MATEMÁTICA SUPERIOR APLICADA

Solución Numérica de Sistemas de Ecuaciones Diferenciales Ordinarias en Ingeniería Química

Universidad Tecnológica Nacional – Facultad Regional Rosario

Dr. Alejandro S. M. Santa Cruz

Solución de un Sistema de EDO's de 1er. Orden (I)

- Planteamos la solución del siguiente sistema de ***n*** ecuaciones diferenciales de 1er. Orden:

$$x'_1(t) = \frac{dx_1}{dt} = f_1(t, x_1, x_2, ..., x_n)$$

$$x'_2(t) = \frac{dx_2}{dt} = f_2(t, x_1, x_2, ..., x_n)$$

$$\vdots$$

$$x'_n(t) = \frac{dx_n}{dt} = f_n(t, x_1, x_2, ..., x_n)$$

Condiciones Iniciales

$$x_1(t_0) = x_1^{(0)}$$

$$x_2(t_0) = x_2^{(0)}$$

$$x_3(t_0) = x_3^{(0)}$$

$$\vdots$$

$$x_n(t_0) = x_n^{(0)}$$

Solución de un Sistema de EDO's de 1er. Orden (II)

- Dado que para obtener la solución utilizamos un algoritmo numérico, cada una de las ecuaciones del sistema puede tratarse en forma independiente.
- Más aún, para cada ecuación se podría utilizar una estrategia de resolución diferente ya que se aplican en paralelo y simultáneamente en cada etapa de integración.
- Esta afirmación vale tanto para los modelos explícitos como los implícitos.

Solución de un Sistema de EDO's de 1er. Orden (III)

- **Criterio práctico:** Utilizar el mismo algoritmo para todo el sistema.
- **Estimación de una cota para el error:** Trabajo muy tedioso cuando no imposible.
- **Métodos multipaso:** Se debe recurrir a esquemas iterativos con sistemas de ecuaciones no lineales (ya sean algebraicas como no algebraicas) con los problemas que ello implica en la convergencia.

Solución de un Sistema de EDO's de 1er. Orden (IV)

- **Métodos multipaso:**
  - **Filosofía: Es la misma que antes pero con la diferencia que cada etapa de cálculo implica un álgebra matricial para resolver SENL.**
  - **Otros aspectos que complican el cálculo:**
    - **Influencia de los diferentes pasos en la determinación del error.**
    - **Estabilidad del método.**

Ejemplo 02

- Resolvemos el sgte. sistema:

$$\frac{dx_1}{dt} = -x_1 - x_2$$

$$\frac{dx_2}{dt} = x_1 - 2x_2$$

Condiciones iniciales

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

- Idénticamente se pueden escribir como:

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -1 & -1 \\ 1 & -2 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Ejemplo 02
Método Explícito de Euler para Resolver Sistemas de EDOs

Archivo de Comandos de MATLAB:

Ejemplo_02.m: Archivo de comandos que llama a la función **EulerCoupled.m**.

Método:

EulerCoupled.m: Función que implementa el algoritmo explícito de Euler para resolver sistemas de EDOs.

Funciones:

TestFunction2.m: Función que implementa las derivadas del sistema de EDOs (llamada por **EulerCoupled.m**).

Archivo de comandos de MATLAB:

```
clear all
clc
echo on
%Ejemplo_02
Start = 0;
Finish = 10;
Nsteps = 100;
InitialValue = [1;1];
%numerical solution
[x,t] = EulerCoupled(@TestFunction2,InitialValue,Start,Finish,Nsteps);
plot(t,x)
title('System of ODEs: Explicit Euler Method')
xlabel('t')
ylabel('x')
legend('x1','x2')
pause
disp('Numerical solution : '),[t x]
```

```
function [x,t] = EulerCoupled(MyFunc,InitialValues,Start,Finish,Nsteps)
% solves the initial value problem dx/dt = f(x)
% Uses the Euler method
% Myfunc = Handle to the function which calculates F(x)
% InitialValues = starting values (a column vector)
% Start = start time
% finish = end time
% Nsteps = number of steps to march forward
x(:,1) = InitialValues;
t(1) = Start;
%calculate the time step
dt = (Finish-Start)/Nsteps;
for i = 1:Nsteps
    %calculate the gradient using the current solution (time = t)
    %the function should now return a vector of gradients
    F = feval(MyFunc, x(:,i),t(i));
   %calculate the new value of x and t
    t(i+1) = dt + t(i);
   x(:,i+1) = F*dt + x(:,i);
end
t = t';
x = x';
return
```

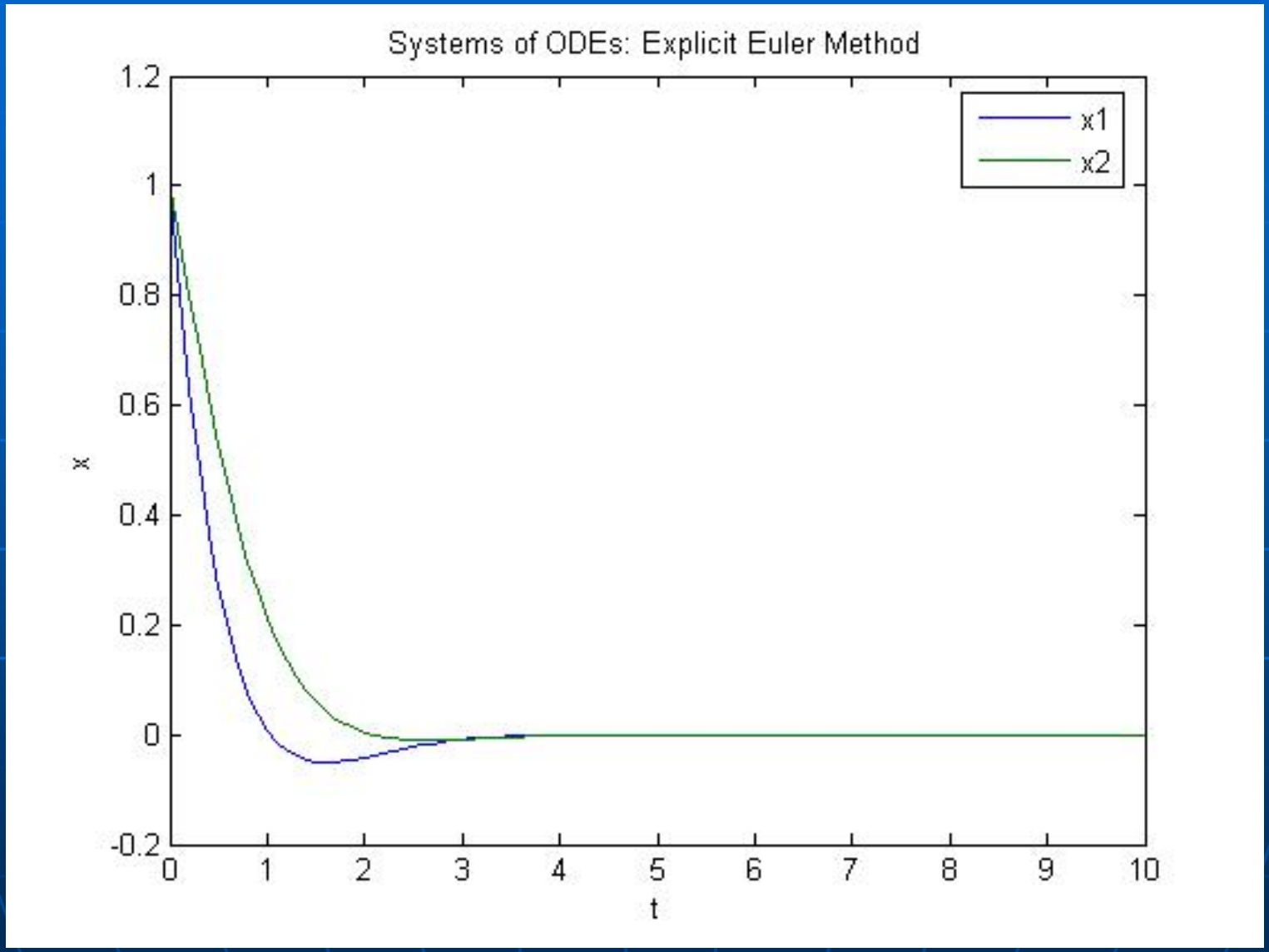
Necesitamos escribir la *function* que me devuelva las derivadas de las funciones:

```
function [dx_dt]= TestFunction2(x,t)
%a function which returns a rate of change vector
dx_dt(1) = -1*x(1)-1*x(2);
dx_dt(2) = 1*x(1) -2*x(2);
%note: I could have just multiplied x by a matrix
%change it to a column vector
dx_dt= dx_dt';
return
```

Ejemplo 02
Método Explícito de Euler
Para Resolver un Sistema de EDOs

Ejecución del Programa en MATLAB

```
>> Ejemplo_02
Start = 0;
Finish = 10;
Nsteps = 100;
InitialValue = [1;1];
%numerical solution
[x,t] = EulerCoupled(@TestFunction2,InitialValue,Start,Finish,Nsteps);
plot(t,x)
title('System of ODEs: Explicit Euler Method')
xlabel('t')
ylabel('x')
legend('x1','x2')
pause
```


Systems of ODEs: Explicit Euler Method
x1
x2
x
t
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
0
1
2
3
4
5
6
7
8
9
10

Ejemplo 02
Método Explícito de Euler
Para Resolver un Sistema de EDOs

Ejecución del Programa en MATLAB (continuación):

```
disp('Numerical solution : '),[t x]
Numerical solution :
ans =
         0    1.0000    1.0000
    1.0000    0.0076    0.2110
    2.0000   -0.0413    0.0032
    3.0000   -0.0094   -0.0084
    4.0000   -0.0003   -0.0020
    5.0000    0.0003   -0.0001
    6.0000    0.0001    0.0001
    7.0000    0.0000    0.0000
    8.0000   -0.0000    0.0000
    9.0000   -0.0000   -0.0000
   10.0000   -0.0000   -0.0000
```

Método Implícito de Euler para Resolver Sistemas de EDOs (I)

- Cuando tenemos un método implícito tenemos que resolver un SENL para obtener el valor actualizado de la función vectorial $\underline{x}(t)$.
- Para un sistema gobernado por:

$$\frac{d\,\underline{x}}{dt} = \underline{f}\left(\underline{x}\right)$$

con el Método Implícito de Euler, el valor de $\underline{x}_{i+1}$, viene dado por la ecuación implícita:

$$g\left(\underline{x}_{i+1}\right) = \underline{x}_i + h\,\underline{f}\left(\underline{x}_{i+1}\right) - \underline{x}_{i+1} = 0$$

- Podríamos resolver esta ecuación utilizando el método de Newton para cada paso del tiempo para la incógnita $\underline{x}_{i+1}$:

$$\underline{\underline{J}}_g \cdot \left(\underline{x}^{new} - \underline{x}^{old}\right) = -g\left(\underline{x}^{old}\right)$$

Método Implícito de Euler para Resolver Sistemas de EDOs (II)

- J_g es el Jacobiano de la función vectorial $\underline{g(x)}$ *de* donde resulta que:

$$\underline{\underline{J}}_g = h\underline{\underline{J}}_f - \underline{\underline{I}}$$

donde I es la matriz identidad, J_f es el Jacobiano de la función f.

- Por consiguiente, si arrancamos con una estimación de $\underline{x}^{old}(t_{i+1})$ podemos actualizar esa estimación resolviendo:

$$\left(h\underline{\underline{J}}_f - \underline{\underline{I}}\right).\left(\underline{x}_{i+1}^{new} - \underline{x}_{i+1}^{old}\right) = -\left[\underline{x}_i + h\underline{f}\left(\underline{x}_{i+1}^{old}\right) - \underline{x}_{i+1}^{old}\right]$$

que representa un sistema de ecuaciones lineales.

Método Implícito de Euler para Resolver Sistemas de EDOs (III)

- Una estimación inicial para $\underline{x}^{old}(t_{i+1})$ sería utilizar $\underline{x}_i$ lo cual significa que para la primera iteración obtendríamos:

$$\left(h\underline{\underline{J}}_f - \underline{\underline{I}}\right).\left(\underline{x}_{i+1}^{new} - \underline{x}_i\right) = -h\underline{f}\left(\underline{x}_i\right)$$

- Si el paso temporal es pequeño podríamos esperar una sola iteración del método de Newton.
- Utilizando sólo una iteración del método de Newton para resolver la relación implícita resulta en un esquema dependiente del tiempo denominado Método de Euler Implícito Linealizado (nótese que para nuestro análisis de estabilidad, siempre supondremos que el método de Newton converge).

Método Implícito de Euler para Resolver Sistemas de EDOs (IV)

- Un método implícito requerirá resolver un conjunto no lineal (o lineal) de ecuaciones en cada paso temporal.
- Esto tiene un alto costo computacional. Usualmente, el esquema de cálculo requerirá que se le suministre la función, devolviendo el Jacobiano de $f(t, \underline{x})$ o bien calculará el Jacobiano en forma numérica.

Ejemplo 02
Método Implícito de Euler para Resolver Sistemas de EDOs

Archivo de Comando de MATLAB:

Ejemplo_02.m: Archivo de comandos que llama a la función **EulerImplicit.m**.

Método:

EulerImplicit.m: Función que implementa el algoritmo implícito de Euler para resolver sistemas de EDOs.

Funciones:

TestFunction2.m: Función que implementa las derivadas del sistema de EDOs (llamada por **EulerImplicit.m**).

funToSolve.m: Función que calcula el residuo en el esquema implícito (llamada por **EulerImplicit.m**).

fsolve.m: Función que resuelve el sistema implícito de ecuaciones para obtener el valor actualizado de $\underline{x}$.

Archivo de comandos de MATLAB:

```
clear all
clc
echo on
%Ejemplo_02
Start = 0;
Finish = 10;
Nsteps = 100;
InitialValue = [1;1];
%numerical solution
[x,t] = EulerImplicit(@TestFunction2,InitialValue,Start,Finish,Nsteps);
plot(t,x)
title('System of ODEs: Implicit Euler Method')
xlabel('t')
ylabel('x')
legend('x1','x2')
pause
disp('Numerical solution : '),[t x]
```

```
function [x,t] = EulerImplicit(MyFunc,InitialValue,Start,Finish,Nsteps)
% solves the initial value problem dx/dt = f(x)
% Uses the Euler (implicit) method
% Myfunc = Handle to the function which calculates F(x)
% InitialValue = starting value
% Start = start time
% finish = end time
% Nsteps = number of steps to march forward
x(:,1) = InitialValue;
t(1) = Start;
%calculate the time step
dt = (Finish-Start)/Nsteps;
for i = 1:Nsteps
    %solve the implicit equation to get the updated value of x
    %calculate the new value of x and t
    t(i+1) = dt + t(i);
    x(:,i+1) = fsolve(@funToSolve,x(:,i),[],x(:,i),t(i+1),MyFunc,dt);
end
t = t';
x = x';
return
```

Funciones

```
function [dx_dt]= TestFunction2(x,t)
%a function which returns a rate of change vector
dx_dt(1) = -1*x(1)-1*x(2);
dx_dt(2) = 1*x(1) -2*x(2);
%note: I could have just multiplied x by a matrix
%change it to a column vector
dx_dt= dx_dt';
return
```

```
function residual = funToSolve(x,xo,t,MyFunc,dt)
residual=xo+feval(MyFunc,x,t)*dt-x;
return
```

Ejemplo 02
Método Implícito de Euler
Para Resolver un Sistema de EDOs

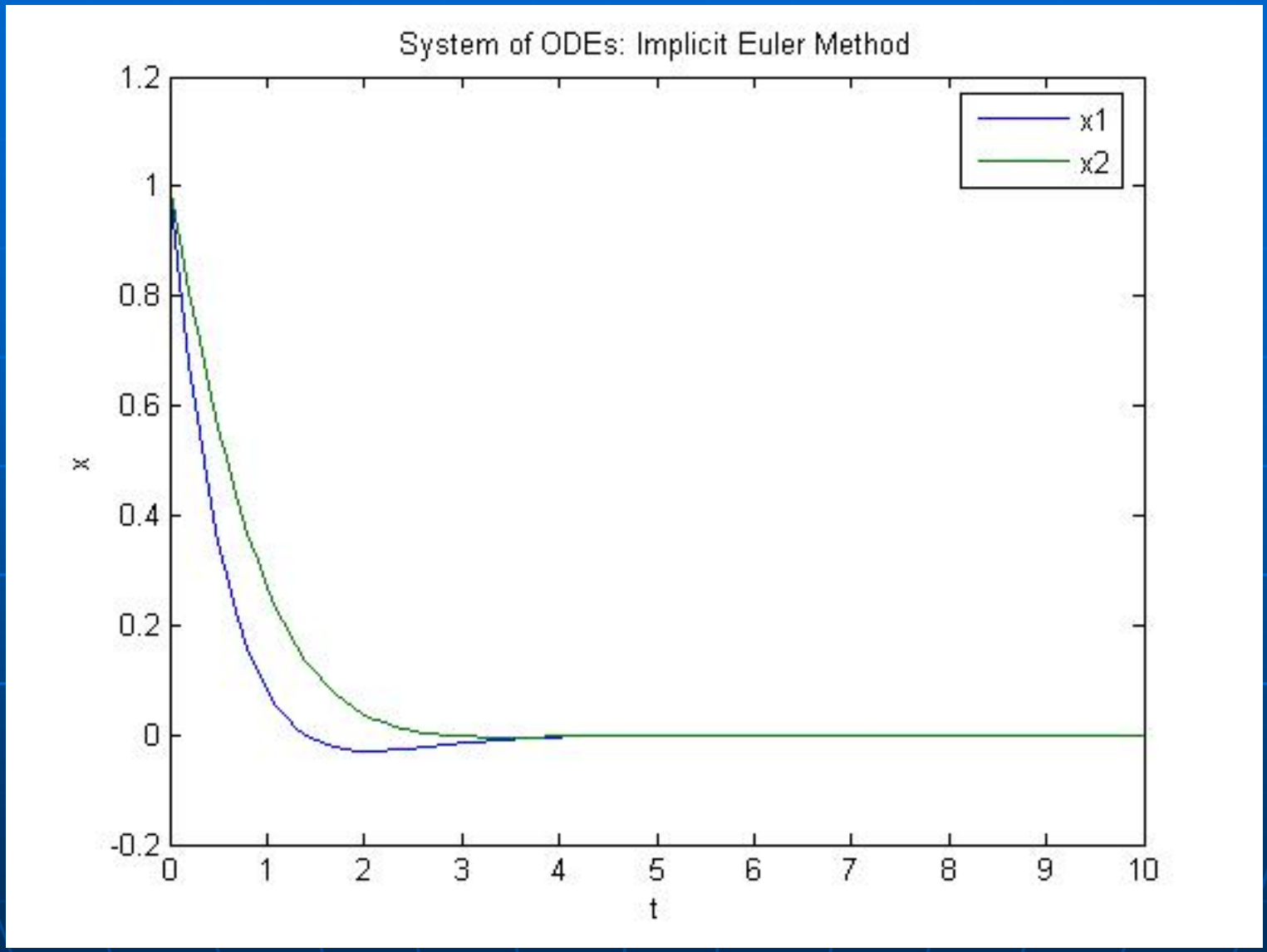
Ejecución del Programa en MATLAB:

```
>> Ejemplo_02
%Ejemplo_02
Start = 0;
Finish = 10;
Nsteps = 100;
InitialValue = [1;1];
%numerical solution
[x,t] = EulerImplicit(@TestFunction2,InitialValue,Start,Finish,Nsteps);
Optimization terminated: first-order optimality is less than options.TolFun.
```

Ejemplo 02
Método Implícito de Euler
Para Resolver un Sistema de EDOs

Ejecución del Programa en MATLAB (continuación):

```
plot(t,x)
title('System of ODEs: Implicit Euler Method')
xlabel('t')
ylabel('x')
legend('x1','x2')
pause
```


System of ODEs: Implicit Euler Method
x1
x2
x
t
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
0
1
2
3
4
5
6
7
8
9
10

Ejemplo 02
Método Implícito de Euler
Para Resolver un Sistema de EDOs

Ejecución del Programa en MATLAB (continuación):

```
disp('Numerical solution : '),[t x]
Numerical solution :
ans =
         0    1.0000    1.0000
    1.0000    0.0808    0.2703
    2.0000   -0.0294    0.0372
    3.0000   -0.0150   -0.0026
    4.0000   -0.0036   -0.0030
    5.0000   -0.0004   -0.0009
    6.0000    0.0001   -0.0001
    7.0000    0.0000    0.0000
    8.0000    0.0000    0.0000
    9.0000    0.0000    0.0000
   10.0000    0.0000    0.0000
```

Consideraciones Acerca del Uso del Método Implícito de Euler para Resolver Sistemas de EDOs (I)

- Para resolver grandes sistemas de EDOs, la ***EulerImplicit*** no es una rutina muy eficiente. Esto se debe a que a ***fsolve*** no le hemos suministrado el ***Jacobiano*** de la transformación, razón por la cual debe evaluarlo analíticamente.
- Para un sistema de 100 EDOs el Jacobiano será una matriz de 100*100 elementos, que deberá evaluarse en cada iteración de la solución del sistema no lineal de ecuaciones para cada paso del tiempo.
- Podríamos incrementar la velocidad:
  - Suministrando a ***fsolve*** una función para que evalúe el Jacobiano.
  - Suministrando a ***fsolve*** una ***matriz rala*** (***sparse***) que nos diga cuando el Jacobiano es no nulo (no se desperdicia tiempo en calcular elementos nulos). Esto se denomina ***sparsity pattern***, y es una matriz de ceros y unos.

Consideraciones Acerca del Uso del Método Implícito de Euler para Resolver Sistemas de EDOs (II)

- ***fsolve*** es lo suficientemente sagaz; optimizará el cálculo cuando sea necesario recalcular el Jacobiano.
- El MATLAB suministra resolvedores (por ej., ***ODE15s***) que nos permiten hacer todo lo que hicimos previamente. Son programas que hacen cosas inteligentes como optimizar cuando recalculan el jacobiano y varían el tamaño del paso de integración para optimizar la exactitud de la solución.
- Los solvers del MATLAB son de orden superior (los solvers de Euler son de primer orden de exactitud (O(h); ODE15s es hasta de 5to. orden de exactitud).
- **NO ESCRIBA SU SOLVER. USE MATLAB!!!**

Estabilidad de las Soluciones Numéricas (I)

Estabilidad Inherente de un Sistema de EDOs

- Supongamos que tenemos el sistema de ODEs lineales, como el que ya vimos:

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -1 & -1 \\ 1 & -2 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

- O en forma más general:

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \underline{\underline{M}} . \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

- Existe un estado estacionario en $(0,0)$. Frecuentemente, es importante saber si el sistema decaerá o divergerá desde dicho estado, si es perturbado. Para sistema lineales, podemos obtener una solución analítica. Podemos factorizar M de la siguiente forma:

$$\underline{\underline{M}} = \underline{\underline{U}}^{-1} . \underline{\underline{\Lambda}} . \underline{\underline{U}}$$

donde Λ es la matriz de los autovalores de M, y U la matriz de los autovectores.

Estabilidad de las Soluciones Numéricas (II)

Estabilidad Inherente de un Sistema de EDOs

- Si reemplazamos ***M*** por su expresión factorizada, obtenemos:

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \underline{\underline{U}}^{-1} \cdot \underline{\underline{\Lambda}} \cdot \underline{\underline{U}} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

- Premultiplicando por ***U*** se obtiene:

$$\frac{d}{dt}\left(\underline{\underline{U}} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right) = \underline{\underline{\Lambda}} \cdot \left(\underline{\underline{U}} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right)$$

- Podemos resolver el sistema en términos de un nuevo conjunto de variables:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \underline{\underline{U}} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$\frac{d}{dt}\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \underline{\underline{\Lambda}} \cdot \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

Estabilidad de las Soluciones Numéricas (III)

Estabilidad Inherente de un Sistema de EDOs

- Dado que Λ es diagonal, entonces:

$$\frac{d}{dt}\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

- Tenemos ahora un sistema de EDOs desacoplado:

$$\frac{dy_1}{dt} = \lambda_1 y_1 \Rightarrow y_1 = \alpha\, exp(\lambda_1 t)$$

$$\frac{dy_2}{dt} = \lambda_2 y_2 \Rightarrow y_2 = \beta\, exp(\lambda_2 t)$$

- O, en notación vectorial:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \alpha\, exp(\lambda_1 t) \\ \beta\, exp(\lambda_2 t) \end{bmatrix}$$

Estabilidad de las Soluciones Numéricas (IV)

Estabilidad Inherente de un Sistema de EDOs

- Para encontrar la solución hacemos:

$$\underline{\underline{U}}^{-1} \cdot \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

- Finalmente obtenemos:

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \underline{\underline{U}}^{-1} \cdot \begin{bmatrix} \alpha\, exp(\lambda_1 t) \\ \beta\, exp(\lambda_2 t) \end{bmatrix}$$

donde U es la matriz de los autovectores de M y λ_1 y λ_2 son los correspondientes autovalores.

Estabilidad de las Soluciones Numéricas (V)

Estabilidad Inherente de un Sistema de EDOs

- Los autovalores de M nos dicen cómo el sistema se aproxima al estado estacionario $(0,0)$, y si el sistema volverá a éste si es perturbado en una pequeña cantidad.
- Esta clase de estabilidad no tiene nada que ver con la solución numérica, sino que refleja la naturaleza del sistema.
- En el caso general los autovalores pueden ser complejos (por ejemplo: $\lambda = Re + Im\ j$), pudiendo expandir los términos exponenciales en la solución:

$$exp(\lambda_i t) = exp(Re_i t)\ exp(Im_i j\ t) = exp(Re_i t)\ [cos(Im_i t) + j\ sen\ (Im_i t)]$$

- **Si los autovalores tienen una parte imaginaria la solución oscilará! siendo la parte real la que determinará si la solución se aproximará al estado estacionario o divergerá.**

Estabilidad de las Soluciones Numéricas (VI)

Estabilidad Inherente de un Sistema de EDOs

- **Para nuestro sistema:**

$$\underline{\underline{M}} = \begin{bmatrix} -1 & -1 \\ 1 & -2 \end{bmatrix}$$

y los autovalores son:

$$\lambda_1 = -1.5 + j\frac{\sqrt{3}}{2} \quad ; \quad \lambda_2 = -1.5 - j\frac{\sqrt{3}}{2}$$

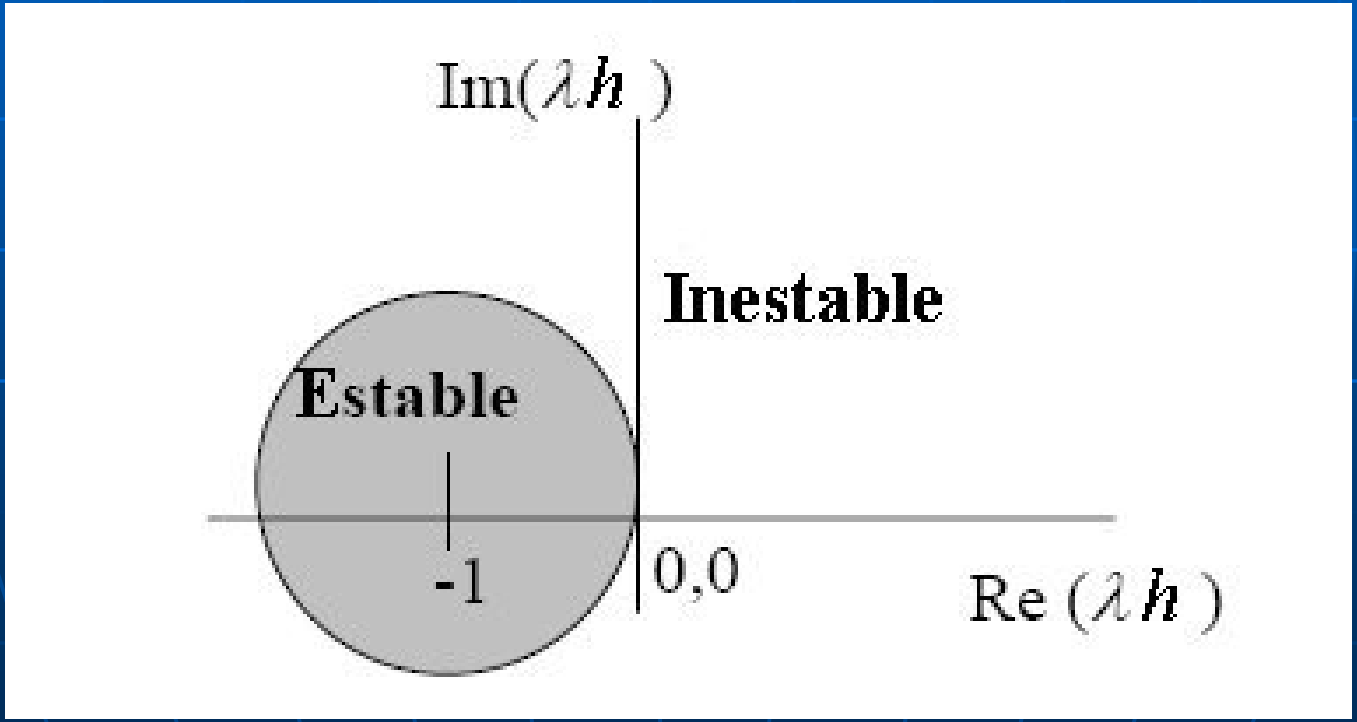
- **Ambas partes reales son negativas, por consiguiente la solución oscilará, pero decaerá al estado estacionario.**

Estabilidad de las Soluciones Numéricas para Sistemas Lineales (I)

- Cuando resolvimos una EDO simple vimos que era el gradiente local, λ, quien determinaba la estabilidad inherente de la solución numérica.
- Ahora tenemos que encontrar un criterio similar para EDOs acopladas.
- El objetivo es reducir las ecuaciones acopladas a un conjunto de ecuaciones desacopladas de manera de reutilizar el análisis de estabilidad que efectuamos para las ecuaciones diferenciales simples.
- El análisis previo nos había permitido desacoplar las ecuaciones mediante una transformación de variables del sistema.
- Son los autovalores de la matriz M, los que determinan la estabilidad numérica de la solución.
- Cada autovalor es el recíproco de un tiempo característico y (por ej., para el Método Explícito de Euler) pone un límite al paso de tiempo más grade que se puede utilizar para resolver las ecuaciones.

Estabilidad de las Soluciones Numéricas para Sistemas Lineales (II)

- Habíamos visto cuando utilizábamos el Método Explícito de Euler que la solución numérica era estable si $|1+\lambda h| < 1$. Esta situación se puede representar por una región en el plano complejo determinada por la condición: $|(\lambda h)-(-1)| < 1$



Estabilidad de las Soluciones Numéricas para Sistemas Lineales (III)

- Los autovalores de nuestro ejemplo eran:

$$\lambda_1 = -1.5 + j\frac{\sqrt{3}}{2} \quad ; \quad \lambda_2 = -1.5 - j\frac{\sqrt{3}}{2}$$

- Podemos elegir un paso temporal que nos dará una solución numérica estable para el Método Explícito de Euler:

$$\left|\left(-1.5 + 0.866\, j\right)h - \left(-1\right)\right| < 1$$

$$\left(1 - 1.5h\right)^2 + \frac{3}{4}h^2 < 1$$

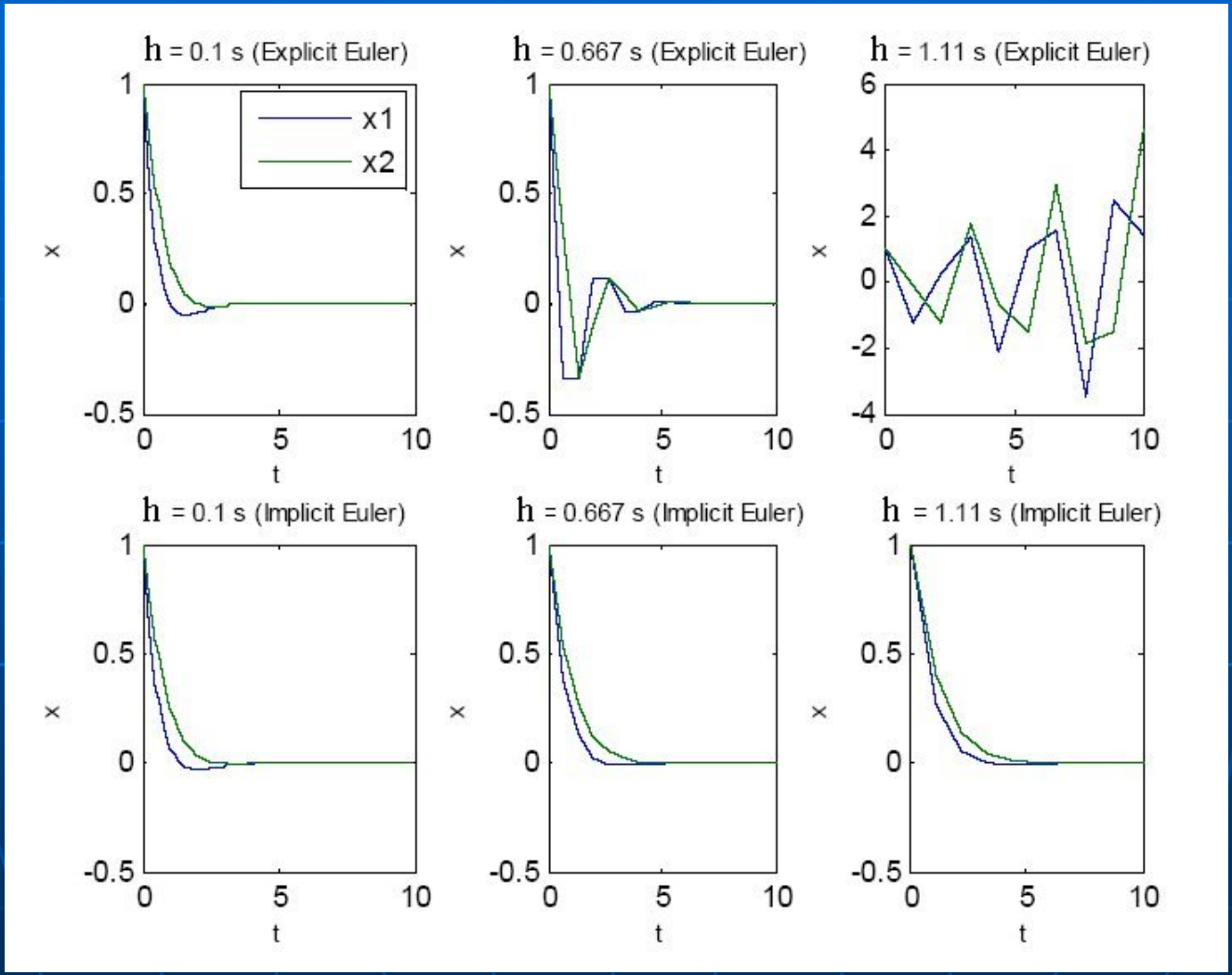
$$1 + \left(1.5h\right)^2 - 3h + \frac{3}{4}h^2 < 1$$

$$3h^2 - 3h + 1 < 1$$

$$h\left(h - 1\right) < 0$$

$$h < 1$$

Solución numérica del sistema acoplado de EDOs utilizando el método Explícito de Euler para tres pasos diferentes de tiempo. También se muestra la solución cuando se utiliza el método implícito de Euler.



Extensión del Análisis de Estabilidad a Sistemas No Lineales (I)

- Extendemos nuestro análisis a sistemas de ecuaciones diferenciales no lineales:

$$x'_1(t) = \frac{dx_1}{dt} = f_1(t, x_1, x_2, ..., x_n)$$
$$x'_2(t) = \frac{dx_2}{dt} = f_2(t, x_1, x_2, ..., x_n)$$
$$\vdots$$
$$x'_n(t) = \frac{dx_n}{dt} = f_n(t, x_1, x_2, ..., x_n)$$

- Podemos linealizar el lado derecho estas ecuaciones (al menos localmente):

$$\begin{bmatrix} f_1(\underline{x}^* + d\underline{x}) \\ f_2(\underline{x}^* + d\underline{x}) \\ \vdots \end{bmatrix} = \begin{bmatrix} f_1(\underline{x}^*) \\ f_2(\underline{x}^*) \\ \vdots \end{bmatrix} \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} & \cdots \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} & \cdots \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} & \cdots \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} + \begin{bmatrix} dx_1 \\ dx_2 \\ dx_3 \\ \vdots \end{bmatrix}$$

Extensión del Análisis de Estabilidad a Sistemas No Lineales (II)

- La forma linealizada adopta la siguiente forma:

$$\frac{d\left(\underline{x}-\underline{x}^{*}\right)}{dt} \approx \underline{f}\left(\underline{x}^{*}\right)+\underline{\underline{J}}.\left(\underline{x}-\underline{x}^{*}\right)$$

- Si comparamos esta expresión con la expresión para el sistema de ecuaciones diferenciales lineales, observamos que J es el equivalente de M.
- Son los autovalores del Jacobiano los que determinan la estabilidad del estado estacionario y los que también determinan la estabilidad de cualquier esquema numérico utilizado para resolver sistemas de ecuaciones diferenciales no lineales.

Sistemas Stiff y Elección del Paso Temporal (I)

- Los autovalores de la matriz Jacobiana nos hablan de las velocidades de cambio características del sistema (son la inversa de las constantes de tiempo). Se pueden plantear dos situaciones:
  1) **Todos los autovalores tienen magnitudes similares**. En este caso necesitamos elegir un paso de tiempo suficientemente pequeño para lograr suficiente exactitud. Generalmente, tendrá que ser más pequeño que el mínimo paso de tiempo que necesitamos para obtener una solución estable. En este caso, los esquemas explícitos de Euler resultan atractivos.
  2) **Los autovalores son muy diferentes**. En este caso debemos hacer la simulación suficientemente prolongada en el tiempo para apreciar la dinámica del sistema. El período de tiempo sobre el cual necesitamos integrar está determinado por el proceso más lento (autovalor más pequeño). Sin embargo, el máximo paso temporal que podemos usar está determinado por el autovalor de mayor magnitud. Si usamos un solver explícito, entonces debemos utilizar pasos muy pequeños sobre largos tiempos de simulación. Esto es muy ineficiente.

Sistemas Stiff y Elección del Paso Temporal (II)

- Un sistema es ***stiff*** si el número de condición del Jacobiano definido como la relación del autovalor más grande al más pequeño es mayor que 1.
- La solución a este problema es utilizar un ***solver*** implícito como el esquema implícito de Euler que vimos anteriormente.