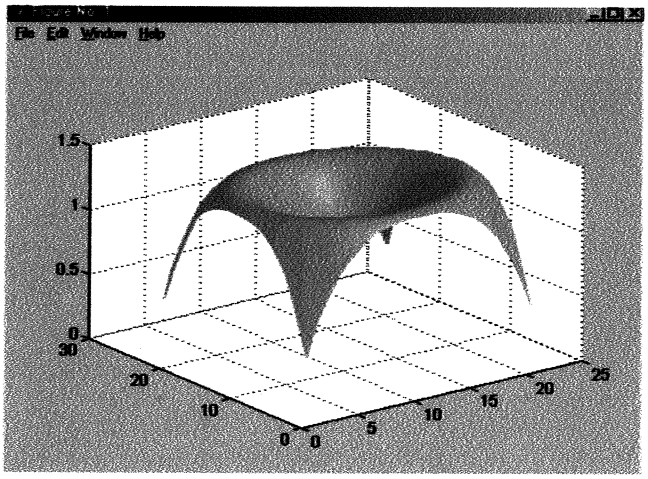
```
[x,y] = meshgrid(-1:.100:1),
z = sin(x.^2+y.^2),
surfl(z),
shading interp,
colormap(pink);
```



2.- Dibujar la función $z = \text{sen}(x^2 + y^2) + \cos(x^2 + y^2)$ siendo: $-1 \leq x \leq 1$ y $-1 \leq y \leq 1$

Solución:

```
[x,y] = meshgrid(-1:.100:1),
z = sin(x.^2+y.^2)+cos(x.^2+y.^2),
surfl(z),
shading interp,
colormap(summer);
```



ECUACIONES CON DERIVADAS PARCIALES

En este apartado vamos a resolver ecuaciones con derivadas parciales, las cuales serán de los siguientes tipos:

- Ecuaciones Parabólicas: Una ecuación de este tipo es la ecuación del calor, la cual tiene la siguiente forma:

$$\frac{\partial u}{\partial t} = \alpha * \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) + \ldots$$

A esta ecuación hay que añadirle algunas condiciones iniciales y algunas condiciones de contorno.

- Ecuaciones Hiperbólicas: Una ecuación de este tipo es, por ejemplo, la ecuación de ondas, también llamada ecuación de la cuerda vibrante, cuya forma es la siguiente:

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \ldots$$

A esta ecuación hay que añadirle también algunas condiciones iniciales y algunas condiciones de contorno.

- Ecuaciones Elípticas: Son también llamadas ecuaciones de Laplace o de Poison. Un ejemplo de ecuación de este tipo es la ecuación del potencial. La forma de estas ecuaciones es la siguiente:

$$\frac{\partial^2 u}{\partial^2 x^2} + \frac{\partial^2 u}{\partial y^2} = \ldots$$

A la ecuación hay que añadirle algunas condiciones de contorno. No tiene condiciones inciales (como ocurre en los casos anteriores) porque la ecuación no depende del tiempo.

NOTA:

$$\frac{\partial^2 u}{\partial^2 x^2} + \frac{\partial^2 u}{\partial y^2} = \ldots \longrightarrow$$ Es lo mismo que $u_{xx} + u_{yy}$

Ecuación de Laplace (Poison) en un rectángulo

Dado un rectángulo del tipo R = [a, b] x [c, d] se quiere resolver el siguiente problema (el cual se conoce como problema de Dirichlet):

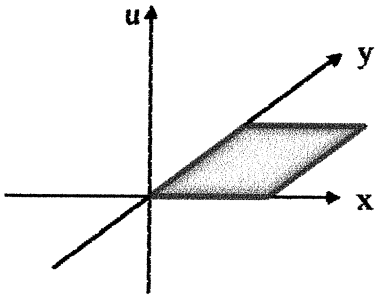
f es una fuerza externa y cuando es cero se dice que el problema es de Laplace, mientras que si es distinta de cero se dice que el problema es de Poison.

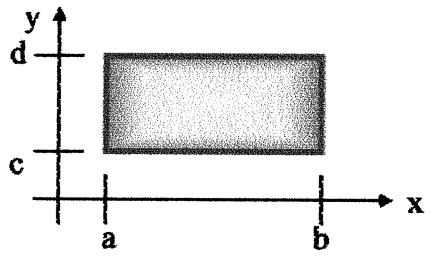
$$\begin{cases} \Delta u = u_{xx} + u_{yy} = f(x,y) & (x,y) \in R \\ u/_{\partial R} = g(x,y) & (x,y) \in \partial R \end{cases}$$

∂R es la frontera del rectángulo

Lo que se debe resolver en este problema es el valor de u en el interior del rectángulo.

De forma gráfica es lo siguiente:





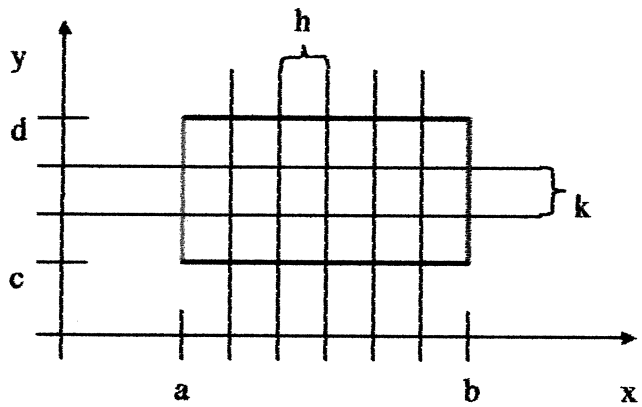
Los valores de u son conocidos.

Los valores de u son desconocidos.

Forma de resolver el problema:

Para resolver el problema es necesario realizar una malla al rectángulo para calcular en cada punto el valor de u.

$y = y_0, \dots, y_m$



$h = (b - a) / n$

$k = (d - c) / m$

$x = x_0, \dots, x_n$

Para resolver este problema hay que hacer uso de la fórmula de segundo orden para segundas derivadas.

$$f''(x_0) = \frac{f(x_0 + h) - 2f(x_0) + f(x_0 - h)}{h^2} + O(h^2)$$

Teniendo en cuenta esta fórmula:

$$u_{xx}(x_i, y_j) = \frac{u(x_i + h, y_j) - 2u(x_i, y_j) + u(x_i - h, y_j)}{h^2} + O(h^2)$$

$$u_{yy}(x_i, y_j) = \frac{u(x_i, y_j + k) - 2u(x_i, y_j) + u(x_i, y_j - k)}{k^2} + O(k^2)$$

Retomando el problema decir que lo que debemos calcular son los puntos interiores del rectángulo los cuales se definen como:

$u(x_i, y_j) \approx U_{ij}$ $\quad i = 0, \ldots, n \quad j = 0, \ldots, m$

Se tienen $(n - 1) * (m - 1)$ incógnitas.

Del problema se conoce:

$u(x_0, y_j) = g(x_0, y_j)$ _____

$u(x_n, y_j) = g(x_n, y_j)$ _____

$u(x_i, y_0) = g(x_i, y_0)$ _____

$u(x_i, y_0) = g(x_i, y_0)$ _____

Ahora se debe discretizar la ecuación $u(x_i, y_j)$ para lo cual se va a utilizar su aproximación U_{ij}, obteniéndose:

$$\frac{U_{i+1,j} - 2U_{i,j} + U_{i-1,j}}{h^2} + \frac{U_{i,j+1} - 2U_{i,j} + U_{i,j-1}}{k^2} = f(x_i, y_j)$$

Si se separan las incógnitas de los datos conocidos y se colocan en forma de matrices se obtiene algo que mantiene la forma siguiente:

$$\begin{pmatrix} \quad A \quad \end{pmatrix} * \begin{pmatrix} U \end{pmatrix} = \begin{pmatrix} b \end{pmatrix}$$

Con esto ya se puede calcular el resultado, que es $U = A \setminus b$

Vamos a ver el contenido de las matrices suponiendo que se tiene un ejemplo con seis incógnitas y que las incógnitas se enumeran de izquierda a derecha y de abajo a arriba:

MATRIZ A

$$\begin{pmatrix} -2/h^2-2/k^2 & 1/h^2 & 0 & 1/k^2 & 0 & 0 \\ 1/h^2 & -2/h^2-2/k^2 & 0 & 0 & 1/k^2 & 0 \\ 0 & 1/h^2 & -2/h^2-2/k^2 & 0 & 0 & 1/k^2 \\ 1/k^2 & 0 & 0 & -2/h^2-2/k^2 & 0 & 0 \\ 0 & 1/k^2 & 0 & 1/h^2 & -2/h^2-2/k^2 & 0 \\ 0 & 0 & 1/k^2 & 0 & 1/h^2 & -2/h^2-2/k^2 \end{pmatrix}$$

MATRIZ U

$$\begin{pmatrix} U_{11} \\ U_{21} \\ U_{31} \\ U_{12} \\ U_{22} \\ U_{32} \end{pmatrix}$$

MATRIZ b

$$\begin{pmatrix} f_{11}-U_{01}/h^2-U_{10}/k^2 \\ f_{21}-U_{20}/k^2 \\ f_{31}-U_{41}/h^2-U_{30}/k^2 \\ f_{12}-U_{02}/h^2-U_{13}/k^2 \\ f_{22}-U_{23}/k^2 \\ f_{32}-U_{42}/h^2-U_{33}/k^2 \end{pmatrix}$$

- Código general del método:

```
function laplace(a,b,c,d,n,m)
h=(b-a)/n;
k=(d-c)/m;
incog=((n-1)*(m-1));

matriz_a=zeros(incog);
cont=1;
cont2=1;
for i=1:incog
   matriz_a(i,i)=-2/h.^2-2/k.^2;
end
for j=n:incog
   matriz_a(j,j-(n-1))=1/k.^2;
   matriz_a(j-(n-1),j)=1/k.^2;
end
for r=2:incog
   resto=rem(cont,(n-1));
   if (resto~=0) matriz_a(r,r-1)=1/h.^2;
   end
   cont=cont+1;
   resto=rem(cont2,(n-1));
   if (resto~=0) matriz_a(r-1,r)=1/h.^2;
   end
   cont2=cont2+1;
end
```

rem calcula el resto de la división realizada entre los dos valores dados.

```
matriz_b=zeros(incog,1);
aux=1;
y=a;
for i=1:m-1
   y=y+k;
   x=a;
   for j=1:n-1
      x=x+h;
      matriz_b(aux)=der2x_der2y(x,y);
      aux=aux+1;
   end
end

x=a+h;
for i=1:n-1
   matriz_b(i)=matriz_b(i)-arista_abajo(x);
   x=x+h;
end
x=a+h;
for i=incog-(n-2):incog
   matriz_b(i)=matriz_b(i)-arista_arriba(x);
   x=x+h;
end

y=a+k;
i=1;
while (i<=incog)
   matriz_b(i)=matriz_b(i)-arista_izquierda(y);
   i=i+n-1;
   y=y+k;
end
y=a+k;
i=n-1;
while (i<=incog)
   matriz_b(i)=matriz_b(i)-arista_derecha(y);
   i=i+n-1;
   y=y+k;
end
matriz_u=matriz_a\matriz_b;

%dibujo
z=zeros(n-1,m-1);
mx=zeros(n-1,1);
my=zeros(m-1,1);
indice=1;
y=c+k;
for i=1:m-1
   my(i)=y;
   x=a+h;
   for j=1:n-1
      mx(j)=x;
      z(j,i)=matriz_u(indice);
      indice=indice+1;
      x=x+h;
   end
   y=y+k;
end

surfc(mx,my,z);
colormap(cool);
```

Dibuja la superficie resultante.

Como puede verse el método utilizada una serie de funciones que deben contener los datos que se conocen del problema.

- der2x_der2y: ➤ Es $f(x,y) = u_{xx} + u_{yy}$. Es una función conocida y representa una fuerza externa.

- arista_abajo: ➤ Es la función que define los valores conocidos de la base del rectángulo.

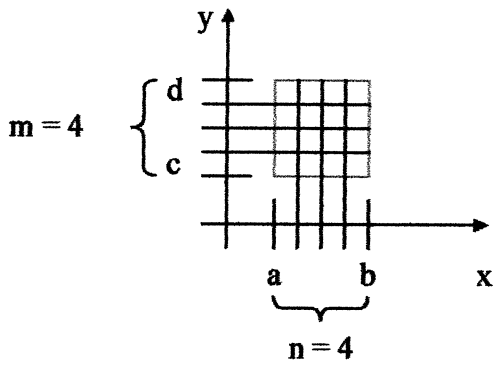
- arista_arriba: ➤ Representa los valores que se conocen de la parte alta del rectángulo.

- arista_izquierda: ➤ Esta función representa la arista izquierda del rectángulo conocido.

- arista_derecha; ➤ Es la función que define los valores que se conocen de la parte derecha del rectángulo.

EJERCICIOS:

1.- Con los siguientes datos:



$a = c = 0$
$b = d = 1$
arista abajo: $y = 0 \longrightarrow u(x,0) = x^2$
arista arriba: $y = 1 \longrightarrow u(x,1) = x^2 + 2x$
arista izquierda: $x = 0 \longrightarrow u(0,y) = 0$
arista derecha: $x = 1 \longrightarrow u(1,y) = 2y + 1$
$f(x,y) = u_{xx} + u_{yy} = 2 + 0 = 2$

a) Obtener lo que vale u en el interior.
b) Una vez obtenida la solución calcular el error sabiendo que la solución real es:

$$u(x,y) = x^2 + 2xy$$

Solución:

1.- Definir las aristas conocidas del rectángulo:

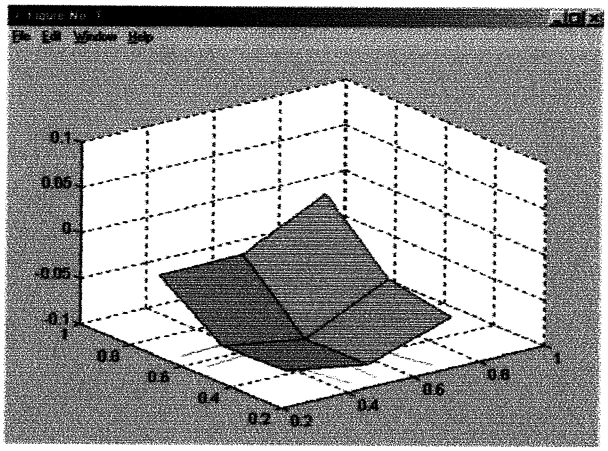
```
function y=arista_abajo(x)
y=x.^2;

function y=arista_arriba(x)
y=x.^2+2*x;

function y=arista_izquierda(x)
y=0;

function y=arista_derecha(x)
y=2*x+1;
```

2.- Definir la función f:

```
function z=der2x_der2y(x,y)
z=2;
```

3.- Ejecutar el método:

matriz_u = laplace(0,1,0,1,4,4)



4.- Anotaciones:

Las matrices que ha creado el método para generar este gráfico han sido:

MATRIZ A

$$\begin{pmatrix} -64 & 16 & 0 & 16 & 0 & 0 & 0 & 0 & 0 \\ 16 & -64 & 16 & 0 & 16 & 0 & 0 & 0 & 0 \\ 0 & 16 & -64 & 0 & 0 & 16 & 0 & 0 & 0 \\ 16 & 0 & 0 & -64 & 16 & 0 & 16 & 0 & 0 \\ 0 & 16 & 0 & 16 & -64 & 16 & 0 & 16 & 0 \\ 0 & 0 & 16 & 0 & 16 & -64 & 0 & 0 & 16 \\ 0 & 0 & 0 & 16 & 0 & 0 & -64 & 16 & 0 \\ 0 & 0 & 0 & 0 & 16 & 0 & 16 & -64 & 16 \\ 0 & 0 & 0 & 0 & 0 & 16 & 0 & 16 & -64 \end{pmatrix}$$

MATRIZ b

$$\begin{pmatrix} 1.9375 \\ 1.7500 \\ -0.0625 \\ 2.0000 \\ 2.0000 \\ 0 \\ 1.4375 \\ 0.7500 \\ -2.5625 \end{pmatrix}$$

MATRIZ U

$$\begin{pmatrix} -0.0688 \\ -0.0713 \\ -0.0220 \\ -0.0830 \\ -0.0850 \\ -0.0205 \\ -0.0532 \\ -0.0400 \\ 0.0249 \end{pmatrix}$$

Por lo que los nueve puntos que ha dibujado el método son:

$$\begin{pmatrix} -0.0688 & -0.0830 & -0.0532 \\ -0.0713 & -0.0850 & -0.0400 \\ -0.0220 & -0.0205 & 0.0249 \end{pmatrix}$$

5.- Cálculo del error:

Para ello es necesario crear una función que calcule la diferencia existente entre la solución real y la solución obtenida.

```
function error=error_laplace(a,b,c,d,n,m,matriz_u)
incog=((n-1)*(m-1));
h=(b-a)/n;
k=(d-c)/m;

real=zeros(incog,1);
indice=1;
y=c+k;
for i=1:m-1
   x=a+h;
   for j=1:n-1
      real(indice)=func_real(x,y);
      z2(j,i)=real(indice);
      indice=indice+1;
      x=x+h;
   end
   y=y+k;
end

error=0;
for i=1:incog
   nuevo_error=abs(matriz_u(i)-real(i));
   if (nuevo_error>error) error=nuevo_error;
   end
end
```

Además hay que definir la ecuación real:

```
function z=func_real(x,y)
z=x.^2+2*x*y;
```

Obtener el error:

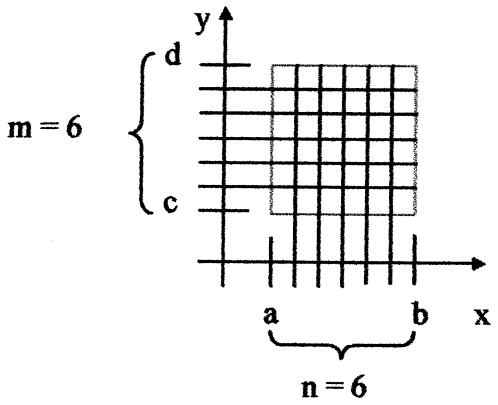
error_laplace(0,1,0,1,4,4,matriz_u)

A esta variable se le ha asignado el valor al hacer la llamada a Laplace (paso3)

1.6626

2.- Dados los siguientes datos:

arista abajo: $y = 0 \longrightarrow u(x,0) = x^3 + 2x$
arista arriba: $y = 2 \longrightarrow u(x,2) = x^3 + 2x - 4$
arista izquierda: $x = 0 \longrightarrow u(0,y) = -y^2$
arista derecha: $x = 2 \longrightarrow u(2,y) = 12 - y^2$
$f(x,y) = u_{xx} + u_{yy} = 6x - 2$



$a = c = 0$
$b = d = 2$

a) Obtener y dibujar u en el interior del rectángulo.
b) Comprobar el resultado con la ecuación $u(x,y) = x^3 + 2x - y^2$

Solución:

1.- Definir las aristas del rectángulo:

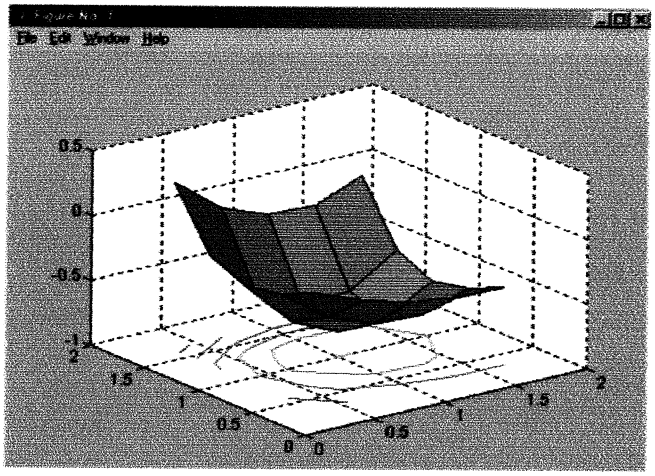
```
function y=arista_abajo(x)
y=x.^3+2*x;

function y=arista_arriba(x)
y=x.^3+2*x-4;

function y=arista_izquierda(x)
y=-x.^2;

function y=arista_derecha(x)
y=12-x.^2;
```

2.- Definir la función f:

```
function z=der2x_der2y(x,y)
z=6*x-2;
```

3.- Ejecutar el método:

matriz_u = laplace(0,2,0,2,6,6)



MATRIZ U

$$\begin{pmatrix} -0.1031 \\ -0.2308 \\ -0.2868 \\ -0.1623 \\ 0.2893 \\ -0.2475 \\ -0.4923 \\ -0.6429 \\ -0.5447 \\ 0.0024 \\ -0.3450 \\ -0.6259 \\ -0.8034 \\ -0.7094 \\ -0.1298 \\ -0.3956 \\ -0.6405 \\ -0.7911 \\ -0.6929 \\ -0.1457 \\ -0.3994 \\ -0.5271 \\ -0.5831 \\ -0.4586 \\ -0.0070 \end{pmatrix}$$

4.- Definir la ecuación final y cálcular el error:

```
function z=func_real(x,y)
z=x.^3+2*x-y.^2;
```

error_laplace(0,2,0,2,6,6,matriz_u) **7.5626**

3.- A partir de los siguientes datos:

$n = m = 14$
$a = 0$
$b = 3$
$c = 2$
$d = 3$
arista abajo: $y = 2 \longrightarrow u(x,2) = 2x + 8 - 3x^5$
arista arriba: $y = 3 \longrightarrow u(x,3) = 3x + 27 - 3x^5$
arista izquierda: $x = 0 \longrightarrow u(0,y) = y^3$
arista derecha: $x = 3 \longrightarrow u(3,y) = 3y + y^3 - 729$
$f(x,y) = u_{xx} + u_{yy} = 60x^3 - 6y$

a) Dibujar el valor de u en su interior:
b) Comprobar el resultado teniendo en cuenta que la solución real es:

$$u(x,y) = xy + y^3 - 3x^5$$

Solución:

1.- Definir las aristas del rectángulo:

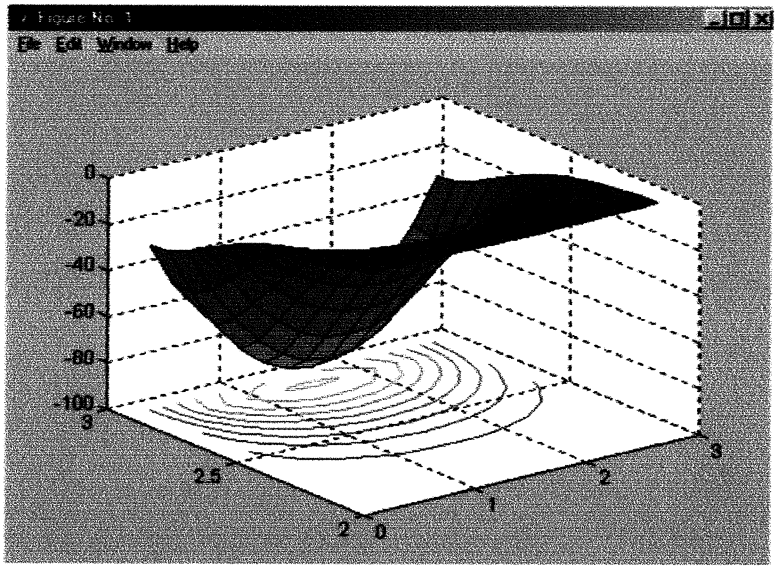
```
function y=arista_abajo(x)
y=2*x+8-3*x.^5;

function y=arista_arriba(x)
y=3*x+27-3*x.^5;

function y=arista_izquierda(x)
y=x.^3;

function y=arista_derecha(x)
y=3*x+x.^3-729;
```

2.- Definir la función f:

```
function z=der2x_der2y(x,y)
z=60*x.^3-6*y;
```

3.- Ejecutar el método:

matriz_u = laplace (0,3,2,3,14,14)



4.- Definir la ecuación final y cálcular el error:

```
function z=func_real(x,y)
z=x*y+y.^3-3*x.^5;
```

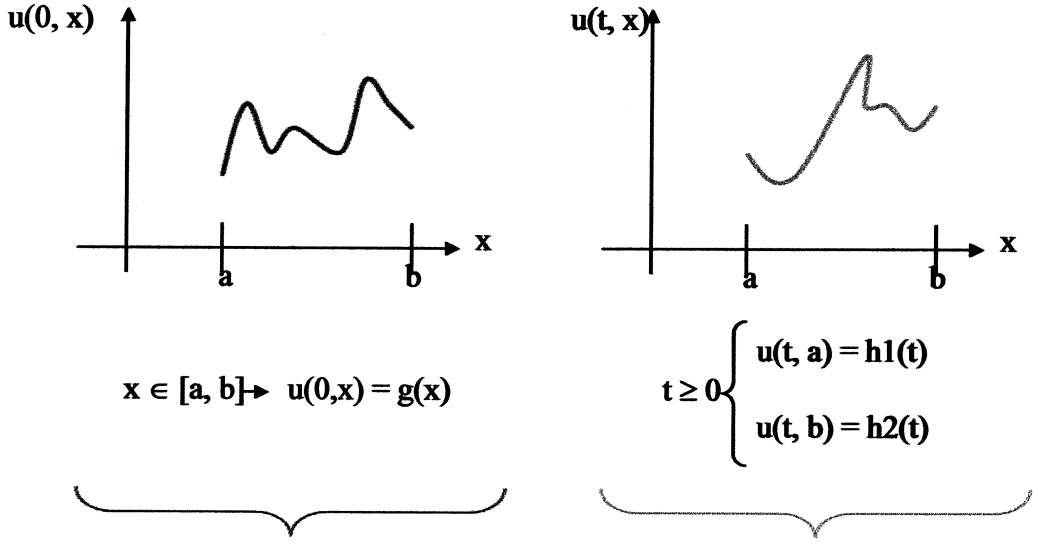
error_laplace(0,3,2,3,14,14,matriz_u) **460.1876**

Ecuación parabólica del calor o de difusión

La ecuación que hay que resolver en este caso es la siguiente:

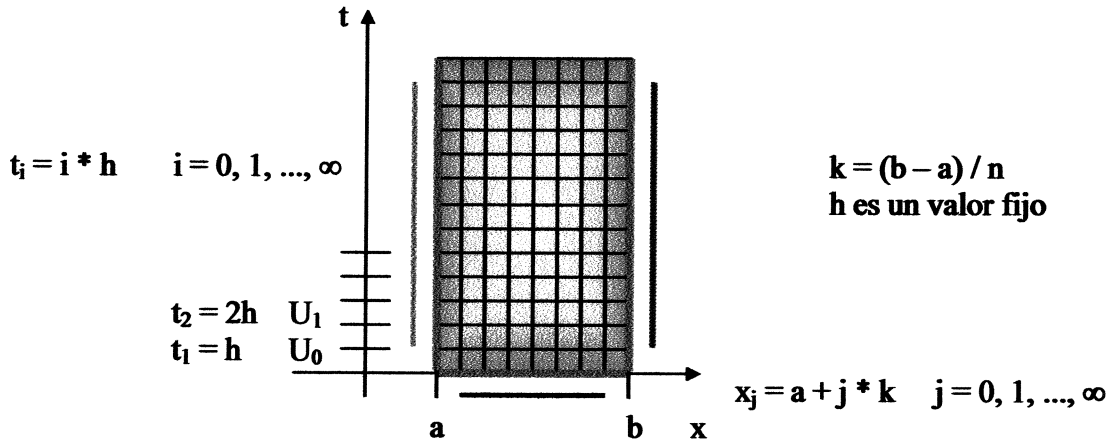
$$u_t = c^2 + u_{xx} + f(x,t)$$

En esta ocasión se trata de averiguar la temperatura de una barra en cualquier instante de tiempo. Inicialmente (t = 0) se conoce la temperatura de la barra para cualquier punto de esta (cualquier valor de x), pero en cualquier otro momento sólo se conoce la temperatura de la barra en los extremos de esta.



Forma de resolver el problema:

Partimos de un gráfico que muestra lo que vale cualquier punto de x a lo largo del tiempo:



Para resolver este problema hay que hacer uso de la fórmula de la derivada:

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0 - h)}{2h} + O(h^2)$$

Pero para que se pueda resolver el problema de forma fácil (sin utilizar la etapa de tiempo siguiente ya que no se conoce) es necesario plantear esta ecuación de la siguiente forma:

$$f'(x_0) = \frac{-f(x_0 - h) + f(x_0)}{h} + O(h)$$

Teniendo en cuenta esta fórmula se obtiene:

$$\frac{u(x;t) - u(x;t-h)}{h} + O(h) = c^2 * \frac{u(x+k;t) - 2u(x;t) + u(x-k;t)}{k^2} + O(k^2) + f(x,t)$$

Datos que se conocen a cerca del problema:

$f(x,t)$
$u(a, t) = h_1(t_i)$ _____
$u(b, t) = h_2(t_i)$ _____
$u(x, 0) = g(x_j)$ _____

Ahora se debe discretizar la ecuación $u(x_i, t_j)$ para lo cual se va a utilizar su aproximación U_{ij}, obteniéndose:

$$\frac{U_{i,j} - U_{i-1,j}}{h} = c^2 * \frac{U_{i,j+1} - 2U_{i,j} + U_{i,j-1}}{k^2} = f(x_i, t_j)$$

Para resolver el problema se van a clasificar las incógnitas en etapas de tal forma que:

$$W_j = \begin{pmatrix} U_{1,j} \\ U_{2,j} \\ \dots \\ U_{n-1,j} \end{pmatrix}$$

- Existen tres métodos para resolver este problema:

  - Método progresivo. Utiliza la fórmula:

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h} - h.f'(\mu)$$

  - Método regresivo. Utiliza la fórmula:

$$f'(x_0) = \frac{f(x_0) - f(x_0 - h)}{h} + h.f'(\overline{\mu})$$

  - Método de Crank Nicolson. Utiliza la fórmula:

$$f'(x_0) = \frac{f(x_0 + h) - 2f(x_0) + f(x_0 - h)}{h^2} + O(h^2)$$

Seguidamente se pasa a codificar cada uno de los métodos mencionados.

Ecuación parabólica del calor usando método progresivo

La ecuación que se debe resolver es la siguiente:

$$\frac{u(x_i;t_{j+1})-u(x_i;t_j)}{h}-k*u_{tt}(x_i,\mu_j)=c^2*\frac{u(x_{i+1};t_j)-2u(x_i;t_j)+u(x_{i-1};t_j)}{k^2}+O(k^2)+f(x_i,t_j)$$

Si llamamos λ al coeficiente de estabilidad, que es $c^2 * h / k^2$, nos queda:

$$U_{i,j+1} - U_{i,j} = \lambda[U_{i+1,j} - 2U_{i,j} + U_{i-1,j}] + h * f_{ij}$$

Despejando $U_{i,j+1}$ se obtiene:

$$U_{i,j+1} = \lambda U_{i-1,j} + (1 - 2\lambda)U_{i,j} + \lambda U_{i+1,j} + h * f_{ij}$$

El problema planteado de forma vectorial queda de la siguiente forma:

$$W_{j+1} = \begin{pmatrix} U_{1,j+1} \\ U_{2,j+1} \\ \dots \\ U_{n-1,j+1} \end{pmatrix} = \begin{pmatrix} & A & \end{pmatrix} * \begin{pmatrix} W_j \end{pmatrix} + \begin{pmatrix} b_j \end{pmatrix}$$

$$A$$

$$\begin{pmatrix} 1-2\lambda & \lambda & 0 & 0 & \dots & 0 & 0 & 0 \\ \lambda & 1-2\lambda & \lambda & 0 & \dots & 0 & 0 & 0 \\ 0 & \lambda & 1-2\lambda & \lambda & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & 0 & \lambda & 0 \\ 0 & 0 & 0 & 0 & \dots & \lambda & 1-2\lambda & \lambda \\ 0 & 0 & 0 & 0 & \dots & 0 & \lambda & 1-2\lambda \end{pmatrix}$$

$$W_j \qquad\qquad b_j$$

$$\begin{pmatrix} U_{1,j} \\ U_{2,j} \\ \dots \\ U_{n-1,j} \end{pmatrix} \qquad \begin{pmatrix} hf_{1j} - \lambda h_1(t_j) \\ hf_2 \\ \dots \\ hf_{n-1,j} - \lambda h_2(t_j) \end{pmatrix}$$

El resultado se obtiene a partir de la ecuación: $W_{j+1} = A * W_j + b_j$ cuyo error es del orden $O(k + h^2)$.

Este método es bastante malo y se le llama condicionalmente estable. Para que el método sea estable λ debe ser menor que ½, de tal forma que si se utiliza c = 1, entonces $h/k^2 < ½$ o lo que es lo mismo $h < k^2/2$. Esto indica que para que el método sea estable, h debe ser mucho menor que k, por lo que es necesario trabajar con dos escalas distintas, lo cual no es conveniente.

Código general del método:

```
function matriz_a=progresivo(a,b,n,h,c,tf)
k=(b-a)/n;   %n -- n° particiones de x.     k -- distancia de x a x
landa=c.^2*h/k.^2;   %h --  distancia entre las etapas
num_etap=tf/h;  %tf -- tiempo final, de la última etapa
incog=(n-1);
matriz_a=zeros(incog);
for i=1:incog
   matriz_a(i,i)=1-2*landa;
end
for r=2:incog
   matriz_a(r,r-1)=landa;
   matriz_a(r-1,r)=landa;
end
pos_x=a+k;  %pos_x -- valor de x que se utiliza
ind=1;   %ind -- indice de x
while (ind<=incog)
   w1(ind)=base(pos_x);
   ind=ind+1;
   pos_x=pos_x+k;
end
for etapa=1:num_etap   %etapa -- indice para las etapas
   pos_x=a+k; ind=1;
   while  (ind<=incog)
      w_real(ind)=func(pos_x,etapa*h);
      if (ind==1)
         b1(ind)=h*der_f(pos_x,etapa*h-h)-landa*izquierda(etapa*h-h);
      else
         if (ind==(n-1))
            b1(ind)=h*der_f(pos_x,etapa*h-h)-landa*derecha(etapa*h-h);
         else
            b1(ind)=h*der_f(pos_x,etapa*h-h);
         end
      end
      pos_x=pos_x+k;
      ind=ind+1;
   end
   invb=b1';
   invw=w1';
   w2=matriz_a*invw+invb;
   w_real
   w1=w2'
   error=norm(w_real-w1)
end
```

error calcula la diferencia entre la solución real (w_real) y la solución obtenida por el método (w1)

Como puede observarse, el método utiliza otras funciones que proporcionan los datos que se conocen del problema. Estas funciones son:

der_f	f(x,t)
base	g(x)
izquierda	u(a,t)
derecha	u(b,t)

También utiliza la función func, que contiene la función original para comparar la solución que proporciona el método con la solución real.

EJERCICIOS:

1.- Resolver el problema para t = 0.002, t = 0.004, t = 0.006, t = 0.008 y t = 0.010 conociendo los siguientes datos:

$a = 0$
$b = 0.5$
$n = 5$
$c = 1$
$u(x,0) = 2x$
$u(0,t) = 0$
$u(1,t) = t^2 + 1.25$
$f(x,t) = 2tx^2 - 2t$

Comprobar el resultado sabiendo que la solución exacta se rige por la ecuación:

$$u(x,t) = t^2x^2 + 2x$$

Solución:

1.- Definir las funciones auxiliares con los datos que se conocen del problema:

```
function y=der_f(x,t)
y=2*t*x.^2-2*t;

function y=base(x)
y=2*x;

function y=izquierda(t)
y=0;

function y=derecha(t)
y=t.^2+1.25;
```

2.- Definir la función original:

```
function y=func(x,t)
y=t.^2*x.^2+2*x;
```

3.- Calcular el resultado:

$k = (0.5 - 0) / 5 = 0.1$
$h = t_1 - t_2 = 0.002$ Como h es menor que k, h es válido

progresivo(0,0.5,5,0.002,1,0.010)

w_real = 0.2000 0.4000 0.6000 0.8000
w1 = 0.2000 0.4000 0.6000 0.3500
error = 0.4500

w_real = 0.2000 0.4000 0.6000 0.8000
w1 = 0.2000 0.4000 0.5100 0.0800
error = 0.7256

w_real = 0.2000 0.4000 0.6000 0.8000
w1 = 0.2000 0.3820 0.4020 -0.1000
error = 0.9217

w_real = 0.2000 0.4000 0.6000 0.8000
w1 = 0.1964 0.3496 0.2976 -0.2296
error = 1.0743

w_real = 0.2000 0.4000 0.6000 0.8000
w1 = 0.1878 0.3086 0.2026 -0.3282
error = 1.1998

matriz_a =

$$\begin{pmatrix} 0.6000 & 0.2000 & 0 & 0 & 0 \\ 0.2000 & 0.6000 & 0.2000 & 0 & 0 \\ 0 & 0.2000 & 0.6000 & 0.2000 & 0 \\ 0 & 0 & 0.2000 & 0.6000 & 0.2000 \\ 0 & 0 & 0 & 0.2000 & 0.6000 \end{pmatrix}$$

4.- Lectura del resultado:

Como k = 0.1, por lo que la diferencia entre cada x es 0.1.

	Método				Real			
	x = 0.1	x = 0.2	x = 0.3	x = 0.4	x = 0.1	x = 0.2	x = 0.3	x = 0.4
t = 0.002	0.2000	0.4000	0.6000	0.3500	0.2000	0.4000	0.6000	0.8000
Error	0.4500							
t = 0.004	0.2000	0.4000	0.5100	0.0800	0.2000	0.4000	0.6000	0.8000
Error	0.7256							
t = 0.006	0.2000	0.3820	0.4020	-0.1000	0.2000	0.4000	0.6000	0.8000
Error	0.9217							
t = 0.008	0.1964	0.3496	0.2976	-0.2296	0.2000	0.4000	0.6000	0.8000
Error	1.0743							
t = 0.010	0.1878	0.3086	0.2026	-0.3282	0.2000	0.4000	0.6000	0.8000
Error	1.1998							

2.- Obtener u(x, t) en t = 0.005 conociendo los siguientes datos:

$a = 0$
$b = 1$
$n = 3$
$h = 0.0005$
$c = 0.25$
$u(x,0) = 2x^2$
$u(0,t) = -t$
$u(1,t) = 2$
$f(x,t) = x - 3$

Comprobar el resultado sabiendo que la solución exacta se rige por la ecuación:

$$u(x,t) = 2x^2 + tx - t$$

Solución:

1.- Definir las funciones auxiliares con los datos que se conocen del problema:

```
function y=der_f(x,t)
y=x-3;

function y=base(x)
y=2*x.^2;

function y=izquierda(t)
y=-t;

function y=derecha(t)
y=2;
```

2.- Definir la función original:

```
function y=func(x,t)
y=2*x.^2+t*x-t;
```

3.- Calcular el resultado:

progresivo(0,1,3,0.0005,0.25,0.005)

matriz_a =

$$\begin{pmatrix} 0.9994 & 0.0003 \\ 0.0003 & 0.9994 \end{pmatrix}$$

La solución final es:

w1 = 0.2101 0.8673 → x1 = 0.3333, x2 = 0.6666
error = 0.6524

No muestro todas las etapas porque son demasiadas.

Ecuación parabólica del calor usando método regresivo

Este método surge para solucionar el problema de la inestabilidad del método anterior.

Ecuación a resolver:

$$\frac{u(x_i;t_{j+1})-u(x_i;t_j)}{h}+k*u_{tt}(x_i,\bar{\mu}_j)=c^2*\frac{u(x_{i+1};t_{j+1})-2u(x_i;t_{j+1})+u(x_{i-1};t_{j+1})}{k^2}+O(k^2)+f(x_i,t_{j+1})$$

λ, como en el caso anterior, equivale a: $c^2 * h / k^2$.

Discretizando se obtiene:

$$U_{i,j+1} - U_{i,j} = \lambda[U_{i+1,j+1} - 2U_{i,j+1} + U_{i-1,j+1}] + h * f_{ij+1}$$

El problema planteado de forma vectorial queda de la siguiente forma:

$$\begin{pmatrix} A \end{pmatrix} * \begin{pmatrix} W_{j+1} \end{pmatrix} = \begin{pmatrix} W_j \end{pmatrix} + \begin{pmatrix} b_j \end{pmatrix}$$

A

$$\begin{pmatrix} 1+2\lambda & -\lambda & 0 & 0 & \dots & 0 & 0 & 0 \\ -\lambda & 1+2\lambda & -\lambda & 0 & \dots & 0 & 0 & 0 \\ 0 & -\lambda & 1+2\lambda & -\lambda & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & 0 & -\lambda & 0 \\ 0 & 0 & 0 & 0 & \dots & -\lambda & 1+2\lambda & -\lambda \\ 0 & 0 & 0 & 0 & \dots & 0 & -\lambda & 1+2\lambda \end{pmatrix}$$

W_{j+1}

$$\begin{pmatrix} U_{1,j+1} \\ U_{2,j+1} \\ \dots \\ U_{n-1,j+1} \end{pmatrix}$$

W_j

$$\begin{pmatrix} U_{1,j} \\ U_{2,j} \\ \dots \\ U_{n-1,j} \end{pmatrix}$$

b_j

$$\begin{pmatrix} hf_{1j+1} - \lambda h_1(t_{j+1}) \\ hf_{2,j+1} \\ \dots \\ hf_{n-1,j+1} - \lambda h_2(t_{j+1}) \end{pmatrix}$$

El resultado se obtiene a partir de la ecuación: $W_{j+1} = A\backslash(W_j + b_j)$ y el error es del orden $O(h + k^2)$.

Este método tiene como inconveniente que necesita más cálculos que el método anterior y es más complejo, pero tiene como ventaja que es estable.

Código del método:

```
function matriz_a=regresivo(a,b,n,h,c,  ,tf)
k=(b-a)/n;  %n -- n° particiones de x.     k -- distancia de x a x
landa=c.^2*h/K.^2;   %h --  distancia entre las etapas
num_etap=tf/h; %tf -- tiempo final, de la última etapa
incog=(n-1);
matriz_a=zeros(incog);
for i=1:incog
   matriz_a(i,i)=1+2*landa;
end                                          matriz A
for r=2:incog
   matriz_a(r,r-1)=-landa;
   matriz_a(r-1,r)=-landa;
end
pos_x=a+k; ind=1;     %pos_x -- valor de x que se utiliza
while (ind<=incog)
   w1(ind)=base(pos_x);                      w inicial
   ind=ind+1;
   pos_x=pos_x+k;
end
for etapa=1:num_etap %etapa -- indice para las etapas
   pos_x=a+k; ind=1;
   while  (ind<=incog)
      w_real(ind)=func(pos_x,etapa*h);
      if (ind==1) %si es el primer x
         b1(ind)=h*der_f(pos_x,etapa*h)+landa*izquierda(etapa*h);
      else
         if (ind==(n-1)) %si es el último x
            b1(ind)=h*der_f(pos_x,etapa*h)+landa*derecha(etapa*h);
         else
            b1(ind)=h*der_f(pos_x,etapa*h);
         end
      end
      pos_x=pos_x+k;
      ind=ind+1;
   end
   invb=b1';
   invw=w1';
   w2=matriz_a\(invw+invb);
   w_real
   w1=w2'
   error=norm(w_real-w1)
end
```

error calcula la diferencia entre la solución real (w_real) y la solución obtenida por el método (w1)

Como puede observarse, el método utiliza otras funciones que proporcionan los datos que se conocen del problema. Estas funciones son:

der_f	f(x,t)
base	g(x)
izquierda	u(a,t)
derecha	u(b,t)

También utiliza la función func, que contiene la función original para poder obtener el error.

EJERCICIOS:

1.- Resolver el problema para t = 0.002, t = 0.004, t = 0.006, t = 0.008 y t = 0.010 conociendo los siguientes datos:

$a = 0$
$b = 0.5$
$n = 5$
$h = 0.002$
$c = 1$
$u(x,0) = 2x$
$u(0,t) = 0$
$u(1,t) = t^2 + 1.25$
$f(x,t) = 2tx^2 - 2t$

Comprobar el resultado sabiendo que la solución exacta se rige por la ecuación:

$$u(x,t) = t^2x^2 + 2x$$

Solución:

1.- Definir las funciones auxiliares con los datos que se conocen del problema:

```
function y=der_f(x,t)
y=2*t*x.^2-2*t;

function y=base(x)
y=2*x;

function y=izquierda(t)
y=0;

function y=derecha(t)
y=t.^2+1.25;
```

2.- Definir la función original:

```
function y=func(x,t)
y=t.^2*x.^2+2*x;
```

3.- Calcular el resultado:

```
regresivo (0,0.5,5,0.002,1,0.010)
```

w_real = 0.2000 0.4000 0.6000 0.8000
w1 = 0.2001 0.4008 0.6053 0.8365
error = 0.0369

w_real = 0.2000 0.4000 0.6000 0.8000
w1 = 0.2004 0.4025 0.6133 0.8637
error = 0.0651

w_real = 0.2000 0.4000 0.6000 0.8000
w1 = 0.2010 0.4051 0.6223 0.8844
error = 0.0874

w_real = 0.2000 0.4000 0.6000 0.8000
w1 = 0.2020 0.4084 0.6315 0.9005
error = 0.1056

w_real = 0.2000 0.4000 0.6000 0.8000
w1 = 0.2031 0.4122 0.6404 0.9133
error = 0.1209

matriz_a =

$$\begin{pmatrix} 1.4000 & -0.2000 & 0 & 0 & 0 \\ -0.2000 & 1.4000 & -0.2000 & 0 & 0 \\ 0 & -0.2000 & 1.4000 & -0.2000 & 0 \\ 0 & 0 & -0.2000 & 1.4000 & -0.2000 \\ 0 & 0 & 0 & -0.2000 & 1.4000 \end{pmatrix}$$

4.- Lectura del resultado:

Como k = 0.1, por lo que la diferencia entre cada x es 0.1.

	Método				Real			
	x = 0.1	x = 0.2	x = 0.3	x = 0.4	x = 0.1	x = 0.2	x = 0.3	x = 0.4
t = 0.002	0.2001	0.4008	0.6053	0.8365	0.2000	0.4000	0.6000	0.8000
Error	0.0369							
t = 0.004	0.2004	0.4025	0.6133	0.8637	0.2000	0.4000	0.6000	0.8000
Error	0.0651							
t = 0.006	0.2010	0.4051	0.6223	0.8844	0.2000	0.4000	0.6000	0.8000
Error	0.0874							
t = 0.008	0.2020	0.4084	0.6315	0.9005	0.2000	0.4000	0.6000	0.8000
Error	0.1056							
t = 0.010	0.2031	0.4122	0.6404	0.9133	0.2000	0.4000	0.6000	0.8000
Error	0.1209							

Con este ejemplo se puede comprobar que este método es mejor que el método progresivo ya que para cada una de las etapas se observa que el error es menor.

2.- Obtener el resultado en t = 0.05 conociendo los siguientes datos:

$a = 0$
$b = 1$
$n = 5$
$c = 1$
$u(x,0) = -x^3$
$u(0,t) = t^3$
$u(1,t) = t^3 + 2t - 1$
$f(x,t) = 3t^2 + 2x + 6x$

Comprobar el resultado sabiendo que la solución exacta se rige por la ecuación:

$$u(x,t) = t^3 + 2tx - x^3$$

Solución:

1.- Decidir el valor de h.

Como únicamente es necesario obtener el resultado en t = 0.05 se podría usar un h = 0.05, pero es mejor usar un h más pequeño para que el método realice etapas intermedias y así obtener un error más pequeño. Voy a usar un h = 0.01

2.- Definir las funciones auxiliares con los datos que se conocen del problema:

```
function y=der_f(x,t)
y=3*t.^2+2*x+6*x;

function y=base(x)
y=-x.^3;

function y=izquierda(t)
y=t.^3;

function y=derecha(t)
y=t.^3+2*t-1;
```

3.- Definir la función original:

```
function y=func(x,t)
y=t.^3+2*t*x-x.^3;
```

4.- Calcular el resultado:

regresivo(0,1,5,0.01,1,0.05)

w_real = -0.0040 -0.0560 -0.2040 -0.4960
w1 = -0.0040 -0.0560 -0.2040 -0.4960
error = 3.5968e-006

w_real = 0.0000 -0.0480 -0.1920 -0.4800
w1 = 0.0000 -0.0480 -0.1920 -0.4800
error = 1.2250e-005

w_real = 0.0040 -0.0400 -0.1800 -0.4640
w1 = 0.0040 -0.0400 -0.1800 -0.4640
error = 2.5495e-005

w_real = 0.0081 -0.0319 -0.1679 -0.4479
w1 = 0.0081 -0.0319 -0.1679 -0.4479
error = 4.2922e-005

w_real = 0.0121 -0.0239 -0.1559 -0.4319
w1 = 0.0122 -0.0238 -0.1558 -0.4318
error = 6.4161e-005

matriz_a =

$$\begin{pmatrix} 1500 & -0.2500 & 0 & 0 \\ -0.2500 & 1500 & -0.2500 & 0 \\ 0 & -0.2500 & 1500 & -0.2500 \\ 0 & 0 & -0.2500 & 1500 \end{pmatrix}$$

5.- Lectura del resultado:

Como a = 0, b = 1, n = 5 y k = (b – a) / n k = 0.2, por lo que la diferencia entre cada x es 0.2.

	Método				Real			
	x = 0.2	x = 0.4	x = 0.6	x = 0.8	x = 0.2	x = 0.4	x = 0.6	x = 0.8
t = 0.01	-0.0040	-0.0560	-0.2040	-4960	0.0040	-0.0560	-0.2040	-0.4960
Error	0.0000035968							
t = 0.02	0.0000	-0.0480	-0.1920	-0.4800	0.0000	-0.0480	-0.1920	-0.4800
Error	0.000012250							
t = 0.03	0.0040	-0.0400	-0.1800	-0.4640	0.0040	-0.0400	-0.1800	-0.4640
Error	0.000025495							
t = 0.04	0.0081	-0.0319	-0.1679	0.4479	0.0081	-0.0319	-0.1679	0.4479
Error	0.000042922							
t = 0.05	0.0122	-0.0238	-0.1558	-0.4318	0.0121	-0.0239	-0.1559	-0.4319
Error	0.000064161							

Ecuación parabólica del calor usando el método de Crank Nicolson

Este método intenta mejorar el funcionamiento de los dos métodos anteriores, utilizando lo bueno de cada uno de ellos; para conseguirlo el método utiliza la suma de las ecuaciones de los mismos.

$$\begin{aligned} 2/h * (U_{i,j+1} - U_{i,j}) &= \lambda/2 * [U_{i+1,j} - 2U_{i,j} + U_{i-1,j}] + \lambda/2 * [U_{i+1,j+1} - 2U_{i,j+1} + U_{i-1,j+1}] + \\ &+ h/2 * [f_{i,j} + f_{i,j+1}] \quad = \\ &= -\lambda/2 * U_{i-1,j+1} + (1 + \lambda) * U_{i,j+1} - \lambda/2 * U_{i+1,j+1} + \lambda/2 * U_{i-1,j} + \\ &(1 + \lambda) * U_{i,j} + \lambda/2 * U_{i+1,j} + h/2 * [f_{i,j} + f_{i,j+1}] \end{aligned}$$

Para poder resolverlo se va a descomponer en vectores, al igual que en los métodos anteriores. Los vectores y matrices necesarios son los siguientes:

$$\begin{pmatrix} A \end{pmatrix} * \begin{pmatrix} W_{j+1} \end{pmatrix} = \begin{pmatrix} B \end{pmatrix} * \begin{pmatrix} W_j \end{pmatrix} + \begin{pmatrix} b_j \end{pmatrix}$$

A

$$\begin{pmatrix} 1+\lambda & -\lambda/2 & 0 & 0 & \dots & 0 & 0 & 0 \\ -\lambda/2 & 1+\lambda & -\lambda/2 & 0 & \dots & 0 & 0 & 0 \\ 0 & -\lambda/2 & 1+\lambda & -\lambda/2 & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & 0 & -\lambda/2 & 0 \\ 0 & 0 & 0 & 0 & \dots & -\lambda/2 & 1+\lambda & -\lambda/2 \\ 0 & 0 & 0 & 0 & \dots & 0 & -\lambda/2 & 1+\lambda \end{pmatrix}$$

B

$$\begin{pmatrix} 1-\lambda & \lambda/2 & 0 & 0 & \dots & 0 & 0 & 0 \\ \lambda/2 & 1-\lambda & \lambda/2 & 0 & \dots & 0 & 0 & 0 \\ 0 & \lambda/2 & 1-\lambda & \lambda/2 & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & 0 & \lambda/2 & 0 \\ 0 & 0 & 0 & 0 & \dots & \lambda/2 & 1-\lambda & \lambda/2 \\ 0 & 0 & 0 & 0 & \dots & 0 & \lambda/2 & 1-\lambda \end{pmatrix}$$

W_{j+1}

$$\begin{pmatrix} U_{1,j+1} \\ U_{2,j+1} \\ \dots \\ U_{n-1,j+1} \end{pmatrix}$$

W_j

$$\begin{pmatrix} U_{1,j} \\ U_{2,j} \\ \dots \\ U_{n-1,j} \end{pmatrix}$$

b_j

$$\begin{pmatrix} h/2 * [f_{1,j} + f_{1,j+1}] + \lambda/2 * [h_1(t_{j+1}) + h_1(t_j)] \\ h/2 * [f_{2,j} + f_{2,j+1}] \\ \dots \\ h/2 * [f_{n-1,j} + f_{n-1,j+1}] + \lambda/2 * [h_2(t_{j+1}) + h_2(t_j)] \end{pmatrix}$$

El resultado se obtiene a partir de la ecuación: $W_{j+1} = A\backslash(BW_j + b_j)$ y el error es del orden $O(h^2 + k^2)$.

Este método es también un método estable (al igual que el método anterior), pero tiene como ventaja que permite utilizar la misma escala (de tiempo y de posición).

Código del método:

```
function nicolson(a,b,n,h,c,  ,tf)
k=(b-a)/n;   %n -- n° particiones de x.    k -- distancia de x a x
landa=c.^2*h/k.^2;   %h --  distancia entre las etapas
num_etap=tf/h;  %tf -- tiempo final, de la última etapa
incog=(n-1);

matriz_a=zeros(incog);
for i=1:incog
   matriz_a(i,i)=1+landa;
end
for r=2:incog                          matriz A
   matriz_a(r,r-1)=-landa/2;
   matriz_a(r-1,r)=-landa/2;
end
matriz_b=zeros(incog);
for i=1:incog
   matriz_b(i,i)=1-landa;
end                                    matriz B
for r=2:incog
   matriz_b(r,r-1)=landa/2;
   matriz_b(r-1,r)=landa/2;
end
pos_x=a+k;  %pos_x -- valor de x que se utiliza
ind=1;   %ind -- indice de x
while (ind<=incog)
   w1(ind)=base(pos_x);                w inicial
   ind=ind+1;
   pos_x=pos_x+k;
end
for etapa=1:num_etap   %etapa -- indice para las etapas
   pos_x=a+k; ind=1;
   while  (ind<=incog)
      w_real(ind)=func(pos_x,etapa*h);
      if (ind==1)
         b1(ind)=h/2*(der_f(pos_x,etapa*h-h)+ der_f(pos_x,etapa*h))+
                 landa/2*izquierda(etapa*h)+landa*izquierda(etapa*h-h);
      else
         if (ind==(n-1))
            b1(ind)=h/2*(der_f(pos_x,etapa*h-h)+der_f(pos_x,etapa*h))+
                  landa/2*derecha(etapa*h)+landa/2*derecha(etapa*h-h);
         else
            b1(ind)=h/2*(der_f(pos_x,etapa*h-h)+der_f(pos_x,etapa*h));
         end
      end
      pos_x=pos_x+k;
      ind=ind+1;
   end
```

```
    invb=b1';
    invw=w1';
    w2=matriz_a\(matriz_b*invw+invb);
    w_real
    w1=w2'
    error=norm(w_real-w1)
end
matriz_a, matriz_b
```

error calcula la diferencia entre la solución real (w_real) y la solución obtenida por el método (w1)

El método utiliza las mismas funciones que utilizan los dos métodos anteriores. Estas funciones son:

der_f	→	f(x,t)
base	→	g(x)
izquierda	→	u(a,t)
derecha	→	u(b,t)
func	→	función original

EJERCICIOS:

1.- Resolver el problema para t = 0.002, t = 0.004, t = 0.006, t = 0.008 y t = 0.010 conociendo los siguientes datos:

$a = 0$
$b = 0.5$
$n = 5$
$h = 0.002$
$c = 1$
$u(x,0) = 2x$
$u(0,t) = 0$
$u(1,t) = t^2 + 1.25$
$f(x,t) = 2tx^2 - 2t$

Comprobar el resultado sabiendo que la solución exacta se rige por la ecuación:

$$u(x,t) = t^2x^2 + 2x$$

Solución:

1.- Definir las funciones auxiliares con los datos que se conocen del problema:

```
function y=der_f(x,t)
y=2*t*x.^2-2*t;

function y=base(x)
y=2*x;

function y=izquierda(t)
y=0;

function y=derecha(t)
y=t.^2+1.25;
```

2.- Definir la función original:

```
function y=func(x,t)
y=t.^2*x.^2+2*x;
```

3.- Calcular el resultado:

nicolson (0,0.5,5,0.002,1,0.010)

w_real = 0.2000 0.4000 0.6000 0.8000
w1 = 0.2000 0.4003 0.6035 0.8420
error = 0.0421

w_real = 0.2000 0.4000 0.6000 0.8000
w1 = 0.2002 0.4015 0.6119 0.8709
error = 0.0719

w_real = 0.2000 0.4000 0.6000 0.8000
w1 = 0.2006 0.4039 0.6219 0.8918
error = 0.0944

w_real = 0.2000 0.4000 0.6000 0.8000
w1 = 0.2013 0.4073 0.6322 0.9074
error = 0.1123

w_real = 0.2000 0.4000 0.6000 0.8000
w1 = 0.2024 0.4113 0.6419 0.9194
error = 0.1271

matriz_a =

$$\begin{pmatrix} 1.2000 & -0.1000 & 0 & 0 & 0 \\ -0.1000 & 1.2000 & -0.1000 & 0 & 0 \\ 0 & -0.1000 & 1.2000 & -0.1000 & 0 \\ 0 & 0 & -0.1000 & 1.2000 & -0.1000 \\ 0 & 0 & 0 & -0.1000 & 1.2000 \end{pmatrix}$$

matriz_b =

$$\begin{pmatrix} 0.8000 & 0.1000 & 0 & 0 & 0 \\ 0.1000 & 0.8000 & 0.1000 & 0 & 0 \\ 0 & 0.1000 & 0.8000 & 0.1000 & 0 \\ 0 & 0 & 0.1000 & 0.8000 & 0.1000 \\ 0 & 0 & 0 & 0.1000 & 0.8000 \end{pmatrix}$$

4.- Lectura del resultado:

Como k = 0.1, por lo que la diferencia entre cada x es 0.1.

	Método				Real			
	x = 0.1	x = 0.2	x = 0.3	x = 0.4	x = 0.1	x = 0.2	x = 0.3	x = 0.4
t = 0.002	0.2000	0.4003	0.6035	0.8420	0.2000	0.4000	0.6000	0.8000
Error	0.0421							
t = 0.004	0.2002	0.4015	0.6119	0.8709	0.2000	0.4000	0.6000	0.8000
Error	0.0719							
t = 0.006	0.2006	0.4039	0.6219	0.8918	0.2000	0.4000	0.6000	0.8000
Error	0.0944							
t = 0.008	0.2013	0.4073	0.6322	0.9074	0.2000	0.4000	0.6000	0.8000
Error	0.1123							
t = 0.010	0.2024	0.4113	0.6419	0.9194	0.2000	0.4000	0.6000	0.8000
Error	0.1271							

En este caso el error es muy parecido al del método anterior (esto se debe a que ambos son estables), la diferencia entre ambos radica en que en este método no hay que restringir h a un valor mucho menor que k.

2.- Resolver el problema para t = 0.5 conociendo los siguientes datos:

$a = 0$
$b = 0.5$
$n = 5$
$h = 0.1$
$c = 1$
$u(x,0) = x^3 + 2x$
$u(0,t) = -t^2$
$u(0.5,t) = 3 - t^2$
$f(x,t) = -2t - 6x$

Mostrar el resultado para cada una de las etapas que genera el método y comprobarlo sabiendo que la solución exacta es:

$$u(x,t) = x^3 + 2x - t^2$$

Solución:

1.- Definir las funciones auxiliares con los datos que se conocen del problema:

```
function y=der_f(x,t)
y=-2*t-6*x;

function y=base(x)
y=x.^3+2*x;

function y=izquierda(t)
y=-t.^2;

function y=derecha(t)
y=1.125-t.^2;
```

2.- Definir la función original:

```
function y=func(x,t)
y=x.^3+2*x-t.^2;
```

3.- Calcular el resultado:

nicolson(0,0.5,5,0.1,1,0,0.5)

w_real = 0.1910 0.3980 0.6170 0.8540
w1 = 0.1910 0.3980 0.6170 0.8540
error = 4.2367e-005

w_real = 0.1610 0.3680 0.5870 0.8240
w1 = 0.1547 0.3641 0.5848 0.8230
error = 0.0078

w_real = 0.1110 0.3180 0.5370 0.7740
w1 = 0.0896 0.3035 0.5281 0.7698
error = 0.0276

w_real = 0.0410 0.2480 0.4670 0.7040
w1 = -0.0030 0.2172 0.4475 0.6945
error = 0.0579

w_real = -0.0490 0.1580 0.3770 0.6140
w1 = -0.1239 0.1048 0.3429 0.5974
error = 0.0994

matriz_a =

$$\begin{pmatrix} 11.0000 & -5.0000 & 0 & 0 & 0 \\ -5.0000 & 11.0000 & -5.0000 & 0 & 0 \\ 0 & -5.0000 & 11.0000 & -5.0000 & 0 \\ 0 & 0 & -5.0000 & 11.0000 & -5.0000 \\ 0 & 0 & 0 & -5.0000 & 11.0000 \end{pmatrix}$$

matriz_b =

$$\begin{pmatrix} -9.0000 & 5.0000 & 0 & 0 & 0 \\ 5.0000 & -9.0000 & 5.0000 & 0 & 0 \\ 0 & 5.0000 & -9.0000 & 5.0000 & 0 \\ 0 & 0 & 5.0000 & -9.0000 & 5.0000 \\ 0 & 0 & 0 & 5.0000 & -9.0000 \end{pmatrix}$$

4.- Lectura del resultado:

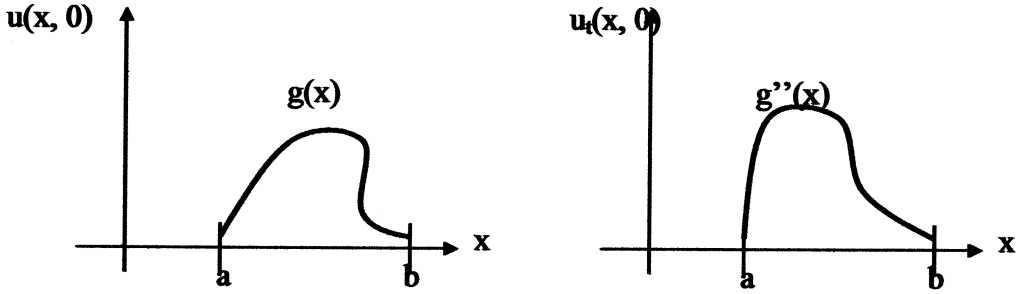
	Método				Real			
	x = 0.1	x = 0.2	x = 0.3	x = 0.4	x = 0.1	x = 0.2	x = 0.3	x = 0.4
t = 0.1	0.1910	0.3980	0.6170	0.8540	0.1910	0.3980	0.6170	0.8540
Error	0.000042367							
t = 0.2	0.1547	0.3641	0.5848	0.8230	0.1610	0.3680	0.5870	0.8240
Error	0.0078							
t = 0.3	0.0896	0.3035	0.5281	0.7698	0.1110	0.3180	0.5370	0.7740
Error	0.0276							
t = 0.4	-0.0030	0.2172	0.4475	0.6945	0.0410	0.2480	0.4670	0.7040
Error	0.0579							
t = 0.5	-0.1239	0.1048	0.3429	0.5974	-0.0490	0.1580	0.3770	0.6140
Error	0.0994							

Ecuación de ondas

La ecuación a resolver es:

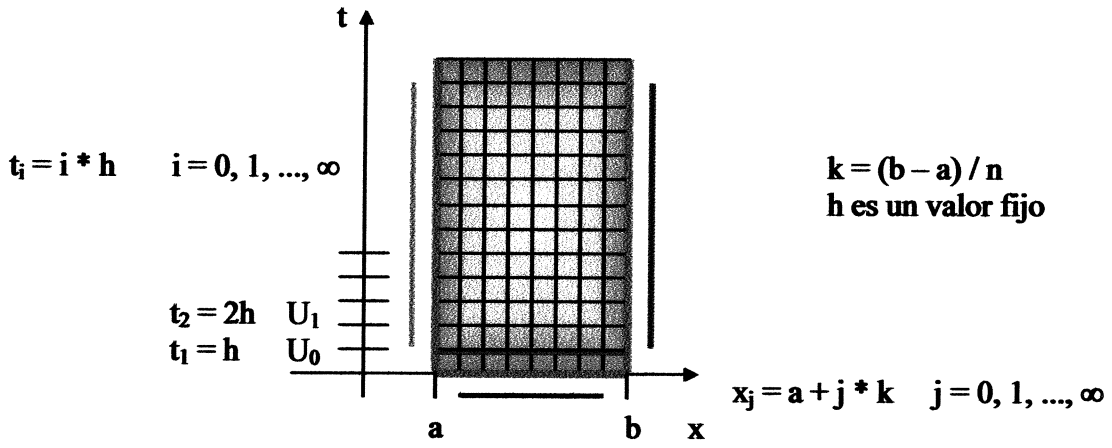
$$u_t = c^2 + u_{xx} + f(x,t)$$

Se trata de averiguar cómo se comporta una onda a lo largo del tiempo, sabiendo el comportamiento inicial de la misma, velocidad que se le aplica inicialmente y el comportamiento de la onda en los extremos de la misma para cualquier instante de tiempo.



Forma de resolver el problema:

Partimos de un gráfico que nuestra cómo se divide el espacio en el que se representa la onda a lo largo del tiempo:



Datos que se conocen acerca del problema:

f(x,t)
$u(a, t) = h_1(t_i)$ _____
$u(b, t) = h_2(t_i)$ _____
$u(x, 0) = g(x_j)$ _____
u(x, etapa1) _____

La ecuación a resolver discretizada es la siguiente:

$$\frac{U_{i,j+1} - 2U_{i,j} + U_{i,j-1}}{h^2} = c^2 * \frac{U_{i+1,j} - 2U_{i,j} + U_{i-1,j-1}}{k^2} + f(x_i, t_j)$$

Esta ecuación tiene un error del orden $O(h^2 + k^2)$, por lo que el método es bueno.

λ va a seguir siendo el coeficiente de estabilidad, pero en este caso se va a calcular como c * h / k.

Para resolver este método de forma sencilla es conveniente conocer inicialmente las dos primeras etapas, ya que a partir de ambas se pueden calcular las demás de la siguiente forma:

$$U_{i,j+1} = \lambda^2 * U_{i-1,j} (2 - 2\lambda^2) * U_{i,j} + \lambda^2 * U_{i+1,j} - U_{i,j-1} + h^2 * f_{ij}$$

El problema planteado de forma vectorial queda de la siguiente forma:

$$W_{j+1} = \begin{pmatrix} U_{1,j+1} \\ U_{2,j+1} \\ \dots \\ U_{n-1,j+1} \end{pmatrix} = \begin{pmatrix} \\ A \\ \\ \end{pmatrix} * \begin{pmatrix} \\ W_j \\ \\ \end{pmatrix} - \begin{pmatrix} \\ W_{j-1} \\ \\ \end{pmatrix} + \begin{pmatrix} \\ b_j \\ \\ \end{pmatrix}$$

A

$$\begin{pmatrix} 2(1-\lambda^2) & \lambda^2 & 0 & 0 & \dots & 0 & 0 & 0 \\ \lambda^2 & 2(1-\lambda^2) & \lambda^2 & 0 & \dots & 0 & 0 & 0 \\ 0 & \lambda^2 & 2(1-\lambda^2) & \lambda^2 & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & 0 & \lambda^2 & 0 \\ 0 & 0 & 0 & 0 & \dots & \lambda^2 & 2(1-\lambda^2) & \lambda^2 \\ 0 & 0 & 0 & 0 & \dots & 0 & \lambda^2 & 2(1-\lambda^2) \end{pmatrix}$$

W_j $\quad$ W_{j-1} $\quad$ b_j

$$\begin{pmatrix} U_{1,j} \\ U_{2,j} \\ \dots \\ U_{n-1,j} \end{pmatrix} \quad \begin{pmatrix} U_{1,j-1} \\ U_{2,j-1} \\ \dots \\ U_{n-1,j-1} \end{pmatrix} \quad \begin{pmatrix} h^2 * f_{1,j} + \lambda^2 * h_1(t_j) \\ h^2 * f_{2,j} \\ \dots \\ h^2 * f_{n-1,j} + \lambda^2 * h_2(t_j) \end{pmatrix}$$

El resultado se obtiene a partir de la ecuación: $W_{j+1} = A * W_j - W_{j-1} + b_j$.

Código del método:

```
function ondas(a,b,n,h,c,tf)
k=(b-a)/n   %n -- nº particiones de x.     k -- distancia de x a x
landa=c*h/k;   %h --  distancia entre las etapas
num_etap=tf/h;  %tf -- tiempo final, de la última etapa
incog=(n-1);
matriz_a=zeros(incog);
for i=1:incog
   matriz_a(i,i)=2*(1-landa.^2);
end
for r=2:incog
   matriz_a(r,r-1)=landa.^2;
   matriz_a(r-1,r)=landa.^2;
end
pos_x=a+k;
ind=1;
while (ind<=incog)
   w1(ind)=base(pos_x);
%w2(ind)=w1(ind)+h*izquierda(pos_x)+h.^2/2*(c.^2*derivada2('base',h,10
^-3,10,pos_x)+der_f(pos_x,0));
w2(ind)=w1(ind)+h*izquierda(pos_x)+h.^2/2*(c.^2*der2_base(pos_x))+
        + der_f(pos_x,0);
   ind=ind+1;  pos_x=pos_x+k;
end
for etapa=2:num_etap
   pos_x=a+k; ind=1;
   while  (ind<=incog)
      w_real(ind)=func(pos_x,etapa*h);
      if (ind==1)
         b1(ind)=h.^2*(der_f(pos_x,etapa*h))+
                 + landa.^2*izquierda(etapa*h);
      else
         if (ind==(n-1))
            b1(ind)=h.^2*(der_f(pos_x,etapa*h))+
                    + landa.^2*derecha(etapa*h);
         else
            b1(ind)=h.^2*(der_f(pos_x,etapa*h));
         end
      end
      pos_x=pos_x+k;  ind=ind+1;
   end
   invb=b1';  invw=w1';  invw2=w2';
   w3=matriz_a*invw2-invw+invb;
   w_real
   w1=w2;
   w2=w3'
   error=norm(w_real-w2)
end
matriz_a
```

Esta línea sustituiría a la línea siguiente en caso de que no se conociese la derivada segunda de la función base.

El método utiliza las siguientes funciones:

der_f	⟶	f(x,t)
base	⟶	g(x)
der2_base	⟶	g''(x)
izquierda	⟶	u(a,t)
derecha	⟶	u(b,t)
func	⟶	función original

EJERCICIOS:

1.- Obtener el comportamiento de una onda para t = 0.002, t = 0.004, t = 0.006, t = 0.008 y t = 0.010 conociendo los siguientes datos:

$a = 0$
$b = 0.5$
$n = 5$
$h = 0.002$
$c = 1$
$u(x,0) = 2x$
$u''(x,0) = 2$
$u(0,t) = 0$
$u(0.5,t) = t^2 + 1.25$
$f(x,t) = 2tx^2 - 2t$

Comprobar el resultado sabiendo que la solución exacta se rige por la ecuación:

$$u(x,t) = t^2x^2 + 2x$$

Solución:

1.- Definir las funciones auxiliares con los datos que se conocen del problema:

```
function y=der_f(x,t)
y=2*t*x.^2-2*t;

function y=base(x)
y=2*x;

function y=der2_base(x)
y=0;

function y=izquierda(t)
y=0;

function y=derecha(t)
y=t.^2+1.25;
```

2.- Definir la función original:

```
function y=func(x,t)
y=t.^2*x.^2+2*x;
```

3.- Calcular el resultado:

ondas (0,0.5,5,0.002,1,0.010)

w_real = 0.2000 0.4000 0.6000 0.8000
w2 = 0.2000 0.4000 0.6000 0.8001
error = 5.8875e-006

w_real = 0.2000 0.4000 0.6000 0.8000
w2 = 0.2000 0.4000 0.6000 0.8003
error = 1.0285e-004

w_real = 0.2000 0.4000 0.6000 0.8000
w2 = 0.2000 0.4000 0.6000 0.8006
error = 3.0214e-004

w_real = 0.2000 0.4000 0.6000 0.8000
w2 = 0.2000 0.4000 0.6000 0.8010
error = 6.0005e-004

matriz_a =

$$\begin{pmatrix} 1.9992 & 0.0004 & 0 & 0 & 0 \\ 0.0004 & 1.9992 & 0.0004 & 0 & 0 \\ 0 & 0.0004 & 1.9992 & 0.0004 & 0 \\ 0 & 0 & 0.0004 & 1.9992 & 0.0004 \\ 0 & 0 & 0 & 0.0004 & 1.9992 \end{pmatrix}$$

4.- Lectura del resultado:

$k = (0 - 1) / 5 = 0.1$, por lo que la diferencia entre cada x es 0.1.

	Método				Real			
	x = 0.1	x = 0.2	x = 0.3	x = 0.4	x = 0.1	x = 0.2	x = 0.3	x = 0.4
t = 0.004	0.2000	0.4000	0.6000	0.8001	0.2000	0.4000	0.6000	0.8000
Error	0.0000058875							
t = 0.006	0.2000	0.4000	0.6000	0.8003	0.2000	0.4000	0.6000	0.8000
Error	0.00010285							
t = 0.008	0.2000	0.4000	0.6000	0.8006	0.2000	0.4000	0.6000	0.8000
Error	0.00030214							
t = 0.010	0.2000	0.4000	0.6000	0.8010	0.2000	0.4000	0.6000	0.8000
Error	0.00060005							

Como puede observarse, el error que se obtiene en cada una de las etapas es muy pequeñito por lo que se ve que el método es bastante bueno.

2.- Obtener el comportamiento de la onda con los siguientes datos:

$a = 0$	$tf = 1$
$b = 1$	$u(x,0) = sen(\pi x)$
$n = 10$	$u''(x,0) = 0$
$h = 0.05$	$u(0,t) = 0$
$k = 0.1$	$u(1,t) = 0$
$\lambda = 1$	$f(x,t) = 0$

Comprobar el resultado sabiendo que la solución exacta se rige por la ecuación:

$$u(x,t) = sen(\pi x)cos(\pi 2t)$$

Solución:

1.- Definir las funciones auxiliares con los datos que se conocen del problema:

```
function y=der_f(x,t)
y=0;

function y=base(x)
y=sin(pi*x);

function y=der2_base(x)
y=0;
```

```
function y=izquierda(t)
y=0;

function y=derecha(t)
y=0;
```

2.- Definir la función original:

```
function y=func(x,t)
y=sin(pi*x)*cos(2*pi*t);
```

3.- Calcular el resultado:

$\lambda = 1$, $\lambda = c * h / k$, $1 = c * 1.05 / 0.1$, $c = 2$

ondas (0,1,10,0.05,2,1)

Como se generan 20 etapas (1/0.05) voy a mostrar únicamente la etapa final.

w_real = 0.3090 0.5878 0.8090 0.9511 1.0000 0.9511 0.8090 0.5878 0.3090
w2 = 0.3090 0.5878 0.8090 0.9511 1.0000 0.9511 0.8090 0.5878 0.3090
error = 1.0084e-015

matriz_a =

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

ANEXO

EJERCICIOS:

1.- (Sección 12.1.- Ejercicio 1.)

Aproximar, mediante el algoritmo de Laplace, la solución de la ecuación diferencial parcial elíptica.

$$f(x,y) = u_{xx} + u_{yy} = 4$$
$$u(x,0) = x^2$$
$$u(x,2) = (x-2)^2$$
$$u(0,y) = y^2$$
$$u(1,y) = (y-1)^2$$

a) **Obtener y dibujar u en el interior del rectángulo.**
b) **Comprobar el resultado con la ecuación** $u(x,y) = (x-y)^2$

Solución:

1.- Definir las aristas del rectángulo:

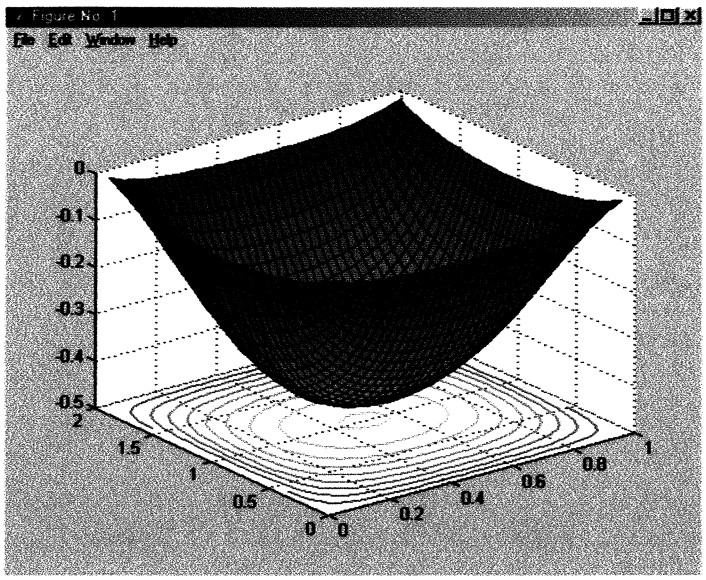
```
function y=arista_abajo(x)
y=x.^2;

function y=arista_arriba(x)
y=(x-2).^2;

function y=arista_izquierda(x)
y=x.^2;

function y=arista_derecha(x)
y=(x-1).^2;
```

2.- Definir la función f:

```
function z=der2x_der2y(x,y)
z=4;
```

3.- Ejecutar el método:

matriz_u = laplace(0,1,0,2,40,40)



4.- Definir la ecuación real y cálcular el error:

```
function z=func_real(x,y)
z=(x-y).^2;
```

error_laplace(0,1,0,2,40,40,matriz_u) **3.7111**

2.- (Sección 12.1.- Ejercicio 8.)

Una placa rectangular de plata de 6 x 5 cm. Tien calor que se genera uniformemente en todos los puntos con una rapidez q = 1.5 cal/cm^3·s. Representemos con x la distancia a lo largo del borde de la placa de una longitud de 6 cm. Y con y la distancia a lo largo del borde de la placa de una longitud de 5 cm. Supóngase que la temperatura u a lo largo de los bordes se mantiene en las siguientes temperaturas:

$$u(x,0) = x(6-x) \qquad u(x,5) = 0 \qquad 0 \le x \le 6$$
$$u(0,y) = y(5-y) \qquad u(6,y) = 0 \qquad 0 \le x \le 5$$

donde el origen se encuentra en una esquina de la placa con las coordenasa (0,0) y los bordes se hallan a lo largo de los ejes positivos x e y. La temperatura de estado estable u = u(x,y) satisface la ecuación de Poisson:

$$f(x,y) = u_{xx} + u_{yy} = -q / K$$

donde K, la conductividad térmica, es 1.04 cal/cm·deg·s. Dibujar la temperatura u(x,y) para un número elevado de particiones de x e y.

Solución:

1.- Definir las aristas del rectángulo:

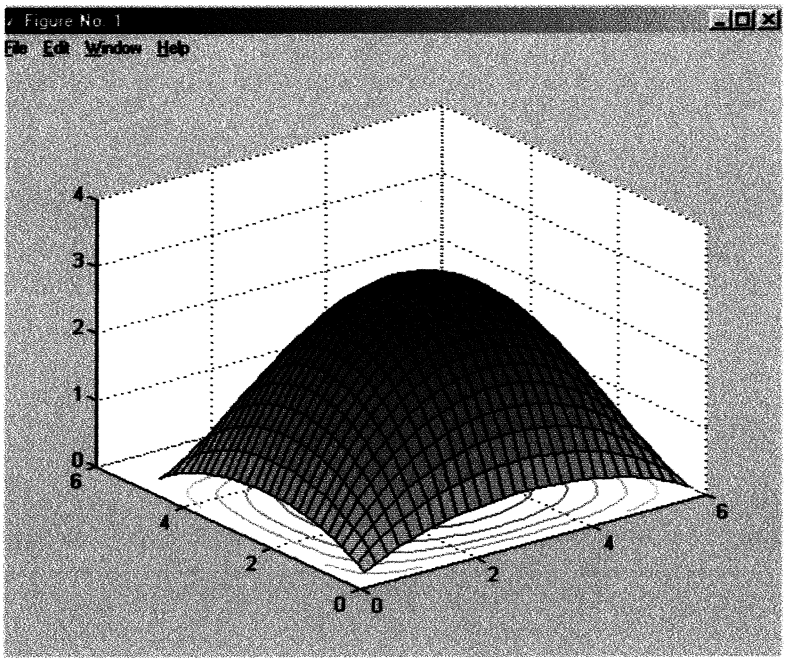
```
function y=arista_abajo(x)
y=x*(6-x);

function y=arista_arriba(x)
y=0;

function y=arista_izquierda(x)
y=x*(5-x);

function y=arista_derecha(x)
y=0;
```

2.- Definir la función f:

```
function z=der2x_der2y(x,y)
z=-1.5/1.04;
```

3.- Ejecutar el método:

matriz_u = laplace(0,6,0,5,30,30)



3.- (Sección 12.2.- Ejercicio 3.)

Usar el método de diferencias progresivas para aproximar la solución de las siguientes ecuaciones diferenciales:

a) $f(x,t) = 0.$ $\quad 0 < x < 2.$ $\quad 0 < t$
$u(0,t) = u(2,t) = 0.$ $\quad 0 < t$
$u(x,0) = \text{sen}(2\pi x)$ $\quad 0 \le x \le 2$

Usar $k = 0.4$ y $h = 0.1$, comparar la respuesta en $t = 0.5$ con la solución real $u(x,t) = e^{-4\pi^2 t} * \text{sen}(2\pi x)$. Después usar $k = 0.4$ y $k = 0.05$.

b) $f(x,t) = 0.$ $\quad 0 < x < \pi.$ $\quad 0 < t$
$u(0,t) = u(\pi,t) = 0.$ $\quad 0 < t$
$u(x,0) = \text{sen}(x)$ $\quad 0 \le x \le \pi$

Usar $k = \pi/10$ y $h = 0.05$, comparar la respuesta con la solución real $u(x,t) = e^{-t} * \text{sen}(x)$ en $t = 0.5$.

c) $f(x,t) = 4/\pi^2.$ $\quad 0 < x < 4.$ $\quad 0 < t$
$u(0,t) = u(4,t) = 0.$ $\quad 0 < t$
$u(x,0) = \text{sen}((\pi/4) * x) * (1 + 2\cos((\pi/4) * x))$ $\quad 0 \le x \le \pi$

Usar $k = 0.2$ y $h = 0.04$, comparar la respuesta en $t = 0.4$ con la solución real $u(x,t) = e^{-t} * \text{sen}(\frac{\pi}{2}x) + e^{-t/4} * \text{sen}(\frac{\pi}{4}x)$

d) $f(x,t) = 1/\pi^2.$ $\quad 0 < x < 1.$ $\quad 0 < t$
$u(0,t) = u(1,t) = 0.$ $\quad 0 < t$
$u(x,0) = \cos(x) * (x - ½)$ $\quad 0 \le x \le 1$

Usar $k = 0.1$ y $h = 0.04$, comparar la respuesta en $t = 0.4$ con la solución real $u(x,t) = e^{-t} * \cos(\pi) * (x - 1/2)$

Solución:

a)

1.- Definir las funciones auxiliares con los datos que se conocen del problema:

```
function y=der_f(x,t)
y=0;

function y=base(x)
y=sin(2*pi*x);

function y=izquierda(t)
y=0;

function y=derecha(t)
y=0;
```

2.- Definir la función original:

```
function y=func(x,t)
y=exp(-4*pi.^2-t)*sin(2*pi*x);
```

3.- Calcular el resultado:

- **k = 0.4 y h = 0.1.**

Para que k sea igual a 0.4, el valor de n debe ser 5, ya que (2 – 0) / 5 = 0.4.

Para que el método progresivo sea estable, el valor λ debe ser menor que ½, por tanto en este caso c no puede ser 1 ya que si c = 1 entonces $\lambda = 1^2 * 0.1 / (0.4)^2 = 0.625$ valor mayor que 0.5. Para que el método sea estable se debe tomar para c un valor menor que 0.9, por ejemplo se puede tomar c = 0.5 que proporciona un valor para λ de 0.16.

Para t = 0.5 la solución es:

```
progresivo (0,2,5,0.1,0.5,0.5)

w_real =  1.0e-017 * (0.2552   -0.4129    0.4129   -0.2552)
w2 =    0.0091   -0.0148    0.0148   -0.0091
error =   0.0245
```

- **k = 0.4 y h = 0.05.**

En este caso, c = 0.5 sigue siendo válido ya que $\lambda = (0.5)^2 * 0.05 / (0.4)^2 = 0.078 < 0.5$.

Para t = 0.5 la solución es:

progresivo (0,2,5,0.05,0.5,0.5)

w_real = 1.0e-017 * (0.2552 -0.4129 0.4129 -0.2552)
w2 = 0.0212 -0.0343 0.0343 -0.0212
error = 0.0570

b)

1.- Definir las funciones auxiliares con los datos que se conocen del problema:

```
function y=der_f(x,t)
y=0;

function y=base(x)
y=sin(x);

function y=izquierda(t)
y=0;

function y=derecha(t)
y=0;
```

2.- Definir la función original:

```
function y=func(x,t)
y=exp(-t)*sin(x);
```

3.- Calcular el resultado:

Para que k sea igual a $\pi/10$, el valor de n debe ser 10.

Para t = 0.5 la solución es:

progresivo (0,pi,10,0.05,1,0.5)

w_real = 0.1874 0.3565 0.4907 0.5768 0.6065 0.5768 0.4907 0.3565 0.1874
w2 = 0.1858 0.3535 0.4865 0.5719 0.6013 0.5719 0.4865 0.3535 0.1858
error = 0.0116

c)

1.- Definir las funciones auxiliares con los datos que se conocen del problema:

```
function y=der_f(x,t)
y=4/pi.^2;

function y=base(x)
y=sin((pi/4)*x)*(1+2*cos((pi/4)*x));

function y=izquierda(t)
y=0;

function y=derecha(t)
y=0;
```

2.- Definir la función original:

```
function y=func(x,t)
y=exp(-t)*sin((pi/2)*x)+exp(-t/4)*sin((pi/4)*x);
```

3.- Calcular el resultado:

Como k debe ser igual a 0.2, el valor de n debe ser 20, ya que (4 – 0) / 20 = 0.2.

De nuevo, para que el método sea estable, el valor de λ debe ser menor que ½, y si se toma c = 1 esto no se cumple, ya que $\lambda = 1^2 * 0.04 / (0.2)^2 = 1$. Un valor de c que proporciona a λ un valor menor que ½ (concretamente 0.49) es 0.7.

Para t = 0.4 la solución es:

progresivo (0,4,20,0.04,0.7,0.4)

w_real = 0.3487 0.6736 0.9531 1.1694 1.3101 1.3695 1.3485 1.2546 1.1008
0.9048 0.6866 0.4665 0.2639 0.0945 -0.0305 -0.1057 -0.1315 -0.1144 -0.0656
w2 = 0.3957 0.7446 1.0336 1.2529 1.3953 1.4587 1.4456 1.3639 1.2259
1.0478 0.8479 0.6448 0.4559 0.2952 0.1719 0.0894 0.0438 0.0255 0.0177
error = 0.6038

En este apartado se ha obtenido un error más alto que en los ejercicios anteriores.

d)

1.- Definir las funciones auxiliares con los datos que se conocen del problema:

```
function y=der_f(x,t)
y=1/pi.^2;

function y=base(x)
y=cos(pi)*(x-1/2);

function y=izquierda(t)
y=0;

function y=derecha(t)
y=0;
```

2.- Definir la función original:

```
function y=func(x,t)
y=exp(-t)*cos(pi)*(x-1/2);
```

3.- Calcular el resultado:

Como k debe ser igual a 0.1, el valor de n debe ser 10, ya que (1 – 0) / 10 = 0.1.

Para que el método sea estable utilizo c = 0.1.

Para t = 0.4 la solución es:

progresivo (0,1,10,0.04,0.1,0.4)

w_real = 0.2681 0.2011 0.1341 0.0670 0 -0.0670 -0.1341 -0.2011 -0.2681
w2 = 0.2906 0.3161 0.2380 0.1403 0.0405 -0.0593 -0.1570 -0.2362 -0.2215
error = 0.1885

4.- (Sección 12.2.- Ejercicio 4.)

Repetir el ejercicio anterior usando el método de diferencias regresivas.

Solución:

a)

1.- Definir las funciones auxiliares con los datos que se conocen del problema:

```
function y=der_f(x,t)
y=0;

function y=base(x)
y=sin(2*pi*x);

function y=izquierda(t)
y=0;

function y=derecha(t)
y=0;
```

2.- Definir la función original:

```
function y=func(x,t)
y=exp(-4*pi.^2-t)*sin(2*pi*x);
```

3.- Calcular el resultado:

- **k = 0.4 y h = 0.1.**

Para que k sea igual a 0.4, el valor de n debe ser 5, ya que (2 – 0) / 5 = 0.4.

Este método es siempre estable, por tanto se puede utilizar c = 1 ya que no tiene la restricción que presentaba el método progresivo.

Para t = 0.5 la solución es:

```
regresivo (0,2,5,0.1,1,0.5)

w_real =  1.0e-017 * (0.2552  -0.4129   0.4129  -0.2552)
w2 =   0.0016  -0.0026   0.0026  -0.0016
error =  0.0043
```

- **k = 0.4 y h = 0.05.**

Para t = 0.5 la solución es:

```
progresivo (0,2,5,0.05,1,0.5)

w_real =   1.0e-017 * (0.2552  -0.4129   0.4129  -0.2552)
w2 =    1.0e-003 * (0.3049  -0.4933   0.4933  -0.3049)
error =  8.2012e-004
```

b)

1.- Definir las funciones auxiliares con los datos que se conocen del problema:

```
function y=der_f(x,t)
y=0;

function y=base(x)
y=sin(x);

function y=izquierda(t)
y=0;

function y=derecha(t)
y=0;
```

2.- Definir la función original:

```
function y=func(x,t)
y=exp(-t)*sin(x);
```

3.- Calcular el resultado:

Para que k sea igual a $\pi/10$, el valor de n debe ser 10.

Para t = 0.5 la solución es:

```
regresivo (0,pi,10,0.05,1,0.5)
```

w_real = 0.1874 0.3565 0.4907 0.5768 0.6065 0.5768 0.4907 0.3565 0.1874
w2 = 0.1905 0.3623 0.4986 0.5862 0.6163 0.5862 0.4986 0.3623 0.1905
error = 0.0219

c)

1.- Definir las funciones auxiliares con los datos que se conocen del problema:

```
function y=der_f(x,t)
y=4/pi.^2;

function y=base(x)
y=sin((pi/4)*x)*(1+2*cos((pi/4)*x));

function y=izquierda(t)
y=0;

function y=derecha(t)
y=0;
```

2.- Definir la función original:

```
function y=func(x,t)
y=exp(-t)*sin((pi/2)*x)+exp(-t/4)*sin((pi/4)*x);
```

3.- Calcular el resultado:

Como k debe ser igual a 0.2, el valor de n debe ser 20, ya que (4 – 0) / 20 = 0.2.

Para t = 0.4 la solución es:

```
regresivo (0,4,20,0.04,0.7,0.4)
```

w_real = 0.3487 0.6736 0.9531 1.1694 1.3101 1.3695 1.3485 1.2546 1.1008
0.9048 0.6866 0.4665 0.2639 0.0945 -0.0305 -0.1057 -0.1315 -0.1144 -0.0656
w2 = 0.3984 0.7499 1.0419 1.2636 1.4077 1.4713 1.4571 1.3728 1.2312
1.0488 0.8443 0.6370 0.4443 0.2807 0.1558 0.0730 0.0291 0.0141 0.0115
error = 0.6038

Con este método, al igual que con el método progresivo, el error es grande.

d)

1.- Definir las funciones auxiliares con los datos que se conocen del problema:

```
function y=der_f(x,t)
y=1/pi.^2;

function y=base(x)
y=cos(pi)*(x-1/2);

function y=izquierda(t)
y=0;

function y=derecha(t)
y=0;
```

2.- Definir la función original:

```
function y=func(x,t)
y=exp(-t)*cos(pi)*(x-1/2);
```

3.- Calcular el resultado:

Como k debe ser igual a 0.1, el valor de n debe ser 10, ya que (1 – 0) / 10 = 0.1.

Para t = 0.4 la solución es:

```
regresivo (0,1,10,0.04,0.1,0.4)
```

w_real = 0.2681 0.2011 0.1341 0.0670 0 -0.0670 -0.1341 -0.2011 -0.2681
w2 = 0.2961 0.3146 0.2369 0.1401 0.0405 -0.0591 -0.1561 -0.2354 -0.2286
error = 0.1858

5.- (Sección 12.2.- Ejercicio 5.)

Repetir el mismo ejercicio con el método de Crank Nicolson.

Solución:

a)

1.- Definir las funciones auxiliares con los datos que se conocen del problema:

```
function y=der_f(x,t)
y=0;

function y=base(x)
y=sin(2*pi*x);

function y=izquierda(t)
y=0;

function y=derecha(t)
y=0;
```

2.- Definir la función original:

```
function y=func(x,t)
y=exp(-4*pi.^2-t)*sin(2*pi*x);
```

3.- Calcular el resultado:

- k = 0.4 y h = 0.1.

Para que k sea igual a 0.4, el valor de n debe ser 5, ya que (2 – 0) / 5 = 0.4.

Como este método también es siempre estable, c puede ser 1.

Para t = 0.5 la solución es:

nicolson (0,2,5,0.1,1,0.5)

w_real = 1.0e-017 * (0.2552 -0.4129 0.4129 -0.2552)
w2 = 1.0e-006 * (-0.5093 0.8241 -0.8241 0.5093)
error = 1.3700e-006

- k = 0.4 y h = 0.05.

Para t = 0.5 la solución es:

nicolson (0,2,5,0.05,1,0.5)

w_real = 1.0e-017 * (0.2552 -0.4129 0.4129 -0.2552)
w2 = 1.0e-005 * (0.1603 -0.2594 0.2594 -0.1603)
error = 4.3118e-006

b)

1.- Definir las funciones auxiliares con los datos que se conocen del problema:

```
function y=der_f(x,t)
y=0;

function y=base(x)
y=sin(x);

function y=izquierda(t)
y=0;

function y=derecha(t)
y=0;
```

2.- Definir la función original:

```
function y=func(x,t)
y=exp(-t)*sin(x);
```

3.- Calcular el resultado:

Para que k sea igual a π/10, el valor de n debe ser 10.

Para t = 0.5 la solución es:

nicolson (0,pi,10,0.05,1,0.5)

w_real = 0.1874 0.3565 0.4907 0.5768 0.6065 0.5768 0.4907 0.3565 0.1874
w2 = 0.1882 0.3579 0.4927 0.5792 0.6090 0.5792 0.4927 0.3579 0.1882
error = 0.0054

c)

1.- Definir las funciones auxiliares con los datos que se conocen del problema:

```
function y=der_f(x,t)
y=4/pi.^2;

function y=base(x)
y=sin((pi/4)*x)*(1+2*cos((pi/4)*x));

function y=izquierda(t)
y=0;

function y=derecha(t)
y=0;
```

2.- Definir la función original:

```
function y=func(x,t)
y=exp(-t)*sin((pi/2)*x)+exp(-t/4)*sin((pi/4)*x);
```

3.- Calcular el resultado:

Como k debe ser igual a 0.2, el valor de n debe ser 20, ya que (4 – 0) / 20 = 0.2.

Para t = 0.4 la solución es:

nicolson (0,4,20,0.04,0.7,0.4)

w_real = 0.3487 0.6736 0.9531 1.1694 1.3101 1.3695 1.3485 1.2546 1.1008
0.9048 0.6866 0.4665 0.2639 0.0945 -0.0305 -0.1057 -0.1315 -0.1144 -0.0656
w2 = 0.3971 0.7473 1.0378 1.2583 1.4016 1.4651 1.4515 1.3684 1.2286
1.0483 0.8461 0.6408 0.4500 0.2878 0.1637 0.0810 0.0364 0.0197 0.0146
error = 0.5948

El error que se obtiene es el mismo que con el método regresivo.

d)

1.- Definir las funciones auxiliares con los datos que se conocen del problema:

```
function y=der_f(x,t)
y=1/pi.^2;

function y=base(x)
y=cos(pi)*(x-1/2);

function y=izquierda(t)
y=0;

function y=derecha(t)
y=0;
```

2.- Definir la función original:

```
function y=func(x,t)
y=exp(-t)*cos(pi)*(x-1/2);
```

3.- Calcular el resultado:

Como k debe ser igual a 0.1, el valor de n debe ser 10, ya que (1 – 0) / 10 = 0.1.

Para t = 0.4 la solución es:

nicolson (0,1,10,0.04,0.1,0.4)

w_real = 0.2681 0.2011 0.1341 0.0670 0 -0.0670 -0.1341 -0.2011 -0.2681
w2 = 0.2961 0.3146 0.2369 0.1401 0.0405 -0.0591 -0.1561 -0.2354 -0.2286
error = 0.1858

6.- (Sección 12.2.- Ejercicio 13.)

Sagar y Payne [SP] analizan las relaciones de esfuerzo-deformación y las propiedades materiales de un cilindro sujeto alternatiamente al calentamiento y al enfriamiento, y consideran la ecuación

$$\frac{\partial^2 T}{\partial r^2}+\frac{1}{r}\frac{\partial T}{\partial r}=\frac{1}{4K}\frac{\partial T}{\partial t} \qquad \frac{1}{2}\le r\le 1, \quad 0\le T$$

donde T = T(r,t) es la temperatura, r es la distancia radial respecto al centro del cilindro, t es el tiempo y K es el coeficiente de difusividad.

Obtener las aproximaciones a T(r, 10) para un cilindro con radio externo 1, dadas las condiciones iniciales y de frontera:

$$\begin{aligned} T(1,t) &= 100 + 40t & 0 \le t \le 10 \\ T(1/2, t) &= t & 0 \le t \le 10 \\ T(r,0) &= 200(r - 0.5) & 0.5 \le t \le 1 \end{aligned}$$

Usar para resolver el problema el método de diferencias regresivas con K =0.1, h = 0.5 y k = 0.1.

Solución:

1.- Definir las funciones auxiliares con los datos que se conocen del problema:

```
function y=der_f(r,t)
K=0.1;
y=(1/(4*K))+1/r;

function y=base(r)
y=200*(r-0.5);

function y=izquierda(t)
y=t;

function y=derecha(t)
y=100+40*t;
```

2.- Calcular el resultado:

Para que k sea igual a 0.1, el valor de n debe ser 5.

Para t = 10 la solución es:

```
regresivo (0.5,1,5,0.5,1,10)
```

w2 = 137.7471 245.5406 340.4608 424.5874

7.- (Sección 12.3.- Ejercicio 1.)

Aproximar la solución de la ecuación de onda con los siguientes datos:

$f(x,t) = 0.$ $\quad 0 < x < 1. \quad 0 < t$
$u(0,t) = u(1,t) = 0.$ $\quad 0 < t$
$u(x,0) = \text{sen}(\pi x)$ $\quad 0 \le x \le 1$
$u''(x,0) = 0.$ $\quad 0 \le x \le 1$
$n = 4 \quad tf = 1$

Comprobar el resultado sabiendo que la solución exacta se rige por la ecuación:

$$u(x,t) = \cos(\pi t) * \text{sen}(\pi x)$$

Solución:

1.- Definir las funciones auxiliares con los datos que se conocen del problema:

```
function y=der_f(x,t)
y=0;

function y=base(x)
y=sin(pi*x);

function y=der2_base(x)
y=0;

function y=izquierda(t)
y=0;

function y=derecha(t)
y=0;
```

2.- Definir la función original:

```
function y=func(x,t)
y=cos(pi*t)*sin(pi*x);
```

3.- Calcular el resultado:

Como el método es ideal cuando $\lambda = 1$, como $\lambda = c * h / k$, si se toma $c = 1$ entonces $h = 0.25$ ya que $k = 0.25$.

Para $t = 1$ la solución es: **ondas** (0,1,4,0.25,1,1)

w_real = -0.7071 -1.0000 -0.7071
w2 = -0.7071 -1.0000 -0.7071
error = 1.5701e-016

	x = 0.25	x = 0.5	x = 0.75
Real	-0.7071	-1.0000	-0.7071
Método	-0.7071	-1.0000	-0.7071
Error	1.5701e-016		

8.- (Sección 12.3.- Ejercicio 3.)

Aproximar la solución de la ecuación de onda con los siguientes datos:

$f(x,t) = 0.$	$0 < x < \pi.$	$0 < t$
$u(0,t) = u(\pi,t) = 0.$	$0 < t$	
$u(x,0) = \text{sen}(x)$	$0 \leq x \leq \pi$	
$u''(x,0) = 0.$	$0 \leq x \leq \pi$	

con $k = \pi/10$ y $h = 0.05$, con $k = \pi/20$ y $h = 0.1$, y luego con $k = \pi/20$ y con $h = 0.05$. Comparar los resultados con la solución real $u(x,t) = \cos(t) * \text{sen}(x)$ en $t = 0.5$.

Solución:

1.- Definir las funciones auxiliares con los datos que se conocen del problema:

```
function y=der_f(x,t)
y=0;

function y=base(x)
y=sin(x);

function y=der2_base(x)
y=0;

function y=izquierda(t)
y=0;

function y=derecha(t)
y=0;
```

2.- Definir la función original:

```
function y=func(x,t)
y=cos(t)*sin(x);
```

3.- Calcular el resultado:

a) $k = \pi/10$ y $h = 0.05$.

Como el enunciado indica que el valor de k debe ser $\pi/10$ y $k = (b - a) / n$, entonces $n = 10$.

Para $t = 0.5$ la solución es:

ondas (0,pi,10,0.05,1,0.5)

w_real =	0.2712 0.5158 0.7100 0.8346 0.8776 0.8346 0.7100 0.5158 0.2712
w2 =	0.2752 0.5234 0.7204 0.8469 0.8904 0.8469 0.7204 0.5234 0.2752
error =	0.0287

b) $k = \pi/20$ y $h = 0.1$.

Como k debe ser $\pi/20$, entonces n vale 10.

Para $t = 0.5$ la solución es:

ondas (0,pi,20,0.1,1,0.5)

w_real = 0.1373	0.2712	0.3984	0.5158	0.6205	0.7100	0.7819	
0.8346	0.8668	0.8776	0.8668	0.8346	0.7819	0.7100	0.6205
0.5158	0.3984	0.2712	0.1373				
w2 = 0.1411	0.2786	0.4094	0.5300	0.6376	0.7295	0.8034	
0.8576	0.8906	0.9017	0.8906	0.8576	0.8034	0.7295	0.6376
0.5300	0.4094	0.2786	0.1411				
error = 0.0762							

c) $k = \pi/20$ y $h = 0.05$.

Para $t = 0.5$ la solución es:

ondas (0,pi,20,0.05,1,0.5)

w_real = 0.1373	0.2712	0.3984	0.5158	0.6205	0.7100		
0.7819	0.8346	0.8668	0.8776	0.8668	0.8346	0.7819	0.7100
0.6205	0.5158	0.3984	0.2712	0.1373			
w2 = 0.1392	0.2750	0.4039	0.5230	0.6292	0.7198		
0.7928	0.8462	0.8788	0.8898	0.8788	0.8462	0.7928	0.7198
0.6292	0.5230	0.4039	0.2750	0.1392			
error = 0.0385							