Métodos Runge-Kutta Implícitos

En esta práctica implementaremos en MATLAB los métodos Runge-Kutta implícitos para resolver el problema de valor inicial (PVI):

$$\begin{aligned} &Y' = f(t, Y), \qquad t \in [t_0, T], \\ &Y(t_0) = Y_0, \end{aligned}$$

con Y_0 vector de tamaño d.

Runge-Kutta Implícitos (RKI)

Un método Runge-Kutta implícito $RKI(A, b)$ viene dado mediante las fórmulas:

$$\begin{aligned} U_i =& y_n + h \sum_{j=1}^{s} a_{ij} f(t_n + c_j h, U_j), \quad i = 1, \ldots, s, \\ y_{n+1} =& y_n + h \sum_{j=1}^{s} b_j f(t_n + c_j h, U_j), \end{aligned}$$

donde A es una matriz cuadrada de orden s, en particular, no es una matriz triangular inferior, y b es el vector pesos de tamaño s. Los nodos temporales en los pasos parciales satisface la condición suma:

$$c_i = \sum_{j=1}^{s} a_{ij}.$$

Resolución numérica de las ecuaciones implícitas de los RKI

Como ya sabemos, en los métodos RKI cada vez que damos un paso $(t_n, y_n) \underset{h}{\to} (t_{n+1}, y_{n+1})$ hay que resolver un sistema implícito (SI) de dimensión ds para calcular las s etapas del método:

$$U = e \otimes y_n + h(A \otimes I)F(U), \qquad \text{(SI)}$$

donde

$$U = \begin{pmatrix} U_1 \\ \vdots \\ U_s \end{pmatrix}, \quad F(U) = \begin{pmatrix} f(t_n + c_1 h, U_1) \\ \vdots \\ f(t_n + c_s h, U_s) \end{pmatrix},$$

son vectores de $\mathbb{R}^{sd}$. En (SI), $e = (1, 1, \ldots, 1)^T \in \mathbb{R}^s$, $\otimes$ denota el producto de Kronecker, A es la matriz del RKI e I es la matriz identidad de tamaño d. Este producto se define de la siguiente manera para A matriz $M(\alpha_1, \alpha_2)$, B matriz $M(\beta_1, \beta_2)$, entonces $A \otimes B$ es una matriz $M(\alpha_1\beta_1, \alpha_2\beta_2)$ tal que:

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1\alpha_2} \\ a_{21} & a_{22} & \dots & a_{2\alpha_2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{\alpha_1 1} & a_{\alpha_1 2} & \dots & a_{\alpha_1\alpha_2} \end{bmatrix} \otimes \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1\beta_2} \\ b_{21} & b_{22} & \dots & b_{2\beta_2} \\ \vdots & \vdots & \ddots & \vdots \\ b_{\beta_1 1} & b_{\beta_1 2} & \dots & b_{\beta_1\beta_2} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} & \dots & a_{11}b_{1\beta_2} & \dots & \dots & a_{1\alpha_2}b_{11} & a_{1\alpha_2}b_{12} & \dots & a_{1\alpha_2}b_{1\beta_2} \\ a_{11}b_{21} & a_{11}b_{22} & \dots & a_{11}b_{2\beta_2} & \dots & \dots & a_{1\alpha_2}b_{21} & a_{1\alpha_2}b_{22} & \dots & a_{1\alpha_2}b_{2\beta_2} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{11}b_{\beta_1 1} & a_{11}b_{\beta_1 2} & \dots & a_{11}b_{\beta_1\beta_2} & \dots & \dots & a_{1\alpha_2}b_{\beta_1 1} & a_{1\alpha_2}b_{\beta_1 2} & \dots & a_{1\alpha_2}b_{\beta_1\beta_2} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{\alpha_1 1}b_{11} & a_{\alpha_1 1}b_{12} & \dots & a_{\alpha_1 1}b_{1\beta_2} & \dots & \dots & a_{\alpha_1\alpha_2}b_{11} & a_{\alpha_1\alpha_2}b_{12} & \dots & a_{\alpha_1\alpha_2}b_{1\beta_2} \\ a_{\alpha_1 1}b_{21} & a_{\alpha_1 1}b_{22} & \dots & a_{\alpha_1 1}b_{2\beta_2} & \dots & \dots & a_{\alpha_1\alpha_2}b_{21} & a_{\alpha_1\alpha_2}b_{22} & \dots & a_{\alpha_1\alpha_2}b_{2\beta_2} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{\alpha_1 1}b_{\beta_1 1} & a_{\alpha_1 1}b_{\beta_1 2} & \dots & a_{\alpha_1 1}b_{\beta_1\beta_2} & \dots & \dots & a_{\alpha_1\alpha_2}b_{\beta_1 1} & a_{\alpha_1\alpha_2}b_{\beta_1 2} & \dots & a_{\alpha_1\alpha_2}b_{\beta_1\beta_2} \end{bmatrix}$$

Se pueden utilizar diferentes métodos numéricos que resuelven el sistema implícito (SI).

(IF) Iteración funcional

La iteración funcional se calcula como:

$$U^{k+1} = e \otimes y_n + h(A \otimes I)F(U^k), \quad k = 0, 1, \dots.$$

Partiendo de una iteración U^0 que normalmente se suele tomar $U^0 = e \otimes y_n$. Paramos la iteración cuando el avance es menor de una cota exigida o cuando hemos iterado ya un gran número de veces. La iteración funcional utiliza simplemente la propiedad de que F es contractiva.
Observación: Método muy sencillo pero convergencia lenta.

(MN) Método de Newton

La idea es aplicar el método de Newton clásico, es decir, buscar un cero de la función escalar $g(x)$ mediante la iteración

$$x_{n+1} = x_n - \frac{g(x_n)}{g'(x_n)}.$$

Esto se generaliza a una función vectorial $G(x)$ en $\mathbb{R}^d$ con $\in \mathbb{R}^d$ como:

$$x_{n+1} = x_n - (G'(x_n))^{-1}G(x_n),$$

donde $G'(x_n)$ es el jacobianode G en x_n. Así, aplicamos el método de Newton clásico (vectorial) a la ecuación en $\mathbb{R}^{sd}$

$$G(U) := U - e \otimes y_n - h(A \otimes I)F(U) = 0.$$

Para ello, calculamos su derivada

$$G'(U) = I - h(A \otimes I)J(U), \tag{1}$$

con

$$J(U) := \frac{\partial F}{\partial U}(U) = \begin{pmatrix} J_1(U) & & 0 \\ & \ddots & \\ 0 & & J_s(U) \end{pmatrix}, \tag{2}$$

y donde J_i se escribe en función del jacobiano de f

$$J_i(U) = \frac{\partial f}{\partial y}(t_n + c_i h, U_i).$$

Ahora, partiendo de un valor inicial U^0, normalmente utilizamos $U^0 = e \otimes y_n$, resulta el siguiente esquema del método numérico(MN):

$$\begin{array}{l} [I - h(A \otimes I)J(U^k)]\Delta^k = -G(U^k), \\ U^{k+1} = U^k + \Delta^k, \quad k = 0, 1, \ldots, \end{array} \qquad \text{(MN)}$$

donde G y J son definidos por (1) y (2), respectivamente.
Observación: Método más complejo, convergencia mejor (cuadrática), pero muchas evaluaciones del jacobiano e inversión de la matriz jacobiana (más costoso).

(QN) Método de Newton modificado o Quasi-Newton

El método anterior es demasiado costoso pues hay que calcular en cada iteración n una evaluación del jacobiano de dimension ms en s puntos distintos y resolver un sistema lineal. Por eso se prefieren algunas variaciones que son más baratas computacionalmente.

El método más usado en la práctica, es el método de Newton modificado o Quasi-Newton. Es una modificación del (MN) que reduce su coste fijando $G'(U)$ definida en (1) sin tener que depender de la iteración. Así la matriz jacobiana sólo se evalúa una vez por paso y los sistemas lineales de la iteración tienen la misma matriz de coeficientes para todas las iteraciones del mismo paso, lo que implica una sola descomposición LU por cada paso de tiempo.

En definitiva, definiendo la matriz J_n de s bloques diagonales donde cada uno de ellos es el jacobiano de f en (t_n, y_n), es decir,

$$J_n \approx \begin{pmatrix} \frac{\partial f}{\partial u}(t_n, y_n) & & 0 \\ & \ddots & \\ 0 & & \frac{\partial f}{\partial u}(t_n, y_n) \end{pmatrix}$$

resulta el esquema Quasi-Newton:

$$\begin{array}{l} [I - h(A \otimes J_n)]\Delta^k = -G(U^k) \\ U^{k+1} = U^k + \Delta^k, k = 0, 1, \ldots, \end{array} \qquad \text{(QN)}$$

arrancando de una aproximación inicial $U^0 = e \otimes y_n$.

Programa Matlab

La función a construir es la siguiente:

```
function [u,t,it]=RKIqN(f,df,N,t0,T,y0,b,c,A,tol,itmax)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Esta función resuelve el problema de valor inicial
%       Y'=f(t,Y)
%       Y(t0)=y0
% utilizandodo un método Runge-Kutta implícito con una iteración quasi Newton
%
%          [u,t,it]=RKIqN(f,df,N,t0,T,y0,b,c,A,tol,itmax)
%
```

```
%  Variables de Entrada:
%
%       f: vector columna. función que rige el sistema de EDO,
%          tiene dos argumentos f(t,u) donde t es escalar (ó vector)
%          y u vector columna.
%       df: matriz jacobiana de f para un escalar t y vector u
%       t0: tiempo inicial
%       T: tiempo final
%       N: número de pasos
%       u0: vector columna. Dato inicial
%       A,b,c: coeficientes del tablero de BUTCHER.
%              A, matriz cuadrada de orden s
%              c, vector columna de tamaño de orden s
%              b, vectorcolumna de tamaño de orden s
%       tol: tolerancia para las iteraciones
%       itmax: Número máximo de iteraciones
%
%  Variables de Salida:
%
%       u: matriz de length(u0) x length(t) que contiene la solución
%       t: vector de tiempos
%       it: Vector que contiene el numero de iteraciones utilizadas, para
%           cada paso de tiempo, por el método iterativo funcional
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Tableros de Butcher que usaremos:

- Método de Gauss: Regla Implícita del punto medio (orden 2)

$\frac{1}{2}$	$\frac{1}{2}$
	1

- Radau IIA: Euler Implícito

1	1
	1

- Lobatto IIIA. Regla Trapezoidal

0	0	0
1	1/2	1/2
	1/2	1/2

Comentarios: Para realizar esta práctica se puede implementar todas las funciones o ficheros .m auxiliares que se necesiten aparte de los citados aquí.

1 Ejercicios:

1. Utilizar la función Runge-Kutta implícito para resolver los problemas de valor inicial:

   (a) Para $t \in [0, 10]$,

$$\begin{cases} Y' = \begin{bmatrix} -2 & 1 \\ 1 & -2 \end{bmatrix} Y + \begin{bmatrix} 2\sin(t) \\ 2(\cos(t) - \sin(t)) \end{bmatrix} \\ Y(0) = \begin{bmatrix} 2 \\ 3 \end{bmatrix} \end{cases}$$

cuya solución es:

$$Y(t) = \begin{pmatrix} 2e^{-t} + \sin(t) \\ 2e^{-t} + \cos(t) \end{pmatrix}$$

(b) Para $t \in [0, 10]$,

$$\begin{cases} Y' = \begin{bmatrix} -2 & 1 \\ 998 & -999 \end{bmatrix} Y + \begin{bmatrix} 2\sin(t) \\ 999(\cos(t) - \sin(t)) \end{bmatrix} \\ Y(0) = \begin{bmatrix} 2 \\ 3 \end{bmatrix} \end{cases}$$

cuya solución también es:

$$Y(t) = \begin{pmatrix} 2e^{-t} + \sin(t) \\ 2e^{-t} + \cos(t) \end{pmatrix}$$

2. Representar gráficamente la solución aproximada utilizando los diferentes tableros de BUTCHER.
3. Representar gráficamente en escala doblemente logarítmica las poligonales formadas por los valores (h, error), ($h, \max(it)$) para una cantidad significativa de valores de h.

Observaciones para Matlab

- $\otimes$ indica el producto de Kronecker. En matlab este producto se hace con la función **kron**.
- I indica la matriz identidad, en MATLAB se pueden crear de diferentes tamaños utilizando el comando **eye**.
- Para generar vectores o matrices de unos, se utiliza el comando **one**.
- Para generar una matriz con las matrices J_1,..., J_s en su diagonal, se utiliza el comando **blkdiag**:

$$blkdiag(J_1, \ldots, J_s) = \begin{pmatrix} J_1(U) & & 0 \\ & \ddots & \\ 0 & & J_s(U) \end{pmatrix}$$

- Para resolver el sistema $Ax = b$, se suele utilizar $x = A \backslash b$.
- Para consultar cada comando, se puedo usar **help**.