1.4 – Métodos de paso adaptativo

1.4.1 – Introducción

En un método numérico para resolución de EDO, en ocasiones es inconveniente mantener el paso de integración constante, como consecuencia del costo asociado con las operaciones a realizar. Cuando la solución varíe suavemente, convendrá adoptar un paso "grande", mientras que cuando la misma varíe "rápidamente", será conveniente adoptar un paso menor.
Los métodos de paso adaptativo plantean, además del cálculo de y_{n+1} y de f en cada paso de discretización, determinar el valor adecuado para el paso siguiente, de acuerdo al siguiente procedimiento:

1) Estimar el error local
2) Decidir si el valor calculado de y_{n+1} puede ser aceptado, o si se debe usar un paso más pequeño desde el punto anterior.
3) Determinar el tamaño del paso siguiente a usarse

1.4.2 – Implementación de métodos con paso variable

Se estudiará el error local con el objetivo de controlar indirectamente el error global cometido en la resolución de la EDO.
Considérese un método de un paso, de orden de consistencia p; el error local de truncamiento es:

$$e_{n+1} = \tilde{y}(x_{n+1}) - y_{n+1} = C(x_n)h_n^{p+1} + 0(h_n^{p+2}) \qquad (1)$$

(siendo $\tilde{y}$ es la solución exacta que pasa por $(x_{n.}, y_n)$.
Remitiéndose al análisis de la evolución del error global realizado en 1.2.2, si el error local por unidad de longitud satisface

$$\left|\frac{e_{n+1}}{h_n}\right| < tol \qquad (2)$$

y las condiciones de unicidad de la solución se cumplen (*f* lipschitziana), entonces el error global de truncamiento satisface,

$$|E_n| < tol\left|\frac{e^{L(x_n-a)}-1}{L}\right|,\ \forall x_n \in [a,b]$$

La estrategia a adoptar para la elección de cada paso deberá garantizar que se satisfaga la cota (2).

La primera dificultad que aparece detrás de esta estratregia es que el valor de h_n no se puede calcular directamente de lo anterior porque generalmente la función $C(x)$ de (1) no se conoce, aunque se asumirá que la misma varía suavemente.
El procedimiento que se usa para salvar la dificultad expuesta es tomar un paso *H,* **estimar el error local cometido en** $x_n + H$, y utilizar dicha estimación para calcular el paso $h_n = qH$ que se usará efectivamente.

Si e^*_{n+1} es el error local estimado asociado con el paso *H*, entonces

$$e^*_{n+1} \approx C(x_n) H^{p+1} \qquad (3)$$

y el error local producido por un paso qH es, asumiendo el mismo valor de $C(x_n)$.

$$e_{n+1} \approx C(x_n)(qH)^{p+1} \qquad (4)$$

La desigualdad (2) se puede satisfacer en caso que

$$\left| C(x_n)\ (q\ H)^p \right| < tol$$

Usando la estimación (3) para eliminar $C(x_n)$, se tiene que la elección

$$q < \left| \frac{tol\ H}{d_{n+1}^*} \right|^{1/p} \qquad (5)$$

cumple con la cota.

En términos de costo de las operaciones, es conveniente tomar el mayor paso posible, pero como rechazar un paso puede resultar inadecuado, se toma: $q = 0{,}8 \left| \frac{tol\ H}{e^*_{n+1}} \right|^{1/p}$

El factor 0,8 es un **factor de seguridad** que compensa en parte las aproximaciones hechas al deducir el valor de q en (5)

Observaciones:

1) Si se tiene q >1 entonces *H* utilizado era menor de lo necesario, en vez de recalcular el paso se acepta el valor y_{n+1} con $h_n = H$ (ya que el resultado se calculó con mayor precisión de la necesaria). Cuando un paso es aceptado se comienza el siguiente usando qH en lugar de *H*. (Siempre asumiendo que $C(x)$ no varía demasiado en el intervalo, es decir que $C(x_{n+1})$ no es muy distinto a $C(x_n)$)
2) Si q <1 entonces se debe repetir el paso con qH en vez de *H*.
3) Hay otras restricciones que deben ser impuestas a h_n. Por ejemplo, si el mismo se vuelve demasiado pequeño, los errores de redondeo pueden hacer que los resultados obtenidos carezcan de valor.
   Por otra parte, una longitud de paso demasiado grande no debe usarse, pues se podría perder alguna característica de la solución. El valor h_{max} depende de cada problema y debe ser proporcionada por el usuario.
   Es inconveniente cambios demasiado abruptos de la longitud del paso.
   Usualmente se restringe el incremento del paso a un factor de 2.
   Un factor por el cual se puede achicar el paso es difícil de asignar, pues si es muy grande entonces muchos pasos posteriores deberán ser rechazados. Un número sugerido es $1/10$.

1.4.3 – Estimación del error local de truncamiento

A continuación se estudia el cálculo de la estimación e_{n+1}^{*}, que quedó pendiente de la sección anterior.

Si y_{n+1} se calcula por métodos de orden *p* y *p+1*, entonces la diferencia entre los valores calculados de y_{n+1} por ambos métodos consiste en una buena aproximación del error local de truncamiento en el método de orden *p*.

Efectivamente, si se usan $\bar{y}$ y $\hat{y}$ para denotar el método de orden *p* y $p+1$ respectivamente, se tiene que

$$\tilde{y}(x_{n+1}) - \bar{y}_{n+1} = C(x_n) h_n{}^{p+1} + 0\left(h_n{}^{p+2}\right) \qquad (6)$$

y

$$\tilde{y}(x_{n+1}) - \hat{y}_{n+1} = 0\left(h_n{}^{p+2}\right) \qquad (7)$$

Restando estas dos ecuaciones se obtiene ,

$$\hat{y}_{n+1} - \bar{y}_{n+1} = C(x_n) h_n{}^{p+1} + 0\left(h_n{}^{p+2}\right) \qquad (8)$$

por lo cual una estimación del error local (mediante su parte principal) es,

$$e^{*}_{n+1} = \hat{y}_{n+1} - \bar{y}_{n+1} \qquad (10)$$

Nótese que al ser calculada mediante un método de mayor orden de consistencia, $\hat{y}_{n+1}$ será generalmente una mejor aproximación a $\tilde{y}(x_{n+1})$ que $\bar{y}_{n+1}$ y por lo tanto deberá ser usada como valor de la aproximación y_{n+1}.

Observación

En este caso y_{n+1} se toma como el resultado obtenido usando el método de orden $p+1$. Si el resultado de mayor orden es usado en los cálculos pero la selección del paso se basa en el esquema de orden menor, la solución calculada será más precisa que lo predicho por (4).

1.4.4 – Método de Runge Kutta con paso variable (RKF)

Siguiendo las ideas presentadas anteriormente para la estimación del error local, Fehlberg desarrolló un par de métodos Runge Kutta de orden 4 y 5 asociados, de tal forma que ambos utilizan en cada paso las mismas evaluaciones de ***f*** con el fin de ahorrar cálculos.

El método de orden 5 requiere 6 evaluaciones de ***f*** en cada paso. Fehlberg demostró que es posible usar 5 de esos valores para desarrollar un método de orden 4 (en lugar del método clásico, que usa 4 evaluaciones de ***f***). El estimador de orden 4 así obtenido puede utilizarse junto con el de orden 5 para estimar el error local, sin necesidad de nuevas evaluaciones de ***f***, y habilitando una estimación del nuevo paso h.

Los coeficientes del método RKF son:

$\frac{1}{4}$	$\frac{1}{4}$				
$\frac{3}{8}$	$\frac{3}{32}$	$\frac{9}{32}$			
$\frac{12}{13}$	$\frac{1932}{2197}$	$-\frac{2200}{2197}$	$\frac{7296}{2197}$		
1	$\frac{439}{216}$	-8	$\frac{3680}{513}$	$-\frac{845}{4104}$	
$\frac{1}{2}$	$-\frac{8}{27}$	2	$-\frac{3544}{2565}$	$\frac{1859}{4104}$	$-\frac{11}{40}$
	$\left[\frac{16}{135}\right.$	0	$\frac{6656}{12825}$	$\frac{28561}{56430}$	$-\frac{9}{50}$ $\left.\frac{2}{55}\right] = \mathbf{a}$
	$\left[\frac{1}{360}\right.$	0	$-\frac{128}{4275}$	$-\frac{2197}{75240}$	$\frac{1}{50}$ $\left.\frac{2}{55}\right] = \mathbf{b}$

Tabla – Coeficientes de Runge-Kutta-Fehlberg de orden 5

donde:

$$y_{n+1} = y_n + h\mathbf{a}\ \mathbf{k}$$
$$e^*_{n+1} = h\mathbf{b}\mathbf{k}$$

Las funciones ode23 y ode45 de MATLAB implementan los métodos de Runge – Kutta – Fehlberg de orden dos y tres y cuatro y cinco respectivamente, de acuerdo al enfoque explicado. Una sinopsis de su funcionamiento y utilización puede obtenerse mediante la ayuda asociada (help <ode23, ode45>). Las ayudas para MATLAB versiòn 6.1.0 se ofrecen en el anexo I

1.4.5 – Una alternativa para estimar el error local

Como alternativa para estimar el error local e^*_{n+1}, puede aplicarse la técnica de extrapolación de Richardson.

El paso desde x_n a x_{n+1} se realiza dos veces, primero con un paso *H* y luego con paso menor ($H/2$ por ejemplo).

Si el método tiene orden *p* y se denotan los resultados por $\bar{y}_{n+1}$ y $\hat{y}_{n+1}$ respectivamente, entonces

$$\tilde{y}(x_{n+1}) - \bar{y}_{n+1} = C(x_n)\,H^{p+1} + 0\left(H^{p+2}\right)$$

y también se puede probar que

$$\tilde{y}(x_{n+1}) - \hat{y}_{n+1} = 2C(x_n)\left(\frac{H}{2}\right)^{p+1} + 0\left(H^{p+2}\right),$$

donde se ha asumido que la función $C(x)$ varía suavemente, es decir que

$$C(x_n) \cong C\left(x_n + \frac{H}{2}\right)$$

restando, se tiene que

$$\hat{y}_{n+1} - \bar{y}_{n+1} = 2C(x_n)\left(\frac{H}{2}\right)^{p+1}\left(2^p - 1\right) + 0\left(H^{p+2}\right)$$

por lo que una estimación del error local de $\hat{y}_{n+1}$ es, $e^*_{n+1} = \dfrac{\hat{y}_{n+1} - \bar{y}_{n+1}}{2^p - 1}$

La extrapolación local da para el siguiente valor la fórmula

$$y_{n+1} = \hat{y}_{n+1} + e^*_{n+1}$$

Nótese que este método de estimación del error local requiere evaluar la función ***f*** varias veces, por lo que éste enfoque puede resultar inadecuado en términos de costo de las operaciones, en caso que dicha función sea complicada.

Anexo I – Help de MATLAB para ode23 y ode45 (Matlab Version 6.1.0)

ODE23 Solve non-stiff differential equations, low order method.
[T,Y] = ODE23(ODEFUN,TSPAN,Y0) with TSPAN = [T0 TFINAL] integrates the system of differential equations y' = f(t,y) from time T0 to TFINAL with initial conditions Y0. Function ODEFUN(T,Y) must return a column vector corresponding to f(t,y). Each row in the solution array Y corresponds to a time returned in the column vector T. To obtain solutions at specific times T0,T1,...,TFINAL (all increasing or all decreasing), use TSPAN = [T0 T1 ... TFINAL].

[T,Y] = ODE23(ODEFUN,TSPAN,Y0,OPTIONS) solves as above with default integration properties replaced by values in OPTIONS, an argument created with the ODESET function. See ODESET for details. Commonly used options are scalar relative error tolerance 'RelTol' (1e-3 by default) and vector of absolute error tolerances 'AbsTol' (all components 1e-6 by default)

T,Y] = ODE23(ODEFUN,TSPAN,Y0,OPTIONS,P1,P2,...) passes the additional parameters P1,P2,... to the ODE function as ODEFUN(T,Y,P1,P2...), and to all functions specified in OPTIONS. Use OPTIONS = [] as a place holder if no options are set.

ODE23 can solve problems M(t,y)*y' = f(t,y) with mass matrix M that is nonsingular. Use ODESET to set the 'Mass' property to a function MASS if MASS(T,Y) returns the value of the mass matrix. If the mass matrix is
constant, the matrix can be used as the value of the 'Mass' property. If he mass matrix does not depend on the state variable Y and the function MASS is to be called with one input argument T, set 'MStateDependence' to none'. ODE15S and ODE23T can solve problems with singular mass matrices.

T,Y,TE,YE,IE] = ODE23(ODEFUN,TSPAN,Y0,OPTIONS...) with the 'Events' property in OPTIONS set to a function EVENTS, solves as above while also inding where functions of (T,Y), called event functions, are zero. For each function you specify whether the integration is to terminate at a zero and whether the direction of the zero crossing matters. These are the three vectors returned by EVENTS: [VALUE,ISTERMINAL,DIRECTION] =EVENTS(T,Y). For the I-th event function: VALUE(I) is the value of the
function, ISTERMINAL(I)=1 if the integration is to terminate at a zero of this event function and 0 otherwise. DIRECTION(I)=0 if all zeros are to be computed (the default), +1 if only zeros where the event function is increasing, and -1 if only zeros where the event function is decreasing. Output TE is a column vector of times at which events occur. Rows of YE are the corresponding solutions, and indices in vector
IE specify which event occurred.

SOL = ODE23(ODEFUN,[T0 TFINAL],Y0...) returns a structure that can be used with DEVAL to evaluate the solution at any point between T0 and TFINAL. The steps chosen by ODE23 are returned in a row vector SOL.x. For each I, the column SOL.y(:,I) contains the solution at SOL.x(I). If events were detected, SOL.xe is a row vector of points at which events occurred. Columns of SOL.ye are the corresponding solutions, and indices on vector SOL.ie specify which event occurred. If a terminal event has been detected, SOL.x(end) contains the end of the step at which the event occurred. The exact point of the event is reported in SOL.xe(end).

Example
```
[t,y]=ode23(@vdp1,[0 20],[2 0]);
plot(t,y(:,1));
```
solves the system y' = vdp1(t,y), using the default relative error
tolerance 1e-3 and the default absolute tolerance of 1e-6 for each
component, and plots the first component of the solution.

See also
other ODE solvers: ODE45, ODE113, ODE15S, ODE23S, ODE23T, ODE23TB
options handling: ODESET, ODEGET
output functions: ODEPLOT, ODEPHAS2, ODEPHAS3, ODEPRINT
evaluating solution: DEVAL
ODE examples: RIGIDODE, BALLODE, ORBITODE

NOTE:

The interpretation of the first input argument of the ODE solvers and some properties available through ODESET have changed in this version of MATLAB. Although we still support the v5 syntax, any new
functionality is available only with the new syntax. To see the v5 help, type in the command line
more on, type ode23, more off

ODE45 Solve non-stiff differential equations, medium order method.

[T,Y] = ODE45(ODEFUN,TSPAN,Y0) with TSPAN = [T0 TFINAL] integrates the system of differential equations y' = f(t,y) from time T0 to TFINAL with initial conditions Y0. Function ODEFUN(T,Y) must return a column vector corresponding to f(t,y). Each row in the solution array Y corresponds to a time returned in the column vector T. To obtain solutions at specific times T0,T1,...,TFINAL (all increasing or all decreasing), use TSPAN = [T0 T1 ... TFINAL].

[T,Y] = ODE45(ODEFUN,TSPAN,Y0,OPTIONS) solves as above with default integration properties replaced by values in OPTIONS, an argument created with the ODESET function. See ODESET for details. Commonly used options are scalar relative error tolerance 'RelTol' (1e-3 by default) and vector of absolute error tolerances 'AbsTol' (all components 1e-6 by default).

[T,Y] = ODE45(ODEFUN,TSPAN,Y0,OPTIONS,P1,P2...) passes the additional parameters P1,P2,... to the ODE function as ODEFUN(T,Y,P1,P2...), and to all functions specified in OPTIONS. Use OPTIONS = [] as a place holder if no options are set.

ODE45 can solve problems M(t,y)*y' = f(t,y) with mass matrix M that is nonsingular. Use ODESET to set the 'Mass' property to a function MASS if MASS(T,Y) returns the value of the mass matrix. If the mass matrix is
constant, the matrix can be used as the value of the 'Mass' option. If the mass matrix does not depend on the state variable Y and the function MASS is to be called with one input argument T, set 'MStateDependence' to 'none'. ODE15S and ODE23T can solve problems with singular mass matrices.

[T,Y,TE,YE,IE] = ODE45(ODEFUN,TSPAN,Y0,OPTIONS...) with the 'Events' property in OPTIONS set to a function EVENTS, solves as above while also finding where functions of (T,Y), called event functions, are zero. For each function you specify whether the integration is to terminate at a zero and whether the direction of the zero crossing matters. These are the three vectors returned by EVENTS: [VALUE,ISTERMINAL,DIRECTION] = EVENTS(T,Y). For the I-th event function: VALUE(I) is the value of the
function, ISTERMINAL(I)=1 if the integration is to terminate at a zero of this event function and 0 otherwise. DIRECTION(I)=0 if all zeros are to be computed (the default), +1 if only zeros where the event function is increasing, and -1 if only zeros where the event function is decreasing. Output TE is a column vector of times at which events occur. Rows of YE are the corresponding solutions, and indices in vector
IE specify which event occurred.

SOL = ODE45(ODEFUN,[T0 TFINAL],Y0...) returns a structure that can be used with DEVAL to evaluate the solution at any point between T0 and TFINAL. The steps chosen by ODE45 are returned in a row vector SOL.x. For each I, the column SOL.y(:,I) contains the solution at SOL.x(I). If events were detected, SOL.xe is a row vector of points at which event occurred. Columns of SOL.ye are the corresponding solutions, and indices in vector SOL.ie specify which event occurred. If a terminal event has been detected, SOL.x(end) contains the end of the step at which the event occurred. The exact point of the event is reported in SOL.xe(end).

Example
```
    [t,y]=ode45(@vdp1,[0 20],[2 0]);
    plot(t,y(:,1));
```
solves the system y' = vdp1(t,y), using the default relative error tolerance 1e-3 and the default absolute tolerance of 1e-6 for each component, and plots the first component of the solution.

See also
other ODE solvers: ODE23, ODE113, ODE15S, ODE23S, ODE23T, ODE23TB
options handling: ODESET, ODEGET
output functions: ODEPLOT, ODEPHAS2, ODEPHAS3, ODEPRINT
evaluating solution: DEVAL

ODE examples: RIGIDODE, BALLODE, ORBITODE

NOTE:
The interpretation of the first input argument of the ODE solvers and some properties available through ODESET have changed in this version of MATLAB. Although we still support the v5 syntax, any new functionality is available only with the new syntax. To see the v5 help, type in the command line more on, type ode45, more off