MATEMÁTICA SUPERIOR APLICADA

Utilización de Resolvedores de MATLAB para Ecuaciones Diferenciales Ordinarias

Universidad Tecnológica Nacional – Facultad Regional Rosario

Dr. Alejandro S. M. Santa Cruz

Solvers de MATLAB (I)

- Los solvers para EDOs de MATLAB se han escrito para resolver problemas de la forma:

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

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

$$\vdots$$

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

o, en forma compacta:

$$\frac{d\underline{x}}{dt} = \underline{f}(t, x_1, x_2, ..., x_n) = \underline{f}(t, \underline{x})$$

Solvers de MATLAB (II)

- Se accede a los solvers de MATLAB mediante el llamado de funciones (*functions*) del tipo:

Opcional

[T,X] = ode(@f, timespan, X0, *options, P1, P2, P3*)**

f	**Nombre de odefile archivo que describe la ecuación diferencial a resolver.**
tspan	**Vector que especifica el intervalo de integración [t0 tfinal]. Para obtener soluciones en tiempos específicos se usa: tspan = [t0,t1, ..., tfinal].**
X0	**Vector que describe las condiciones iniciales**
options	**Argumento opcional de integración creado usando la función *odeset*.**
T,X	**Matriz solución *X*, donde cada columna corresponde al tiempo retornado en el vector columna *T*.**
P1, P2, P3, …	**Parámetros adicionales que serán pasados a *@f***

Solvers de MATLAB (III)

Sintaxis:

- **options = odeset('name1',value1,'name2',value2,...):** Crea una estructura en la opciones de integración donde ***name1***, ***name2*** representan los nombres de las propiedades seguidos de los valores a especificar. La propiedades no especificadas se establecen con la matriz vacía **[ ]**.
- **odeset:** Escribiendo esta sentencia en la línea de comandos de MATLAB, se muestran en pantalla todas las propiedades y sus posibles valores. Las propiedades disponibles dependen del método de resolución. Algunas propiedades importantes se detallan a continuación:

Solvers de MATLAB (IV)

Propiedad	Valor	Descripción
RelTol	Escalar positivo (1e-3)	Establece la tolerancia de error relativo que se aplica a todos las componentes del vector solución.
AbsTol	Escalar positivo (1e-6)	Tolerancia de error absoluto.
Refine	Entero positivo	Incrementa el número de puntos de salida por un factor de n. El valor por defecto es 1 salvo cuando se utiliza ode45 donde Refine es 4.
MaxStep	Escalar positivo	Establece el límite superior en la magnitud del tamaño de paso utilizado.
InitialStep	Escalar positivo	Establece el tamaño de paso inicial. Si es demasiado grande y afecta el error cometido, el programa usa un tamaño de paso más pequeño.

Solvers de MATLAB (V)

- **La function *f* debe tener la siguiente forma:**

```
function [dx_dt] = f(t, x, P1, P2, P3...)

dx_dt = ......

return
```

CUIDADO!

ESTA ES UNA SINTAXIS LIGERAMENTE DIFERENTE A AQUELLA UTILIZADA EN LOS SOLVERS QUE IMPLEMENTAN EL MÉTODO DE EULER COMO VIMOS EN LAS PRESENTACIONES PREVIAS.

Sentencia	Tipo de método	Tipo de problema	Orden de exactitud	Cuando se usa
ode45	Explícito	No stiff	4to. Orden; exactitud media	En general, *ode45* es el mejor método para aplicar como primer intento para la resolución de muchos problemas.
ode23	Explícito	No stiff	2do./3er. Orden; exactitud baja	Ante bajas tolerancias de error o resolución de problemas moderadamente stiff.
ode113	Explícito	No stiff	13er. Orden; Baja a alta	Ante tolerancias estrictas de error y en la resolución de un archivo odefile computacionalmente intenso.
ode15s	Implícito	Stiff	De 1er. a 5to. Orden; exactitud baja a media	Si *ode45* es lento (sistema stiff).
ode23s	Implícito	Stiff	Baja exactitud, pero puede ser más estable que *ode15s*	
ode23tb	Implícito	Stiff	Baja exactitud, pero puede ser más estable que *ode15s*	

Algoritmos Utilizados (I)

Varían de acuerdo al orden de exactitud y al tipo de sistema (stiff o no stiff).

- **ode45:** Se basa en una fórmula explícita del método de Runge-Kutta (4,5), realizado por Dormand-Prince. Es un método de un solo paso, esto es, para determinar $\underline{x}(t_{i+1})$, es necesario conocer solamente la solución en el tiempo inmediatamente anterior, $\underline{x}(t_i)$.
- **ode23:** Es una implementación del método explícito de Runke-Kutta (2,3) realizado por Bogacki y Shampine. Puede ser más eficiente que ode45 para tolerancias de error bajas y en presencia de problemas stiff moderados.
- **ode113:** Método de orden variable indicado por Adams-Bashforth-Moulton. Es un método de multipaso; normalmente necesita la solución de diversos puntos precedentes para computar la solución actual.

Algoritmos Utilizados (II)

Los algoritmos anteriores se destinan para resolver sistemas no stiff. Si ellos aparecen excesivamente lentos, se utilizan otros métodos como ser:

- **ode15s: Es un método de orden variable basado en la fórmula de diferenciación numérica NDFs. Opcionalmente, usa la fórmula de diferencias hacia atrás, BDFs (backward differentiation formula), también conocida como método de Gear.**
- **ode23s: Se basa en una fórmula modificada de Rosenbrock de orden 2. Dado que es un solver de paso simple puede ser más eficiente que la ode15s para tolerancias más altas. Puede resolver algunos tipos de problemas stiff para los cuales la ode15s no es efectiva.**
- **ode23t: Implementa la regla del trapecio utilizando una *interpolación libre*. Utilizar este solver si el problema es solo moderadamente stiff y se requiere una solución sin amortiguamiento numérico.**

Algoritmos Utilizados (III)

- **ode23tb:** Es una implementación de la fórmula TR-BDF2, fórmula implícita de Runge-Kutta con una primera etapa que es una regla trapezoidal y una segunda etapa que es una fórmula de diferenciación hacia atrás de orden 2. Por construcción, se utiliza la misma matriz de iteración en la evaluación de ambas etapas. Como en la ode23s, este solver puede ser más eficiente que la ode15s para tolerancias más altas.

Consideraciones sobre los Algoritmos Utilizados (I)

- Para un problema stiff las soluciones pueden cambiar sobre una escala de tiempo que es muy corta comparada con el intervalo de integración, pero la solución de interés cambia sobre escalas de tiempo mucho más largas.
- Los métodos no diseñados para problemas stiff son ineficientes sobre intervalos donde la solución cambia lentamente debido a que usan pasos temporales suficientemente pequeños como para resolver el cambio más rápido posible.
- Las **ode15s** y **ode23s** generan numéricamente las matrices Jacobianas.

Consideraciones sobre los Algoritmos Utilizados (II)

- En mayor medida utilizaremos las ***ode45*** y ***ode15s*** (la 's' significa que utiliza un método implícito).
- Por lo general, los solvers de MATLAB serán mejores de los que podría programar Ud. mismo.
- Son capaces de estimar el error en la solución en cada etapa temporal y decidir si el paso es demasiado grande (error alto) o demasiado pequeño (ineficiente).
- Es más importante que Ud. entienda conceptos tales como ***esquemas implícitos, esquemas explícitos*** y ***orden de exactitud del algoritmo*** en lugar de saber exactamente que hace internamente la rutina de cálculo.

Consideraciones sobre los Algoritmos Utilizados (III)

- **ode45** (método explícito de Runge-Kutta) es eficiente, pero se vuelve inestable con sistemas stiff. Este hecho se manifestará con el solver tratando de adoptar pasos temporales cada vez más cortos para compensar. Consecuentemente, al algoritmo le tomará mucho tiempo alcanzar la solución o el paso temporal se reducirá al punto donde la precisión de la máquina provoque la falla del algoritmo.
- **ode15s** debería utilizarse sólo para problemas stiff. Debido a que es un esquema implícito, tendrá que resolver conjuntos de ecuaciones (posiblemente grande) en cada paso temporal.

Utilización de Solvers de MATLAB

Ejemplo 01

Utilizar la ode45 para resolver el siguiente sistema acoplado de EDOs :

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

Condiciones iniciales

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

Ejemplo 01

Archivo de Comandos de MATLAB:

Ejemplo_01_ode45.m: Archivo de comandos que llama al solver **ode45.m**.

Función:

TestFunction.m: Función que implementa en forma matricial las derivadas del sistema de EDOs.

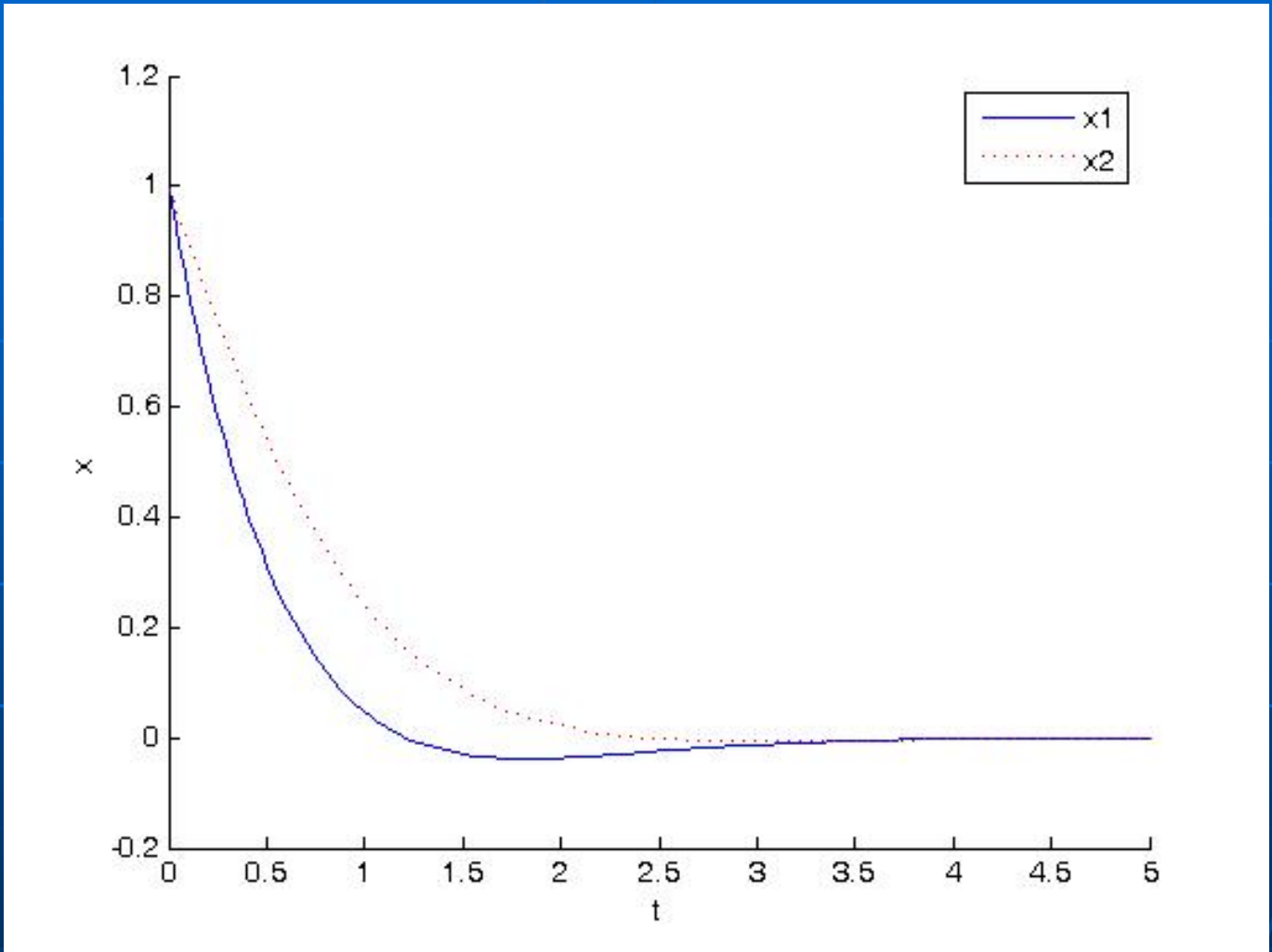
Archivo de Comandos

```
clear all
clc
% Example 01
% A simple example to solve ODE's
% Uses ode45 to solve
% dx_dt(1) = -1*x(1)-1*x(2)
% dx_dt(2) = 1*x(1) -2*x(2)
% set an error
options=odeset('RelTol',1e-6);
% initial conditions
Xo = [1;1];
%timespan
tspan = [0,5];
%call the solver
[t,X] = ode45(@TestFunction,tspan,Xo,options);
%plot the results
figure
hold on
plot(t,X(:,1));plot(t,X(:,2),'r:')
legend('x1','x2');
ylabel('x');
xlabel('t')
```

Función

```
function [dx_dt]= TestFunction(t,x)
%A function which returns a rate of change vector
M = [-1 -1; 1 -2];
dx_dt = M*x;
return
```

Gráfica de la Solución



Solución del sistema de EDOs generada por ode45.

Consideraciones Acerca del Uso del Solver ode45

- No hemos especificado un tamaño de paso. **ode45** utiliza el método de Runge-Kutta-Fehlberg de 4to.orden (explícito), que además nos provee una estimación del error de truncamiento en cada paso. **EL SOLVER ES CAPAZ DE ELEGIR EL TAMAÑO DE PASO QUE REQUIERE LA TOLERANCIA DEL ERROR QUE HEMOS ESPECIFICADO.**
- Todos los solvers de Matlab modificarán el tamaño del paso para generar una solución que sea exacta para una determinada tolerancia del error. Ud. puede pasar por alto este aspecto fijando un tamaño de paso máximo (que puede forzar al solver a adoptar un paso demasiado pequeño). Es mejor dejar que el solver elija el tamaño del paso y utilizar la opción ***refine***. ***Refine*** suaviza la solución interpolando entre puntos.
- Las opciones se establecen creando una estructura de opciones con el comando ***odeset***.

Utilización de Solvers de MATLAB

Ejemplo 02

Resolver el siguiente sistema acoplado de EDOs :

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -1 & -1 \\ 1 & -5000 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Condiciones iniciales

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

Los autovalores de la matriz son:

$$\lambda_1 = -5000$$

$$\lambda_2 = -1.002$$

$\rightarrow$ **Problema Stiff!!**

Utilización de Solvers de MATLAB

Ejemplo 02

- Un problema stiff consiste de diferentes procesos de los cuales al menos uno de ellos tendrá una constante de tiempo muy pequeña.
- En primer lugar utilizamos la **ode45** para resolver el sistema acoplado de EDOs.
- El código para resolver este sistema se muestra a continuación.

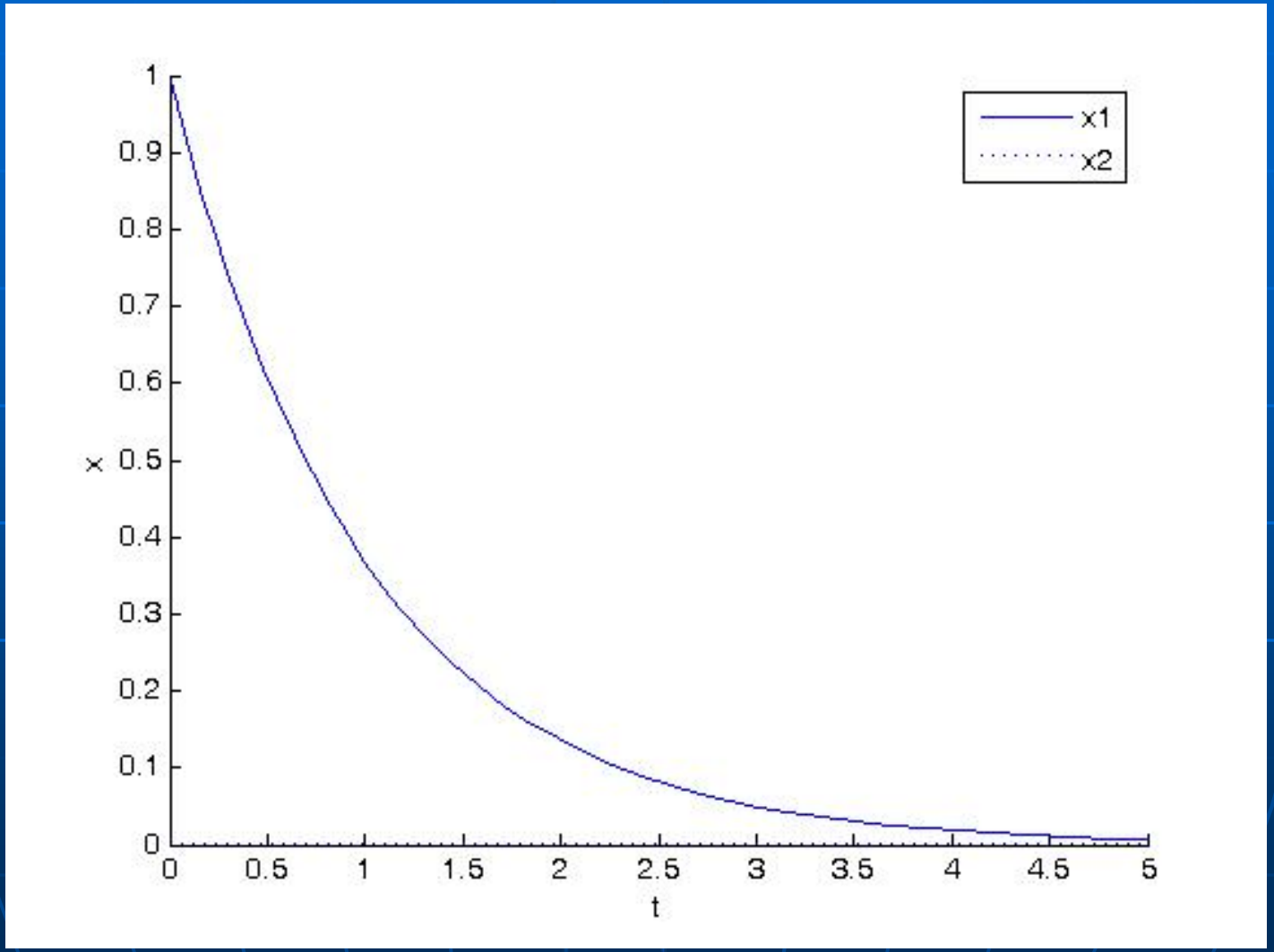
```
clear all
clc
%Example 02: Stiff Problem
% A simple example to solve ODE's
% Uses ODE45 to solve
% dx_dt(1) = -1*x(1)-1*x(2)
% dx_dt(2) = 1*x(1) -5000*x(2)
%set an error
options=odeset('RelTol',1e-6,'Stats','on');
%initial conditions
Xo = [1;1];
%timespan
tspan = [0,5];
%call the solver
tic
[t,X] = ode45(@TestFunction,tspan,Xo,options);
toc
%plot the results
figure;hold on
plot(t,X(:,1));plot(t,X(:,2),':')
legend('x1','x2'); ylabel('x');xlabel('t')
```

Utilización de Solvers de MATLAB

Ejemplo 02

- Los **stats** informados por la solución son:
  - 7557 successful steps
  - 504 failed attempts
  - 48367 function evaluations
  - Elapsed time is 3.797000 seconds.
- El solver ha sido forzado a adoptar un paso temporal muy pequeño para generar una solución estable.

Gráfica de la Solución



Utilización de Solvers de MATLAB

Ejemplo 02

- La opción **'stats'** ha sido activada. El solver mostrará por pantalla información acerca de cómo ha realizado el cálculo.
- La ejecución del código demanda un tiempo relativamente largo.
- Los comandos **'tic'** y **'toc'** que preceden y suceden respectivamente al llamado de la **ode45**, estiman y muestran el tiempo que le demandó al solver alcanzar el resultado final.

Utilización de Solvers de MATLAB

Ejemplo 02

- Si en lugar utilizamos la **ode15s**, debemos cambiar la línea donde se llama al solver de la edo, así:

  [t,X] = ode15s(@TestFunction,tspan,Xo,options)

- En este caso Los **stats** informados por la solución son:
  - 139 successful steps
  - 3 failed attempts
  - 288 function evaluations
  - 1 partial derivatives
  - 27 LU decompositions
  - 284 solutions of linear systems
  - Elapsed time is 0.625000 seconds.

Utilización de Solvers de MATLAB

Ejemplo 02

- El solver puede adoptar pasos temporales mucho más grandes (139 comparado con los 7557 para la **ode45**), lo cual reduce el tiempo computacional total.
- **ode15s** es implícita, por lo tanto resuelve un sistema no lineal de ecuaciones en cada paso temporal.

Mejoras al Código (I)

- En cada etapa de tiempo la **ode15s** resuelve un conjunto de ecuaciones no lineales, para las cuales requerirá del Jacobiano de $f(t,x)$.
- Dado que la rutina no suministra información sobre el Jacobiano, está forzada a calcular el Jacobiano completo en forma numérica.

Mejoras al Código (II)

- Por consiguiente, se nos plantean dos opciones:
  1) Suministrar una **function** que nos devuelva el Jacobiano, o si el Jacobiano es una constante, entonces podemos suministrar la matriz.
  2) Suministrar un Jacobiano patrón. En este caso, la rutina es capaz de evitar el costoso llamado a la velocidad de cambio de la función. Un Jacobiano patrón es una matriz rala de ceros y unos. Los unos sólo aparecen donde el Jacobiano es no nulo.

Mejoras al Código (III)

- **Opción 1:**
  - Para hacer esto fijamos la opción **'Jacobian'** en la estructura **odeset** para la matrix o nombre de la **function**. Generalmente ésta es la opción computacionalmente más eficiente.
  - MATLAB también permite a la rutina devolver un Jacobiano ralo (sparse). En este ejemplo el Jacobiano es una constante, utilizándose las siguientes líneas para suministrar al solver:

    ```
    J=[-1, -1; 1, -5000];
    options = odeset('RelTol',1e-6,'Stats','on','Jac',J);
    ```

  - El código ahora corre más rápido dado que el Jacobiano no necesita más ser evaluado (aún cuando la ganancia sea muy pequeña cuando el Jacobiano es constante, lo cual es detectado por el solver, por lo que el Jacobiano no será actualizado después de la primera evaluación)