Métodos Numéricos para las Ecuaciones Diferenciales.
Grado en Matemáticas. Universidad de Murcia.
Práctica computacional 1.

Observación general: Para evitar problemas de compatibilidad con las fuentes desde distintos sistemas operativos se recomienda no usar espacios ni caracteres latinos (tildes, etc...) en los nombres de los directorios o los códigos. Tampoco es recomendable usar caracteres latinos en los comentarios dentro de los códigos.

Objetivos de la práctica: Observar la importancia de la precisión y aprender a estimar el orden de convergencia de dos métodos distintos. Hacer ejemplos de sistemas.

Ejemplos:

1. En el código **EulerExplicito.m** se muestra como programar de una forma simple el método de Euler explícito. El esquema de programación sirve para cualquier método de un paso. Aquí lo aplicamos al problema

$$\begin{cases} y'(t) &= -a\,y(t), \quad t \in [0,3], \\ y(0) &= 1 \end{cases}$$

cuya solución es $y(t) = e^{-at}$.

```
clear all;
t0=0;
T=3;
y0=1;
N=20;
h=T/N;
a=10;
t=t0:h:t0+T;
y=zeros(N+1,1);
y(1)=y0;
true=t0:0.01:t0+T;
 ztrue=exp(-a*true);
for j=1:N
    y(j+1)=y(j)+h*mifun1(a,t(j),y(j));
end
figure(1);
plot(t,y,'*-',true,ztrue,'r-');
```

El archivo **mifun1.m** es:

```
function dydt=mifun1(aa,tt,yy)
dydt=-aa*yy;
end
```

2. En el código **RK4vsEulerExplicitoOrdenes.m** se muestra la programacion del método de Runge-Kutta clásico y la del método de Euler explícito. Se obtienen además las curvas de pendientes de cada uno de los métodos comprobandose así que el método de Runge-Kutta clásico posee orden 4 mientras que Euler explícito orden 1. Se aplican ambos al problema

$$\begin{cases} y'(t) &= \lambda \cos(\lambda t) y(t), \quad t \in [0, 20], \\ y(0) &= y0 \end{cases}$$

cuya solución es $y(t) = y_0 e^{\sin(\lambda t))}$ con $y0 = 2$ y $\lambda = 7$. Esta solución es oscilante y necesita precisión para poder ser calculada. Se puede usar **dfield.jar** para observar el comportamiento del campo de vectores y sus soluciones.

```
%
% Contrastamos Runge-Kutta clasico de cuatro etapas y de orden 4 con
% Euler explicito estimamos los ordenes  de convergencia usando las
% rectas de pendiente
%
t0=0; %Tiempo inicial
tf=20;  %Tiempo final
T=tf-t0;% Tiempo total
y0=2; %Dato inicial
%Datos para la solucion exacta
kk=7;
% Solucion exacta en una particion fina
ttrue=t0:0.001:t0+T; %Particion fina
ytrue=y0*exp(sin(kk*ttrue));
%
% Numero de aproximaciones a realizar
%
M=18;
nP=zeros(1,M);% guarda el numero de puntos de la particion en cada calculo
errEuler=zeros(1,M); %guarda el error obtenido con Euler
errRK=zeros(1,M); %guarda el error obtenido con RK
%
% Numero de puntos iniciales
%
N=10;
for j=1:M
    nP(j)=N;% Se guarda el numero de puntos a usar
    h=T/N; % Talla de la particion
    t=t0:h:t0+T; %Particion
    h=T/N;% talla de la particion
% Vector para Runge-Kutta orden 4
```

```
    yRK=zeros(1,N+1);% dimensionaliza t e y
    yRK(1)=y0;
% Vector para Euler orden 1
    yEuler=zeros(1,N+1);% dimensionaliza t e y
    yEuler(1)=y0;
% Solucion exacta en la particion
    yt=y0*exp(sin(kk*t));
for n=1:N
    %Calculo RK
    k1=mifun2(kk,t(n),yRK(n));
    k2=mifun2(kk,t(n)+h/2,yRK(n)+h/2*k1);
    k3=mifun2(kk,t(n)+h/2,yRK(n)+h/2*k2);
    k4=mifun2(kk,t(n)+h,yRK(n)+h*k3);
    yRK(n+1)=yRK(n)+h*(k1+2*k2+2*k3+k4)/6;
    %Calculo Euler
    yEuler(n+1)=yEuler(n)+h*mifun2(kk,t(n),yEuler(n));
end
figure(1);
plot(t,yEuler,'+-',t,yRK,'d-',ttrue,ytrue,'r-');
legend('Euler','RK','exacta','Location','Best');
title([' Modelo  con kk= ',num2str(kk),...
       ': RK vs Euler con N= ',num2str(N)]);
errEuler(j)=max(abs(yEuler-yt));
errRK(j)=max(abs(yRK-yt));
   pause(1);
N=2*N; %duplicamos N
disp(['Errores: N= ',num2str(N),' Euler = ',num2str(errEuler(j)),...
    ' RK = ',num2str(errRK(j))])
pause;
end
%
% Visualizamos ahora los datos globales del calculo
%
figure(2)
plot(nP,log(errEuler),'-*',nP,log(errRK),'-+');
legend('LogErrEuler','LogErrRK','Location','Best');
title([' Modelo con kk= ',num2str(kk),...
       ': Decaimiento log errores Euler vs RK ']);
figure(3)
plot(log(nP),log(errEuler),'*',log(nP),-log(nP),'-',...
    log(nP),log(errRK),'+',log(nP),-4*log(nP),'-.');
legend('Euler','-1','RK','-4','Location','Best');
title([' Modelo con kk= ',num2str(kk),...
```

```
': Ordenes convergencia Euler vs RK ']);
```

En este caso el archivo **mifun2.m** es:

```
function dydt=mifun2(kk,tt,yy)
dydt=kk*cos(kk*tt)*yy;
end
```

3. Resolución de un sistema lineal 2d usando herramientas matriciales de Matlab o de Octave. El problema es

$$\begin{cases} u' &= 9u + 24v + 5\cos(t) - \frac{1}{3}\sin(t), \quad u(0) = 4/3 \\ v' &= -24u - 51v - 9\cos(t) + \frac{1}{3}\sin(t), \quad v(0) = 2/3 \end{cases}$$

que se puede escribir en forma matricial como

$$X'(t) = AX(t) + F(t)$$

donde $X = (u; v)$, $A = [9\ 24; -24\ -51]'$ y $F(t) = [f1(t); f2(t)]'$ para $f1(t) = 5\cos(t) - \frac{1}{3}\sin(t)$ y $f2(t) = -9\cos(t) + \frac{1}{3}\sin(t)$. El sistema tiene como solución

$$\begin{aligned} u(t) &= 2e^{-3t} - e^{-39t} + \tfrac{1}{3}\cos(t) \\ v(t) &= -e^{-3t} + 2e^{-39t} - \tfrac{1}{3}\cos(t). \end{aligned}$$

Euler explícito usando matrices y notación obvia queda como

$$X_{n+1} = (I + hA)X_n + hF_n$$

Euler implícito usando matrices y recordando que no se debe de calcular la inversa queda como

$$X_{n+1} = (I - hA)\backslash[X_n + hF_{n+1}]$$

mientras que Crank-Nicolson es

$$X_{n+1} = (I - 0.5hA)\backslash[(I + 0.5hA)X_n + 0.5h(F_n + F_{n+1})]$$

El archivo **sistema2Dgeneral.m** es:

```
clear all;
T=5;
N=100;
x0=1;
y0=2;
h=T/N;
t=[0:h:T];
```

```
x0=4/3;
y0=2/3;
%
% Solucion exacta
%
xv=2*exp(-3*t)-exp(-39*t)+cos(t)/3;
yv=-exp(-3*t)+2*exp(-39*t)-cos(t)/3;
x=zeros(N+1,1);
y=zeros(N+1,1);
x(1)=x0;
y(1)=y0;
h=T/N;
ejes=[0 T -10 10];
%
% Matriz del sistema
%
A=[9 24;-24 -51];
%
% Matriz identidad
%
I=[1 0;0 1];
%
% Matriz de iteracion para Euler explicito
%
Me=I+h*A;
%
% Inicio computo con Euler explicito
%
v=[x(1);y(1)];
for i=1:N
    v=Me*v+h*[f1(t(i));f2(t(i))];
    x(i+1)=v(1);
    y(i+1)=v(2);
end
figure(1);
subplot(2,3,1),plot(t,x,'-o',t,xv,'-');
title([' Euler explicito, N= ',num2str(N)]);
legend('x','xv','location','Best');
%axis(ejes);
subplot(2,3,4),plot(t,y,'-o',t,yv,'-');
title([' Euler explicito, N= ',num2str(N)]);
legend('y','yv','location','Best');%axis(ejes);
%axis(ejes);
```

```
%
% Matriz de iteracion para Euler implicito
%
Mi=I-h*A;
%
% Inicio computo con Euler implicito
%
v=[x(1);y(1)];
for i=1:N
    v=Mi\(v+h*[f1(t(i+1));f2(t(i+1))]);
    x(i+1)=v(1);
    y(i+1)=v(2);
end
subplot(2,3,2),plot(t,x,'*',t,xv,'-');
title([' Euler implicito, N= ',num2str(N)]);
legend('x','xv','location','Best');
axis(ejes);
subplot(2,3,5),plot(t,y,'*',t,yv,'-');
title([' Euler implicito, N= ',num2str(N)]);
legend('y','yv','location','Best');
axis(ejes);
%
% Matriz de iteracion para Crank-Nicolson
%
MCN=I-0.5*h*A;
MCNaux=I+0.5*h*A;
%
% Inicio computo con Crank-Nicolson
%
v=[x(1);y(1)];
for i=1:N
    v=MCN\(MCNaux*v+0.5*h*[f1(t(i+1))+f1(t(i));f2(t(i+1))+f2(t(i))]);
    x(i+1)=v(1);
    y(i+1)=v(2);
end
subplot(2,3,3),plot(t,x,'+',t,xv,'-');
title([' Crank-Nicolson, N= ',num2str(N)]);
legend('x','xv','location','Best');
axis(ejes);
subplot(2,3,6),plot(t,y,'+',t,yv,'-');
title([' Crank-Nicolson, N= ',num2str(N)]);
legend('y','yv','location','Best');
axis(ejes);
```

El archivo **f1.m** es:

```
function valor=f1(tt)
valor=5*cos(tt)-sin(tt)/3;
```

y el archivo **f2.m** es:

```
function valor=f2(tt)
valor=-9*cos(tt)+sin(tt)/3;
```

4. La programación con el estilo de función para este ejemplo es

```
function Sistema2DNohomogeneo(T,N,x0,y0)

h=T/N;
t=[0:h:T];
x0=4/3;
y0=2/3;
xv=2*exp(-3*t)-exp(-39*t)+cos(t)/3;
yv=-exp(-3*t)+2*exp(-39*t)-cos(t)/3;
x=zeros(N+1,1);
y=zeros(N+1,1);
x(1)=x0;
y(1)=y0;
h=T/N;
ejes=[0 T -2 2];
%
% Matriz del sistema
%
A=[9 24;-24 -51];
%
% Matriz identidad
%
I=[1 0;0 1];
%
% Matriz de iteracion para Euler explicito
%
Me=I+h*A;
%
% Inicio computo con Euler explicito
%
v=[x(1);y(1)];
for i=1:N
```

```
    v=Me*v+h*[f1(t(i));f2(t(i))];
    x(i+1)=v(1);
    y(i+1)=v(2);
end
subplot(2,3,1),plot(t,x,t,xv);
title([' Euler explicito, N= ',num2str(N)]);
legend('x','xv','location','Best');
subplot(2,3,4),plot(t,y,t,yv);
title([' Euler explicito, N= ',num2str(N)]);
legend('y','yv','location','Best');%axis(ejes);
axis(ejes);
%
% Matriz de iteracion para Euler implicito
%
Mi=I-h*A;
%
% Inicio computo con Euler implicito
%
v=[x(1);y(1)];
for i=1:N
    v=Mi\(v+h*[f1(t(i+1));f2(t(i+1))]);
    x(i+1)=v(1);
    y(i+1)=v(2);
end
subplot(2,3,2),plot(t,x,t,xv);
title([' Euler implicito, N= ',num2str(N)]);
legend('x','xv','location','Best');
axis(ejes);
subplot(2,3,5),plot(t,y,t,yv);
title([' Euler implicito, N= ',num2str(N)]);
legend('y','yv','location','Best');
axis(ejes);
%
% Matriz de iteracion para Crank-Nicolson
%
MCN=I-0.5*h*A;
MCNaux=I+0.5*h*A;
%
% Inicio computo con Crank-Nicolson
%
v=[x(1);y(1)];
for i=1:N
    v=MCN\(MCNaux*v+0.5*h*[f1(t(i+1))+f1(t(i));f2(t(i+1))+f2(t(i))]);
```

```
    x(i+1)=v(1);
    y(i+1)=v(2);
end
subplot(2,3,3),plot(t,x,t,xv);
title([' Crank-Nicolson, N= ',num2str(N)]);
legend('x','xv','location','Best');
axis(ejes);
subplot(2,3,6),plot(t,y,t,yv);
title([' Crank-Nicolson, N= ',num2str(N)]);
legend('y','yv','location','Best');
axis(ejes);

function valor=f1(tt)
valor=5*cos(tt)-sin(tt)/3;
function valor=f2(tt)
valor=-9*cos(tt)+sin(tt)/3;
```

5. Programación de Runge-Kutta clásico de cuatro etapas y orden cuatro para el sistema de Lorenz

$$\begin{aligned} x'(t) &= -10\,(x(t) - y(t)), \\ y'(t) &= 28\,x(t) - y(t) - x(t)\,z(t), \\ z'(t) &= 2.667\,(x(t)\,y(t) - z(t)). \end{aligned}$$

```
%
% El sistema de Lorenz se calcula mediante
% Runge-Kutta de cuarto orden en el intervalo
% [0,T] con h=T/N con datos iniciales (x0,y0,z0)
%
% Se puede calcular la curva conforme se obtiene simplemente
% dibujando las componentes de los vectores calculadas en cada
% en cada paso de la interacion
 hold off;
 clear all;
 N=1000;
 T=10;
 x0=1;
 y0=2;
 z0=3;
h=T/N;
t=0:h:T;
x=zeros(1,N+1);
y=zeros(1,N+1);
```

```
z=zeros(1,N+1);
x(1)=x0;
y(1)=y0;
z(1)=z0;
%
% Aunque tengamos funciones de la forma f(x,y,z) y el argumento t
% no sea necesario, es bueno ponerlo y escribir f(t,x,y,z) para
% tener un codigo base que pueda servir para otros ejemplos.
%
for i=1:N
    k1x=fx(t(i),x(i),y(i),z(i));
    k1y=fy(t(i),x(i),y(i),z(i));
    k1z=fz(t(i),x(i),y(i),z(i));

    k2x=fx(t(i)+h/2,x(i)+h*k1x/2,y(i)+h*k1y/2,z(i)+h*k1z/2);
    k2y=fy(t(i)+h/2,x(i)+h*k1x/2,y(i)+h*k1y/2,z(i)+h*k1z/2);
    k2z=fz(t(i)+h/2,x(i)+h*k1x/2,y(i)+h*k1y/2,z(i)+h*k1z/2);

    k3x=fx(t(i)+h/2,x(i)+h*k2x/2,y(i)+h*k2y/2,z(i)+h*k2z/2);
    k3y=fy(t(i)+h/2,x(i)+h*k2x/2,y(i)+h*k2y/2,z(i)+h*k2z/2);
    k3z=fz(t(i)+h/2,x(i)+h*k2x/2,y(i)+h*k2y/2,z(i)+h*k2z/2);

    k4x=fx(t(i)+h,x(i)+h*k3x,y(i)+h*k3y,z(i)+h*k3z);
    k4y=fy(t(i)+h,x(i)+h*k3x,y(i)+h*k3y,z(i)+h*k3z);
    k4z=fz(t(i)+h,x(i)+h*k3x,y(i)+h*k3y,z(i)+h*k3z);

    x(i+1)=x(i)+h*(k1x+2*k2x+2*k3x+k4x)/6;
    y(i+1)=y(i)+h*(k1y+2*k2y+2*k3y+k4y)/6;
    z(i+1)=z(i)+h*(k1z+2*k2z+2*k3z+k4z)/6;

end
figure(1);
subplot(3,1,1),plot(t,x,t,y,t,z,'--');
title('Modelo de Lorenz')
legend('x','y','z');
hold off;
subplot(3,1,2),plot(x,y)
title('Efecto mariposa: Plano de Fases x-y')
hold off;
subplot(3,1,3),plot(x,z)
title('Efecto mariposa: Plano de Fases x-z')
hold off;
```

```
function valor=fx(tt,xx,yy,zz)
valor=-10*(xx-yy);
end

function valor=fy(tt,xx,yy,zz)
valor=-xx*zz+28*xx-yy;
end

function valor=fz(tt,xx,yy,zz)
valor=2.667*(xx*yy-zz);
end
```