Ecuaciones Diferenciales Ordinarias

Métodos Numéricos y Simulación.

Segundo de Grado en Física.

¿Qué es una ecuación diferencial ordinaria?

• Una ecuación diferencial ordinaria (ODE) es una ecuación en la que la incógnita es una función y(x) y que involucra derivadas de la función incógnita.

$$B(x,y) = A_1(x,y)\frac{dy}{dx} + A_2(x,y)\frac{d^2y}{dx^2} + \ldots + A_n(x,y)\frac{d^ny}{dx^n}$$

• Si n es el orden más alto de derivación que aparece en la ecuación, se dice que la ecuación diferencial es de orden n.

• Para resolver la ecuación anterior necesitamos cierta información adicional sobre la función y(x). Dependiendo de la información adicional, se distinguen dos tipos de problemas.

Tipos de problemas.

$$B(x,y) = A_1(x,y)\frac{dy}{dx} + A_2(x,y)\frac{d^2y}{dx^2} + \ldots + A_n(x,y)\frac{d^ny}{dx^n}$$

- Problemas de **condiciones iniciales**.

  Cuando para un determinado x_o, nos dan como dato:

$$y_o = y(x_o) \qquad \left(\frac{d^ky}{dx^k}\right)_{x=x_o} \qquad \forall k = 1,\ldots,n-1$$

  Y tenemos que encontrar y(x) para todo $x>x_o$.

  Estos problemas son los que estudiaremos en esta lección.

- Problemas de **condiciones de contorno**:

Nos dan información de la función y y sus derivadas en dos puntos x_o y x_1 y tenemos que encontrar y(x) para $x_o \leq x \leq x_1$.

Estos problemas los estudiaremos en el siguiente tema.

Ejemplo.

• Si tenemos N núcleos de una especie radioactiva, el número ΔN de núcleos que experimenta desintegración radiactiva en un tiempo Δt pequeño (de modo que $\Delta N/N << 1$) es proporcional a N.

$$\Delta N = -kN\Delta t$$

• Esto significa que en cualquier instante t, el número de núcleos N ha de cumplir la ecuación diferencial ordinaria:

$$\frac{dN}{dt} = -kN$$

• Sabemos que la solución de esta ecuación es:

$$N(t) = N_o e^{-kt}$$

Por poner un ejemplo, suponemos que N está dado en moles, t está dado en horas y k en 1/horas.

Clasificando el ejemplo

Comparando:

$$\frac{dN}{dt} = -kN$$

$$B(x,y) = A_1(x,y)\frac{dy}{dx} + A_2(x,y)\frac{d^2y}{dx^2} + \ldots + A_n(x,y)\frac{d^ny}{dx^n}$$

Podemos hacer las identificaciones:

$x \to t$	$B(x,y) \to -kN$	(que, en este caso particular, no depende de x)
$y \to N$	$A_1(x,y) \to 1$	$n \to 1$

Luego la ecuación que estamos estudiando es una **ecuación diferencial de primer orden de coeficientes constantes** (porque $A_1(x,y) = 1$, que es una constante)

El método de Euler

• Supongamos que no supiéramos encontrar la solución analítica de la ecuación diferencial ordinaria que estamos estudiando.

• Cuando presentamos el problema dijimos que en un tiempo Δt pequeño:

$$\Delta N = -kN\Delta t$$

• Si llamamos N_j al número de átomos que hay en el instante t_j y N_{j+1} al número de átomos que hay en el instante t_{j+1}, tenemos que, aproximadamente:

$$N_{j+1} - N_j = -kN_j(t_{j+1} - t_j)$$

• Y si escogemos todos los intervalos de tiempo t_{j+1}-t_j iguales a un terminado valor h:

$$N_{j+1} = N_j - kN_j h$$

El método de Euler

Escrito para un problema general, como:

$$\frac{dy}{dx} = f(x, y; k)$$

Cada paso queda:

1) Dado x_j, y_j, calcular la derivada en ese punto:

$$f_1 = f(x_j, y_j; k)$$

2) Calcular y_{j+1} como:

$$y_{j+1} = y_j + f_1 h$$

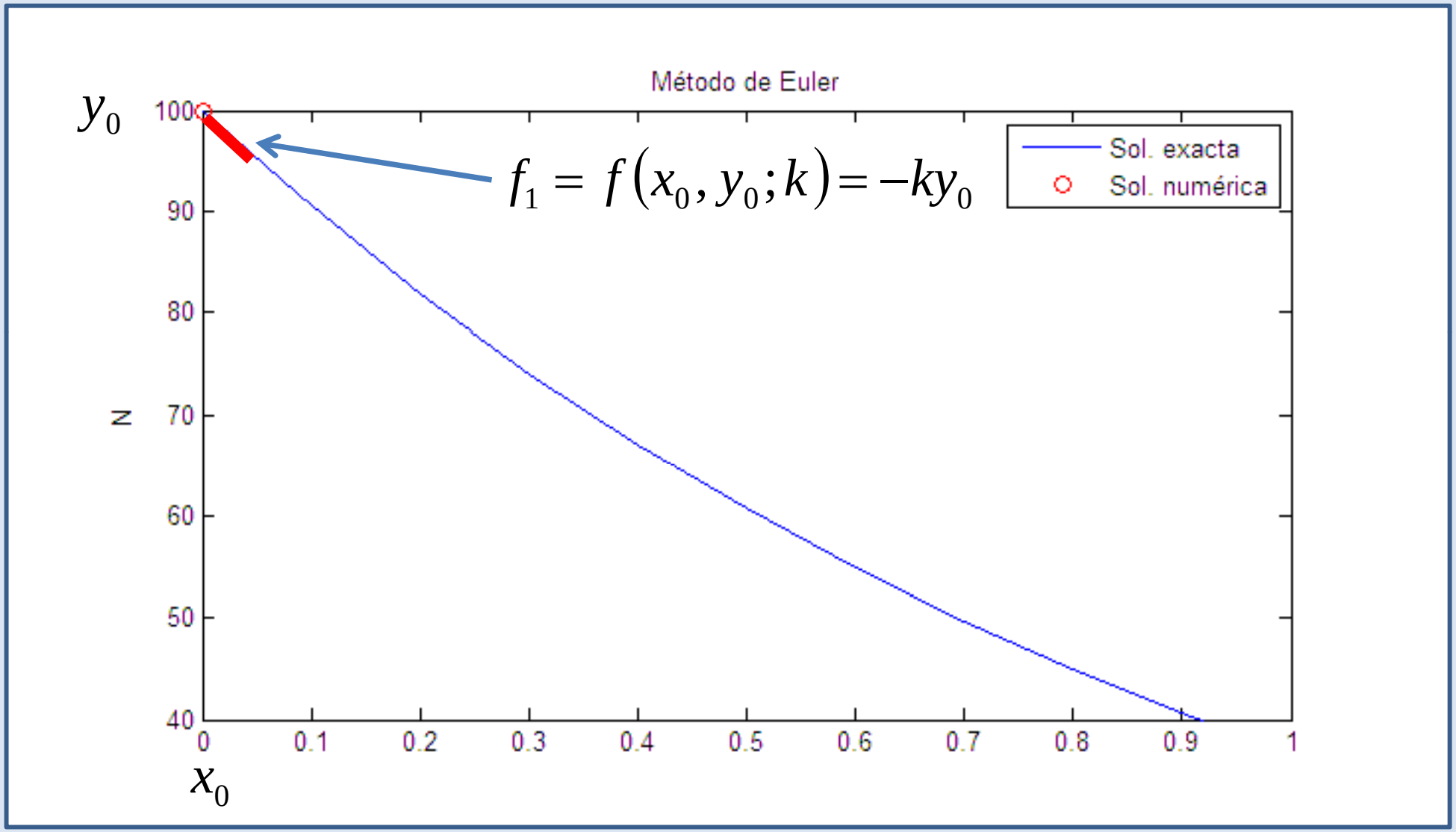
3) Calcular x_{j+1} como:

$$x_{j+1} = x_j + h$$

El método de Euler

1) Dado x_j, y_j, calcular la derivada en ese punto:

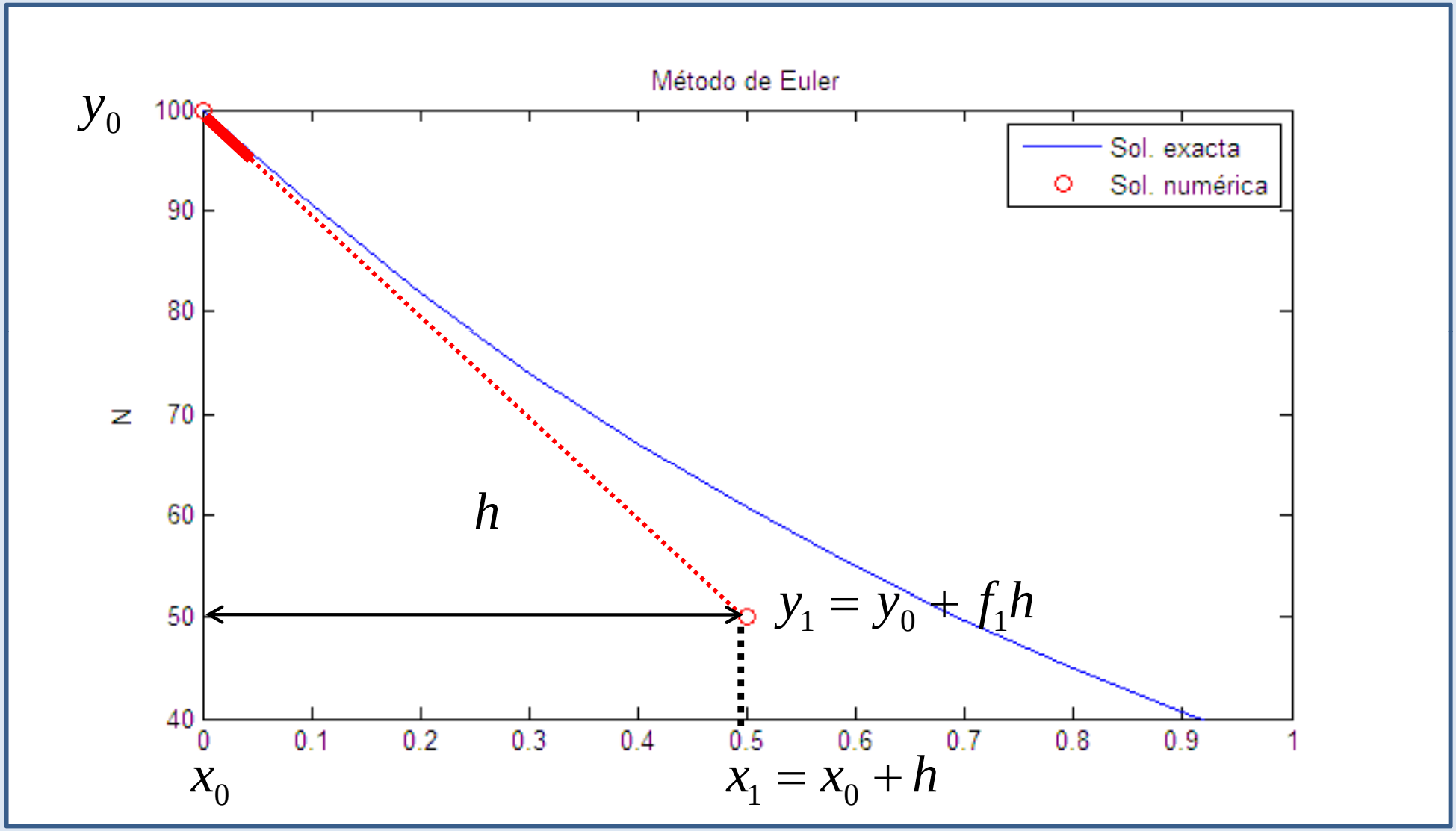
$$f_1 = f(x_j, y_j; k) = \frac{dy}{dx}$$



El método de Euler

2) Calcular y_{j+1} como: $y_{j+1} = y_j + f_1 h$

3) Calcular x_{j+1} como: $x_{j+1} = x_j + h$



Método de Euler. Implementación.

Cuando escribimos la rutina para el método de Euler queremos que nos sirva para cualquier problema del tipo:

$$f(x, y; k) = \frac{dy}{dx}$$

```
26      function [x,y]=feuler(fun,x0,xf,y0,Nh)
27 -    h=(xf-x0)/Nh;
28 -    x=zeros(1,Nh+1);
29 -    y=zeros(1,Nh+1);
30 -    x(1)=x0;
31 -    y(1)=y0;
32 -    for j=2:1:Nh+1
33 -        f = fun(x(j-1),y(j-1));
34 -        y(j) = y(j-1)+h*f;
35 -        x(j) = h+x(j-1);
36 -    end
37 -    end
```

Para ello hacemos que uno de los argumentos de la rutina sea una función anónima.

Declaración de una función anónima.

Para declarar la función anónima ***nombre*** a partir de otra función ***función*** se escribe:

nombre = @(variable1, variable2,...) función(variable1, variable2, ..., parámetro1, parámetro2,....)

nombre: la función anónima que queremos crear a partir de la función ***función***.

variable1,variable2,...: las variables de las que depende la función anónima.

parámetro1,parámetro2,...: variables de las que depende la función ***función*** pero que queremos que tengan valores fijos en la función anónima.

En Matlab, el símbolo para una función anónima es:

Uso de una función anónima

Las funciones anónimas se usan para poder usar una función como una variable de entrada al llamar a una rutina.

Ejemplo:

k = 1;
g = @(t,y) –k*y;

Define una función anónima *g* depende de x e y que vale -1*y.

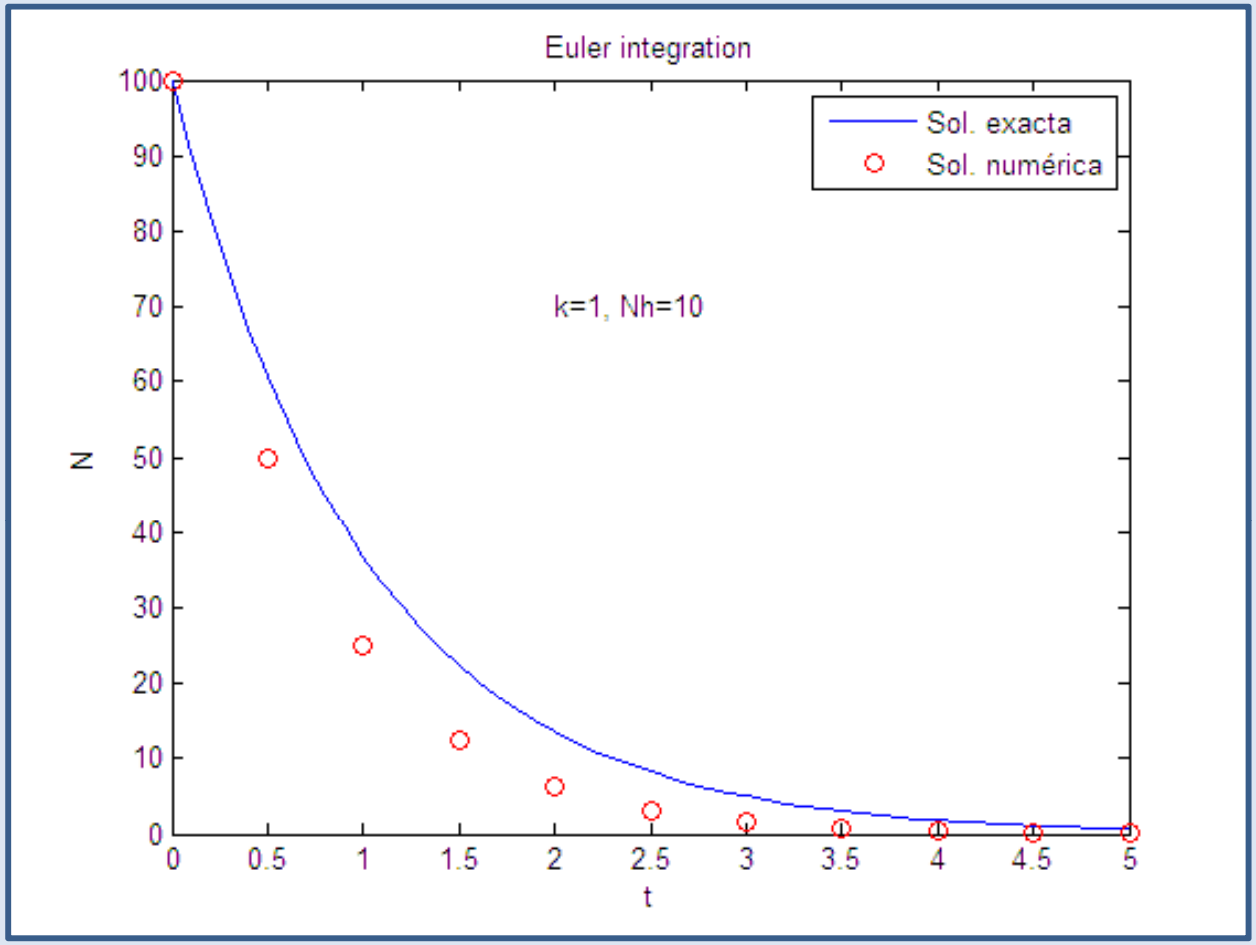
Al escribir ***feuler(g,t0,tf,y0,yf)*** le indicamos a la rutina *feuler* que la función *fun* dentro de su código ha de ser sustituida por -1*y

```
9     %% Aquí encuentro la solución numérica por el método de Euler
10 -  k=1;
11 -  g = @(t,y) -k*y;          % de
12 -  y0=100;
13 -  t0=0;
14 -  tf=5;
15 -  Nh=10;
16 -  [t,y]=feuler(g,t0,tf,y0,Nh);  % llamada a la rutina del método de Euler
17    %% Aquí encuentro la solución exacta y la represento junto con la numérica
18 -  tp=0:0.1:5;
19 -  yp=y0*exp(-k*tp);
20 -  plot(tp,yp,t,y,'o','MarkerEdgeColor','r');
21 -  xlabel('t');
22 -  ylabel('N');
23 -  title('Método de Euler');
```

Aquí defino una función anónima

Aquí uso la función anónima como una variable

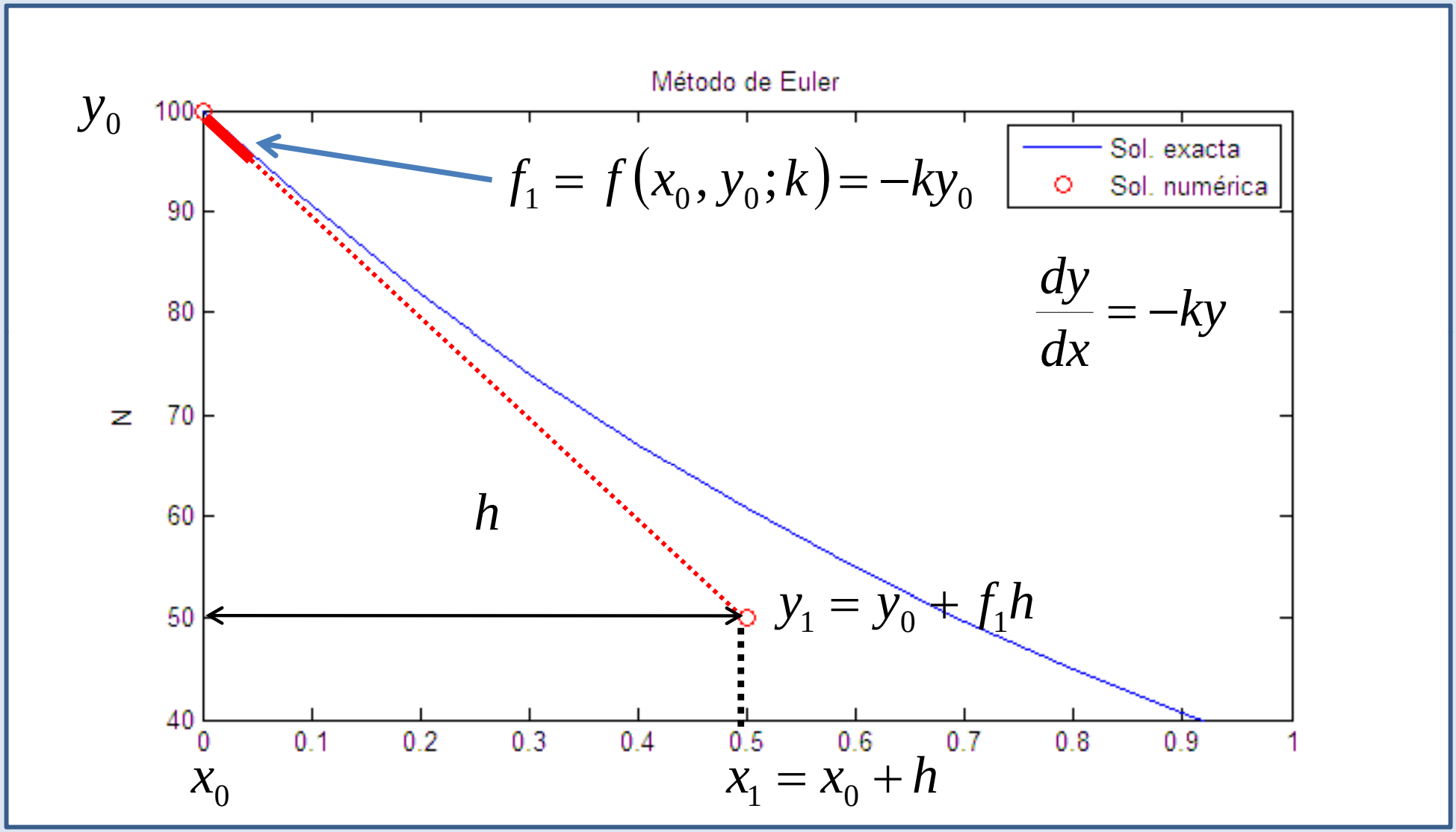
Resultado para nuestro problema



Vemos que la solución que hemos obtenido no coincide con la exacta.

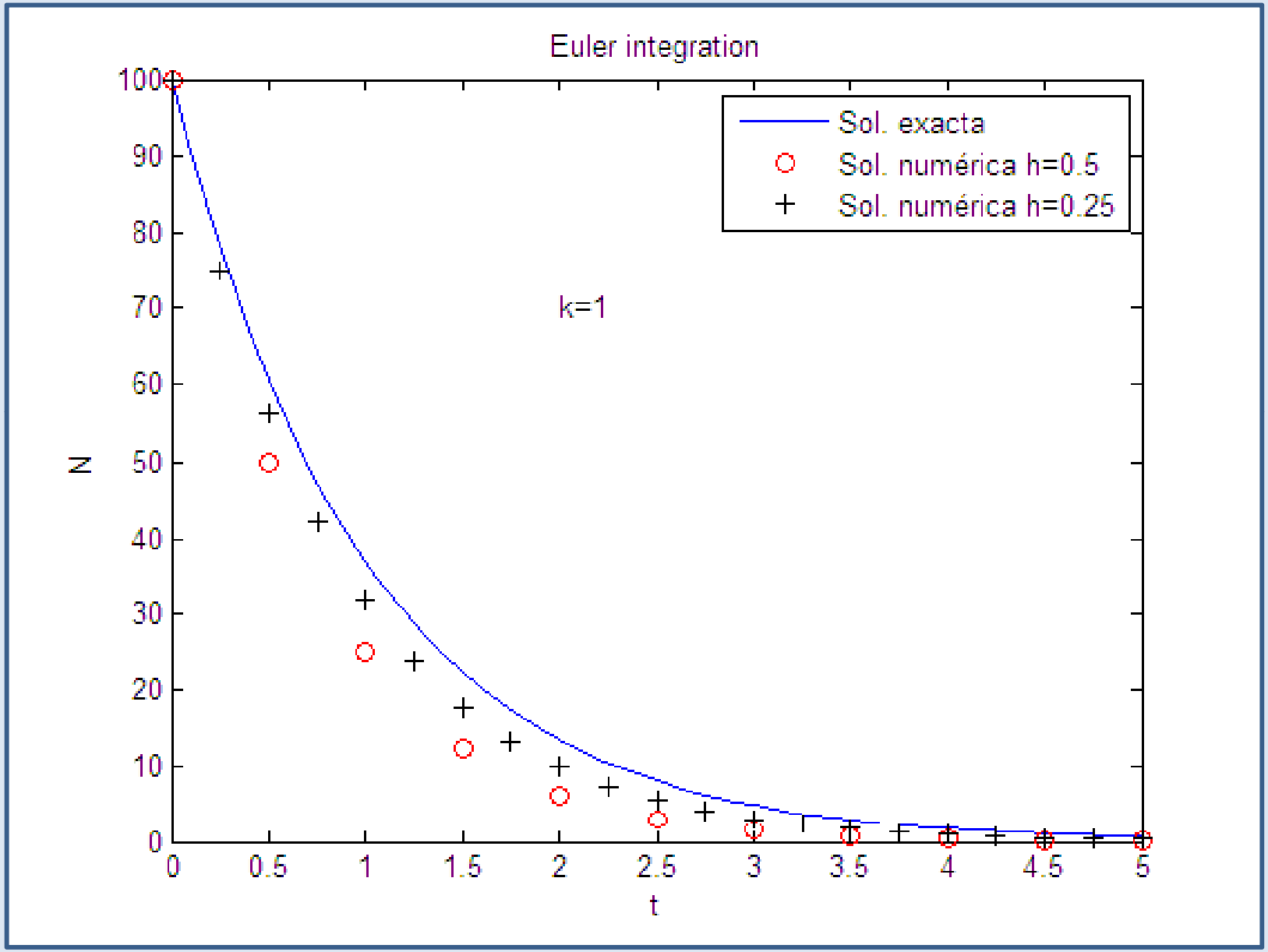
Representación gráfica del método de Euler

Para el primer paso:



Efecto del tamaño del paso.

Una forma de mejorar la precisión es disminuir el tamaño del paso h.



Contribuciones al error

- **El error de redondeo del ordenador.**

  Se debe a que el ordenador no almacena los números con precisión infinita.

- **Error de truncamiento del método numérico.**

Se debe a que el método numérico no resuelve la EDO de forma exacta.

Si $y_{exact}(x_p)$ es el valor exacto de la solución de nuestro problema en el punto P, se define el error local de truncamiento en P como:

$$e(x_p) = \left| y_{exact}(x_p) - y_{numérico}(x_p) \right|$$

De lo que hemos dicho, el error de truncamiento debe disminuir cuando disminuimos el paso h.

Error de redondeo

El ordenador almacena los números como mantisa x exponente.

Por ejemplo: 138.516 → **0.138516×10^{3}**

mantisa **exponente**

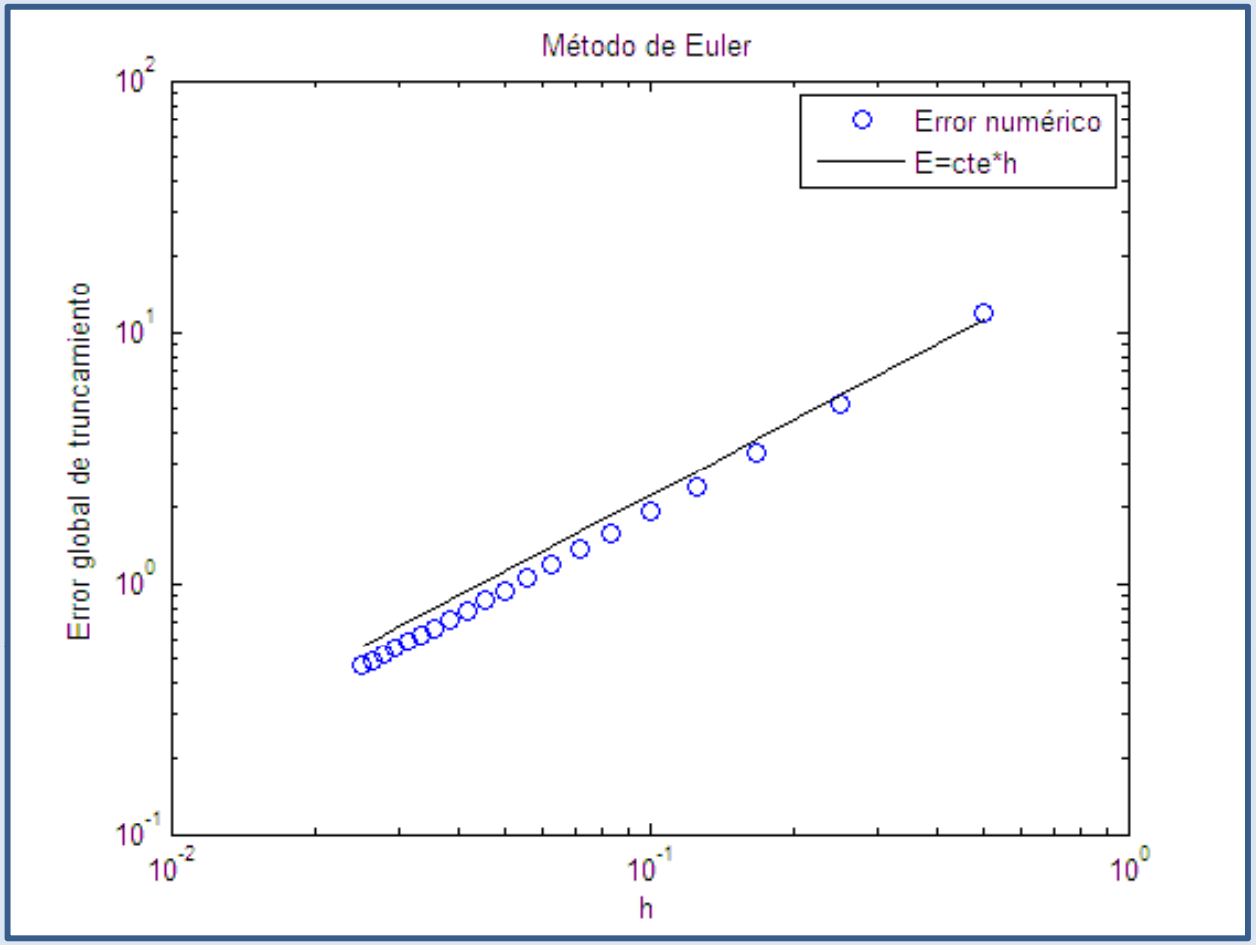
El error relativo con el que se almacenan un número en el ordenador viene determinado por la cantidad de dígitos que se almacenan de la mantisa.

Se muestra en Matlab con el comando *eps*.

Esto significa que en capa paso de un método numérico, no tiene sentido pedir más precisión relativa que *eps.*

En un cálculo bien hecho, el error de redondeo es despreciable frente al error de truncamiento.

Error del método de Euler.



El máximo error de truncamiento del método de Euler es proporcional al tamaño del paso h.

Si reducimos el paso a la mitad, reducimos el error a la mitad

Orden de un método numérico

Se dice que un método de integración es de orden n si el error de truncamiento E depende del paso h como:

$$E \propto h^n$$

Esto significa que el método de Euler es un método de primer orden.

Valor del error en el primer paso del método de Euler:

Valor exacto de la función y en x1.

Valor dado por el método de Euler para la función y en x_1.

$$e(x_1) = |y(x_1) - [y_0 + f(x_0, y_o)h]$$

¿Por qué el método de Euler es de primer orden?

Desarrollando en serie de Taylor

$$y(x_1) = y_0 + y'(x_o)h + C_1 h^2 = y_0 + f(x_o, y_o)h + C_1 h^2$$

Donde C_1 es una constante que depende de la segunda derivada de la función f en el intervalo (x_0, x_1).

Y el error cometido en la primera iteración es:

$$e(x_1) = |y(x_1) - [y_0 + f(x_0, y_o)h]|$$

$$e(x_1) = |C_1| h^2$$

¿Por qué el método de Euler es de primer orden?

Los errores se van acumulando en cada iteración.

Si resolvemos el problema en el intervalo (x_0,x_f) con paso h, daremos N pasos, con

$$N = \frac{x_f - x_o}{h}$$

Y el máximo error de truncamiento será del orden de:

$$Max[e] \approx \text{Cte} \times Nh^2 \approx \text{Cte} \times (x_f - x_o)h$$

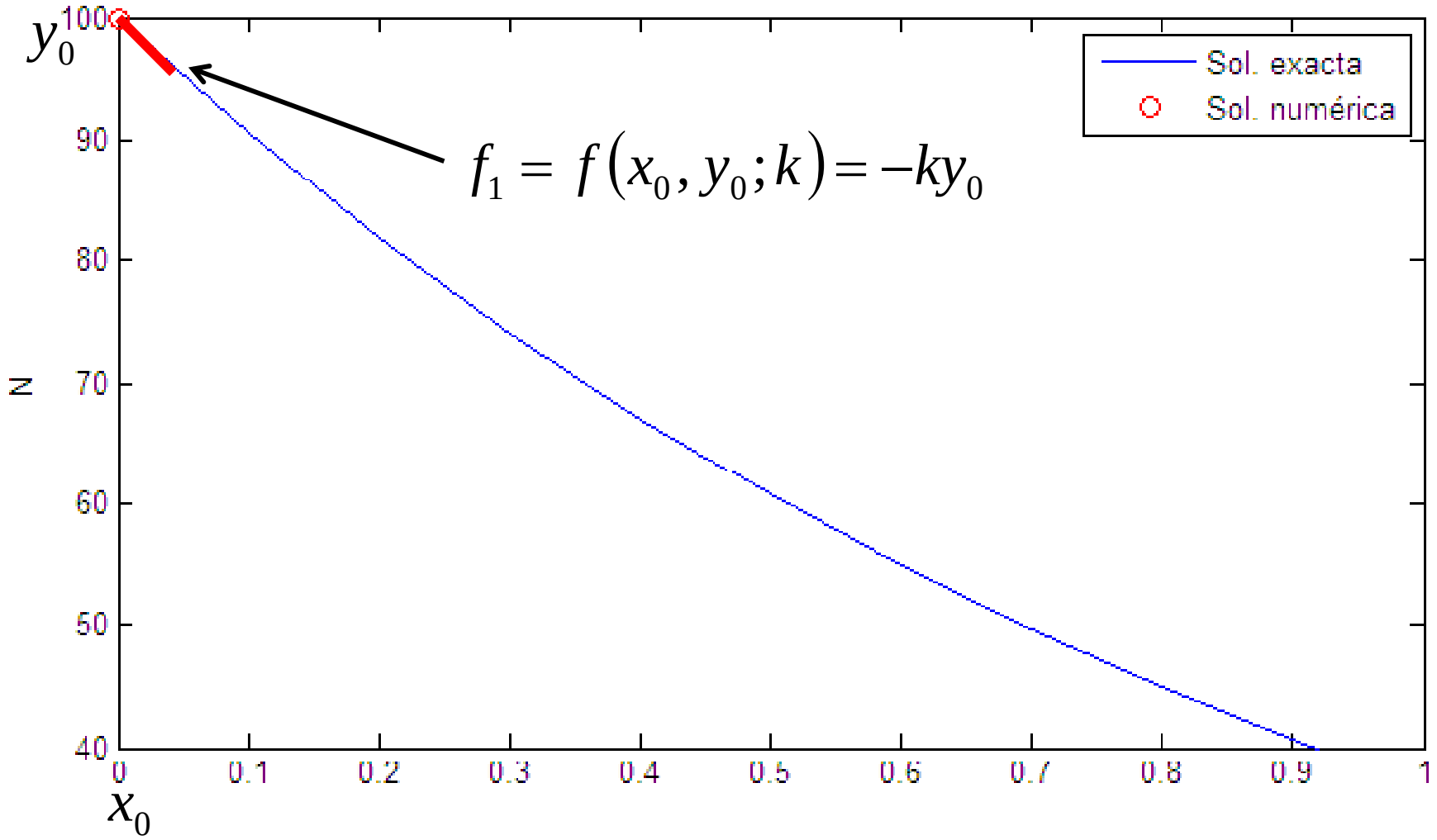
Cuanto mayor sea el orden de un método, mayor puede ser el paso para obtener el mismo error de truncamiento.

Para obtener métodos de orden superior, tenemos que hacer más evaluaciones de la derivada en cada intervalo.

El método del salto de rana (Leapfrog)

1) Dado x_j, y_j, calcular la derivada en ese punto.

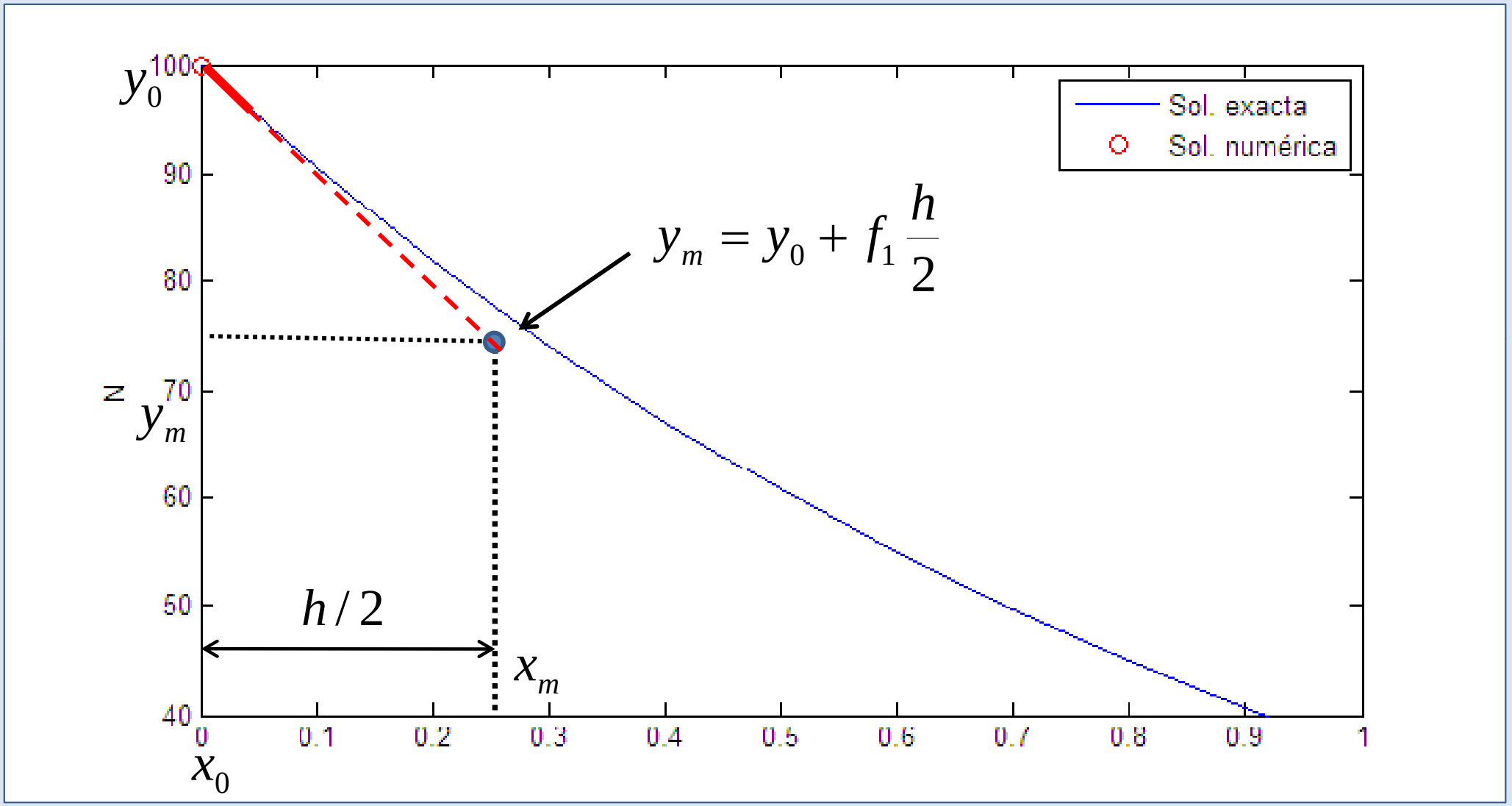
$$f_1 = f\left(x_j, y_j; k\right)$$



El método del salto de rana (Leapfrog)

2) Calcular un punto x_m, y_m a la mitad del paso.

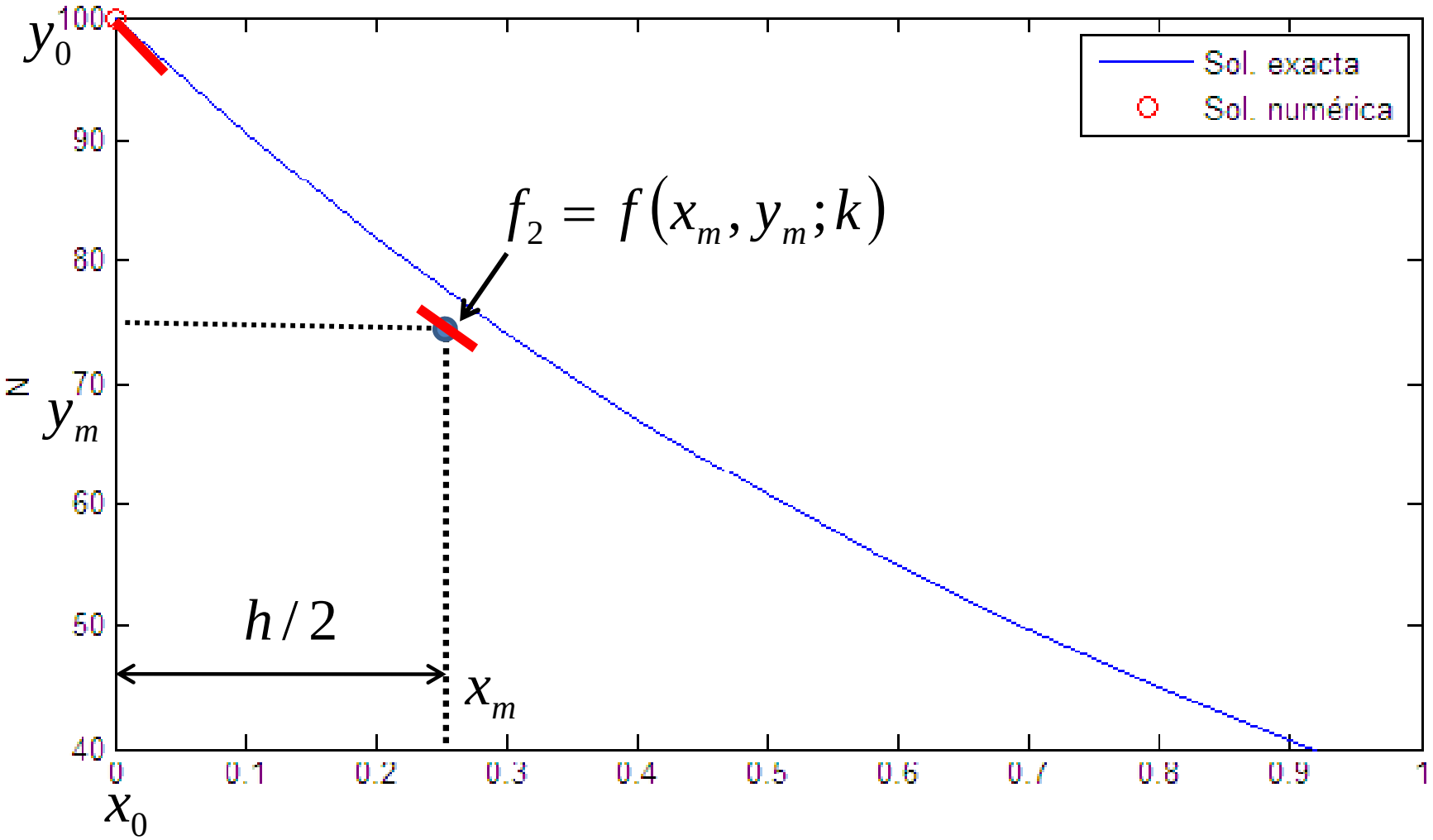
$$y_m = y_j + f_1 \frac{h}{2} \qquad x_m = x_j + \frac{h}{2}$$



El método del salto de rana (Leapfrog)

3) Calcular la derivada en el punto intermedio.

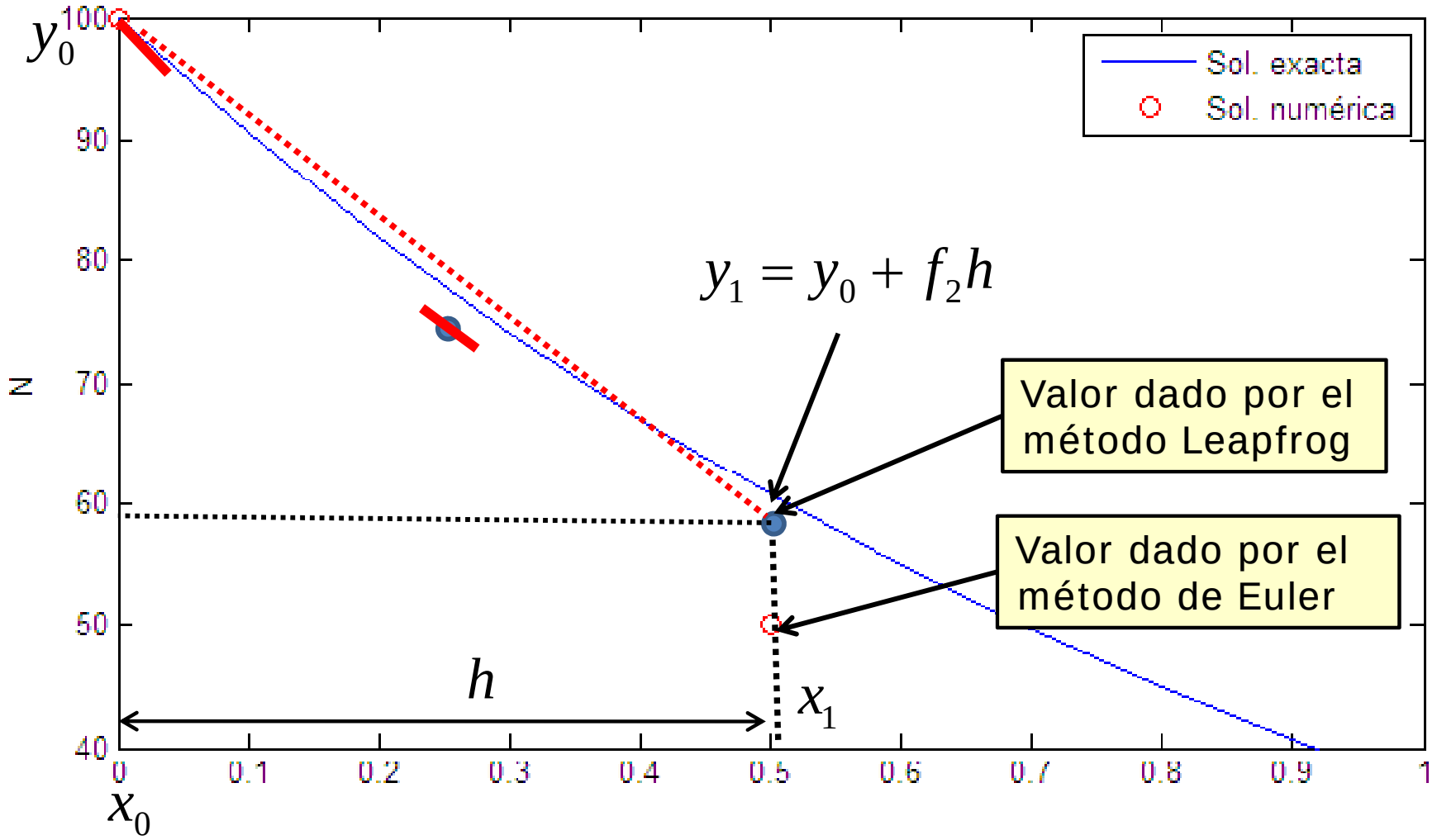
$$f_2 = f(x_m, y_m; k) = f\left(x_o + \frac{h}{2}, y_o + f_1 \frac{h}{2}\right)$$



El método del salto de rana (Leapfrog)

4) Calcular x_{j+1} e y_{j+1} usando la derivada en el punto intermedio

$$y_{j+1} = y_j + f_2 h$$
$$x_{j+1} = x_j + h$$



Ecuaciones del método Leapfrog.

Escrito para un problema general, como: $\frac{dy}{dx} = f(x, y; k)$

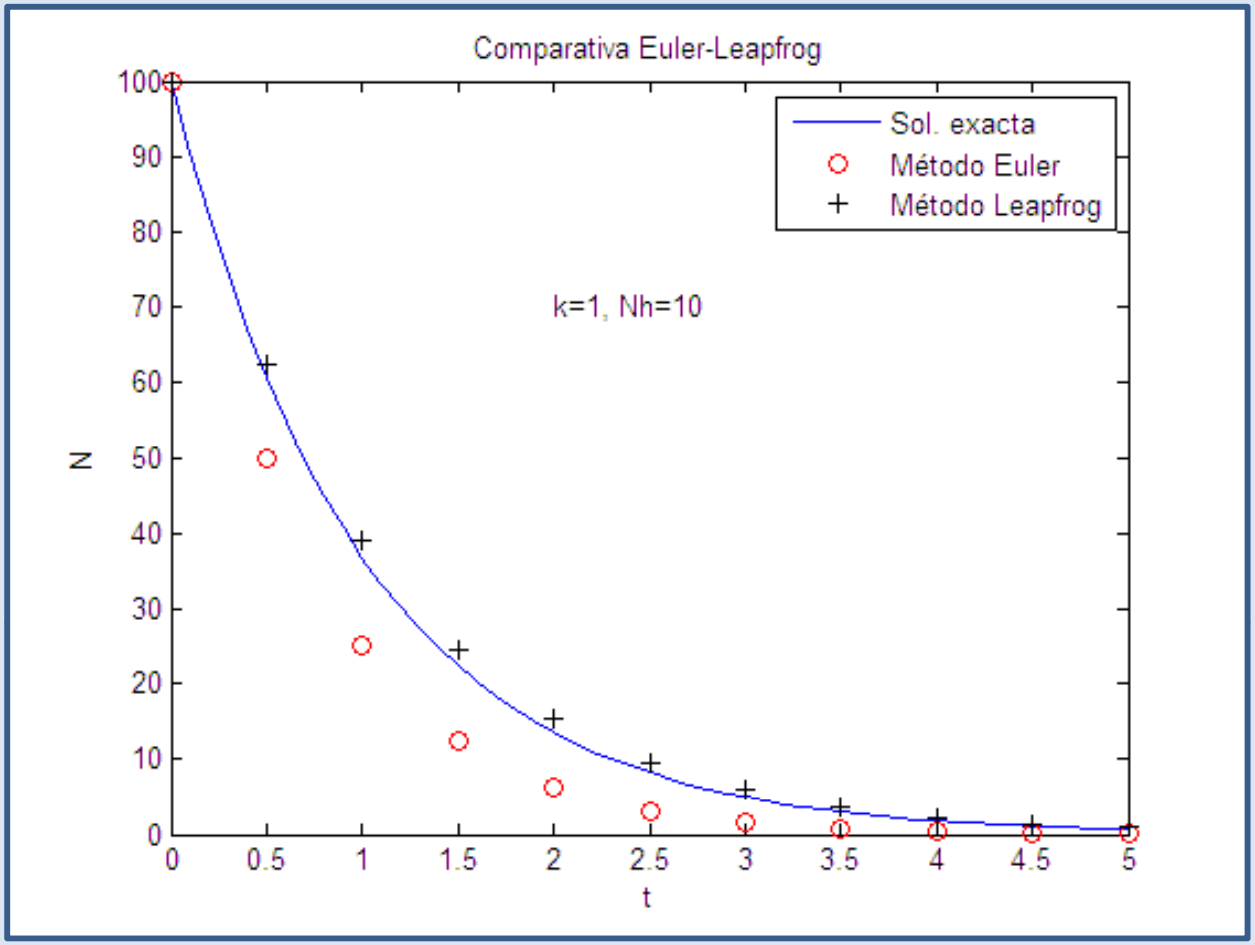
Cada paso queda:

1) Dado x_j, y_j, calcular la derivada en ese punto.	$f_1 = f(x_j, y_j; k)$
2) Calcular un punto x_m, y_m a la mitad del paso.	$y_m = y_j + f_1 \frac{h}{2} \qquad x_m = x_j + \frac{h}{2}$
3) Calcular la derivada en el punto intermedio.	$f_2 = f(x_m, y_m; k) = f\left(x_o + \frac{h}{2}, y_o + f_1 \frac{h}{2}\right)$
4) Calcular x_{j+1} e y_{j+1} usando la derivada en el punto intermedio	$y_{j+1} = y_j + f_2 h$ $x_{j+1} = x_j + h$

Método Leapfrog: implementación en Matlab.

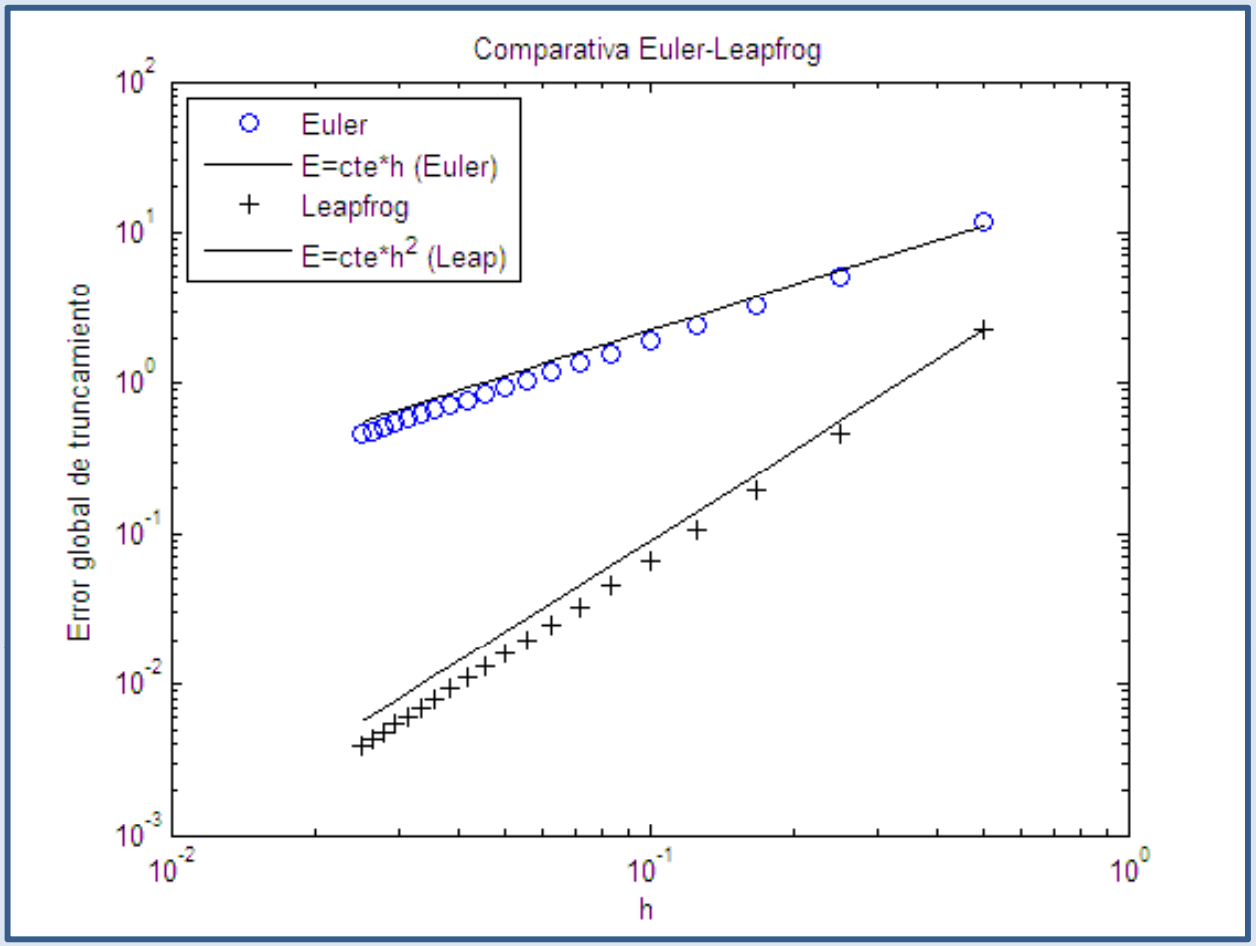
```
26      function [x,y]=fLeapfrog(fun,x0,xf,y0,Nh)
27 -    h=(xf-x0)/Nh;
28 -    x=zeros(1,Nh+1);
29 -    y=zeros(1,Nh+1);
30 -    x(1)=x0;
31 -    y(1)=y0;
32 -    for j=2:1:Nh+1
33 -        f1 = fun(x(j-1),y(j-1));
34 -        xm = x(j-1)+h/2;
35 -        ym = y(j-1)+h/2*f1;
36 -        fm = fun(xm,ym);
37 -        y(j) = y(j-1)+h*fm;
38 -        x(j) = h+x(j-1);
39 -    end
40 -    return
```

Comparativa Euler-Leapfrog



- Hemos conseguido mucha más precisión con el mismo paso.
- El precio que hemos pagado es que hacemos dos evaluaciones de f(x,y) en cada paso.

Comparativa Euler-Leapfrog (errores)



- En el método Leapfrog el error depende del cuadrado del paso h.
- Si reducimos el paso a la mitad, el error se reduce a la cuarta parte.
- Se dice que el método Leapfrog es un **método de segundo orden**.

¿Por qué el método Leapfrog es de 2º orden?

- Para el primer paso:

$$e(x_1) = |y(x_1) - [y_0 + f(x_m, y_m)h]|$$

- (x_m, y_m) es el punto intermedio . En él, la derivada vale aprox.:

$$f(x_m, y_m; k) = f\left(x_o + \frac{h}{2}, y_o + f_1\frac{h}{2}\right) \approx f(x_0, y_0) + f'(x_0, y_0)\frac{h}{2}$$

- $y(x_1)$ es el valor exacto de la solución en el punto x_1. Vale aprox.

$$y(x_1) = y_0 + y'(x_0)h + \frac{1}{2}y''(x_0)h^2 + Ch^3$$

$$y(x_1) = y_0 + f(x_o, y_o)h + f'(x_0, y_0)\frac{h^2}{2} + Ch^3$$

- Luego, sustituyendo en la primera eq.

$$e(x_1) = |C|h^3$$

El método de Runge-Kutta

• Se conoce de forma general como método de Runge-Kutta todas las generalizaciones del método Leapfrog en las que se hacen varias evaluaciones de la función f(x,y) en cada paso h.

• Por ejemplo, el método Leapfrog es también un método de Runge-Kutta de segundo orden.

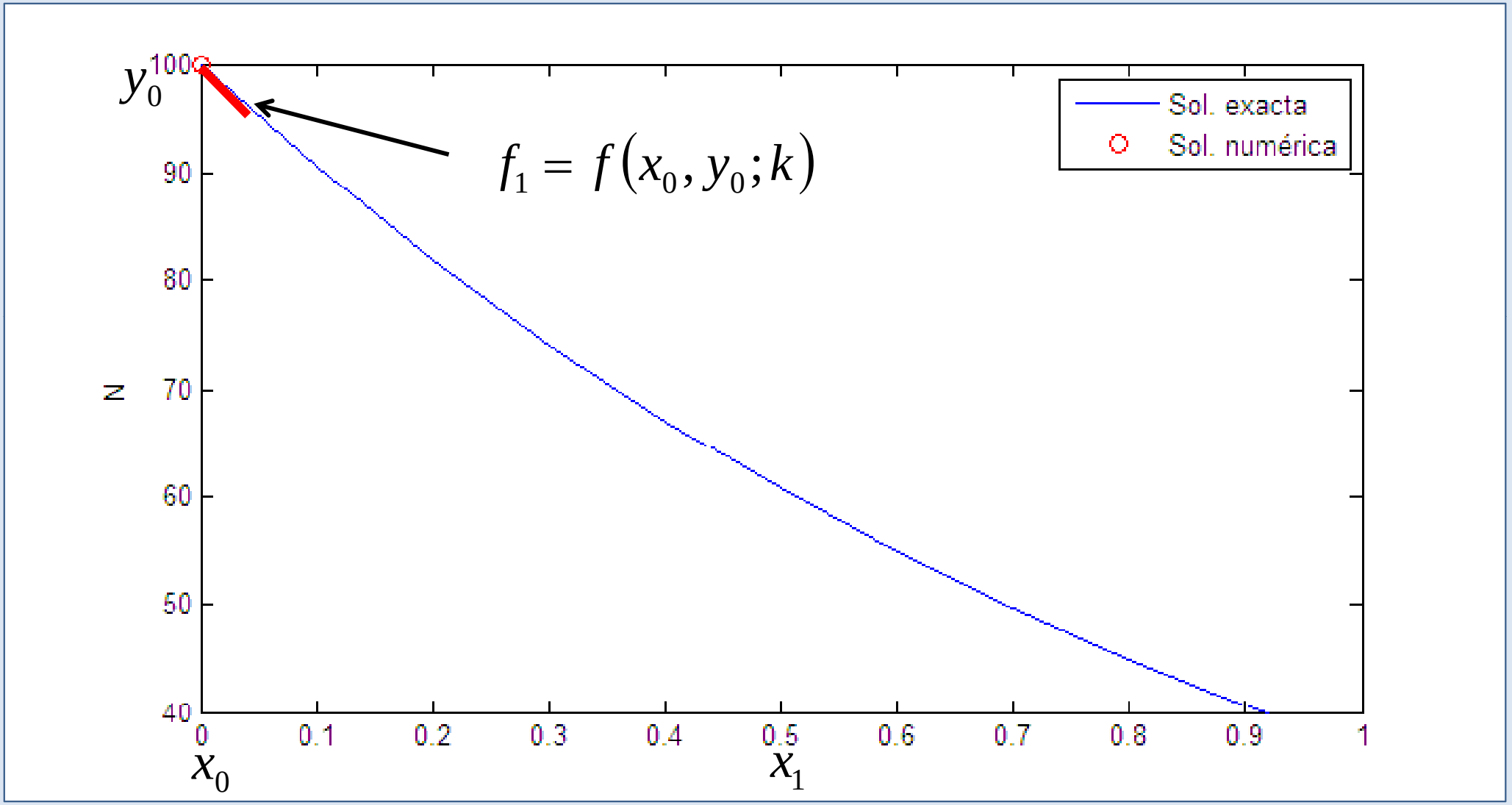
• El método de Runge-Kutta de cuarto orden **es el método serio más sencillo** para hacer integración numérica.

• Se recomienda **usarlo como primer método de prueba**, pero no tiene por qué ser el método que dé más precisión ni el más eficiente computacionalmente para un problema dado.

Runge-Kutta de 4º orden

1) Dado x_j, y_j, calcular la derivada en ese punto.

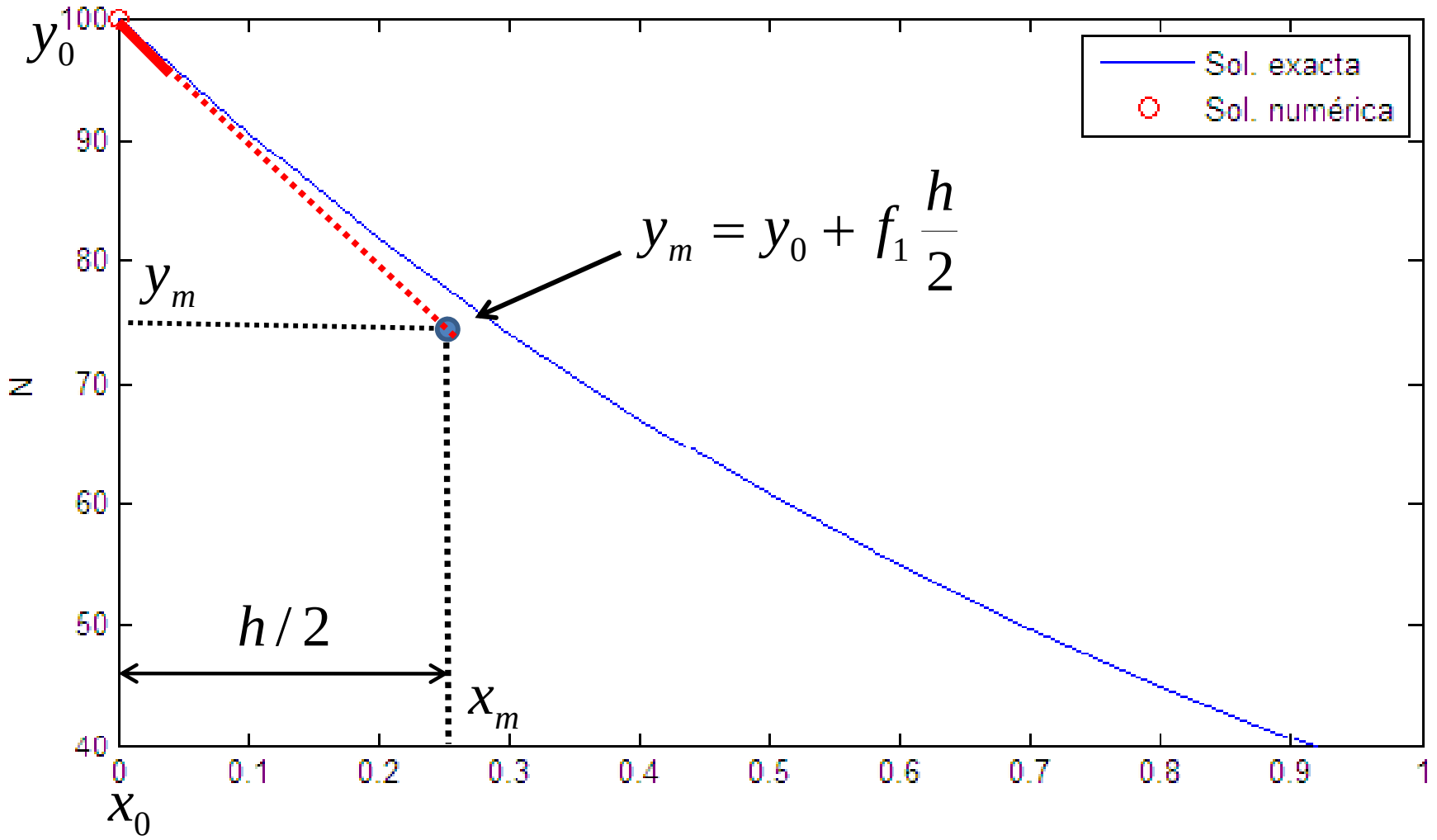
$$f_1 = f(x_j, y_j)$$



Un método de Runge-Kutta de 4º orden

2) Encontrar un punto a mitad del paso.

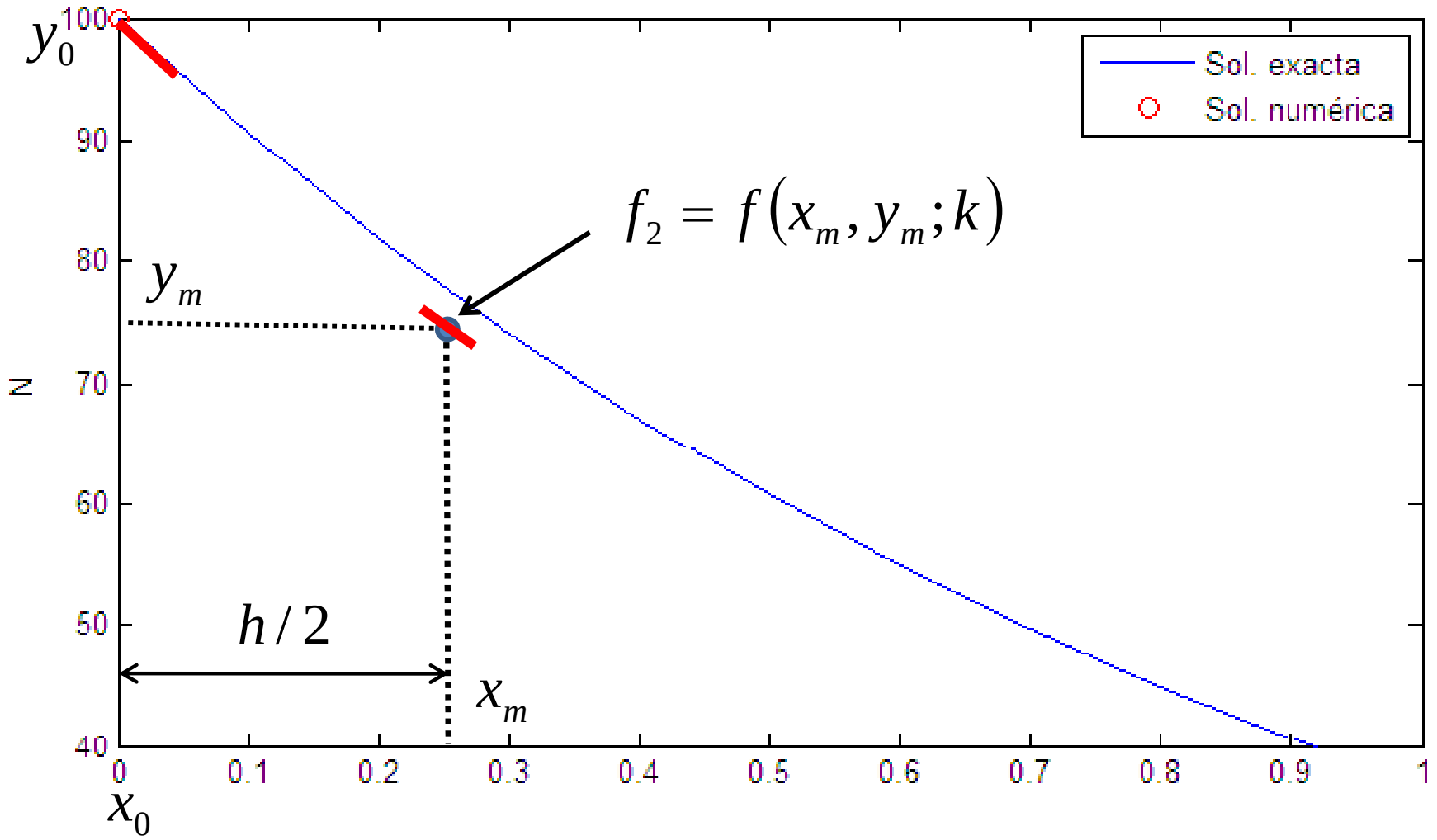
$$y_m = y_j + f_1 \frac{h}{2} \qquad x_m = x_j + \frac{h}{2}$$



Un método de Runge-Kutta de 4º orden

2) Calcular la derivada f_2 en ese punto.

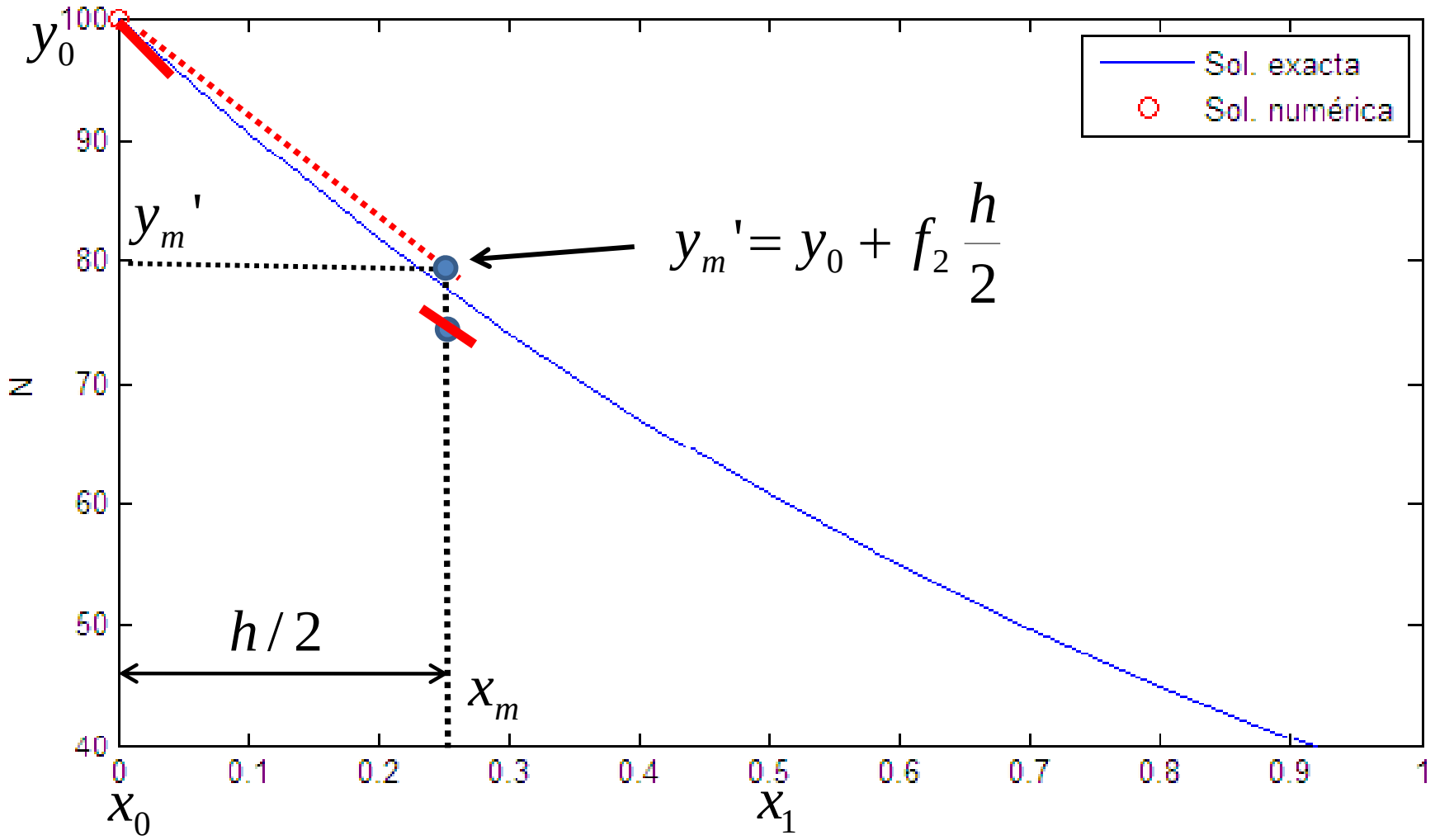
$$f_2 = f\left(x_j + \frac{h}{2}, y_j + f_1 \frac{h}{2}\right)$$



Runge-Kutta de 4º orden

Usar la derivada f_2 para mejorar la estimación del punto intermedio.

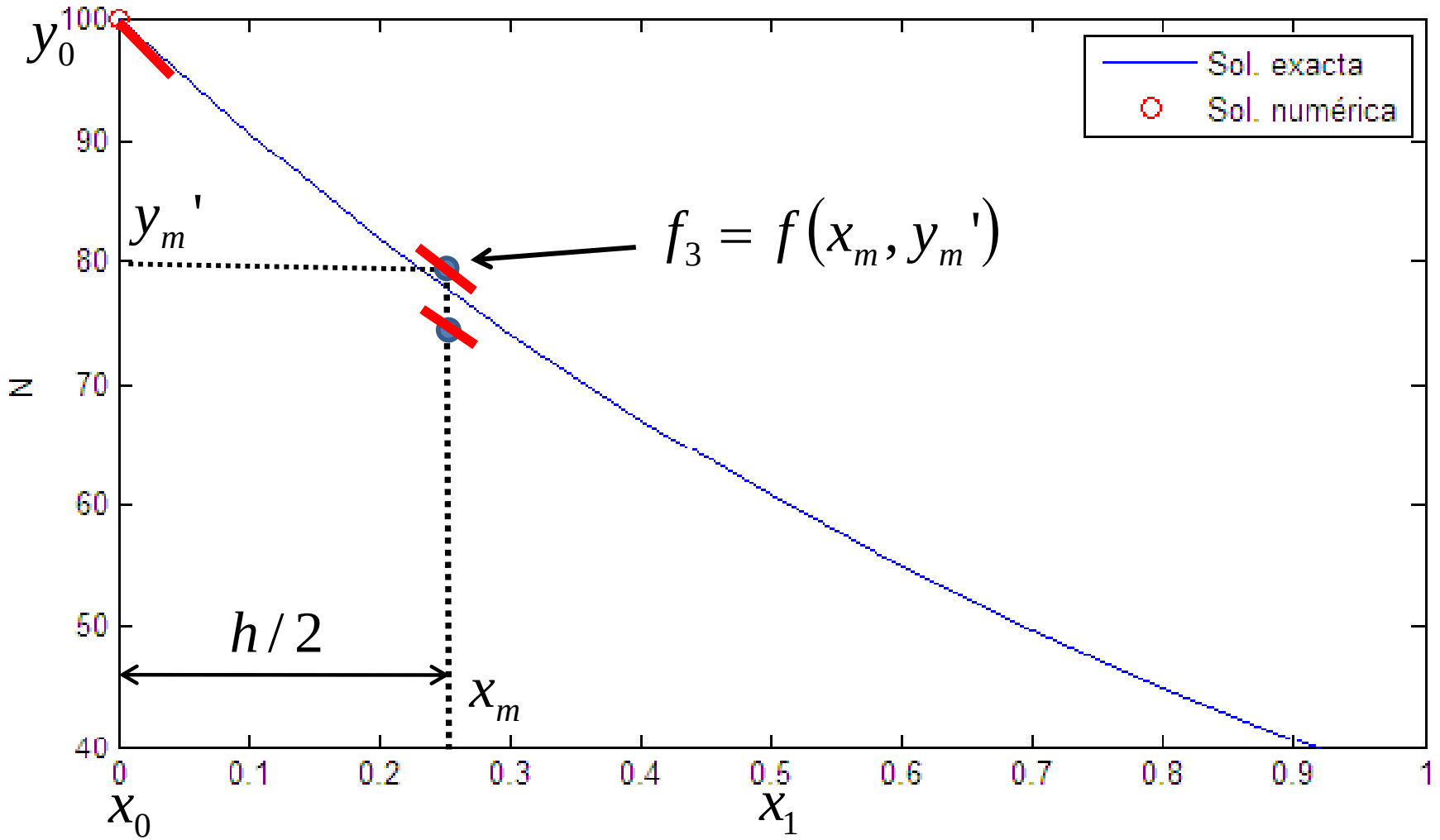
$$y_m' = y_j + f_2 \frac{h}{2} \qquad x_m = x_j + \frac{h}{2}$$



Runge-Kutta de 4º orden

3) Calcular la derivada f_3 en el nuevo punto intermedio

$$f_3 = f\left(x_j + \frac{h}{2}, y_j + f_2\frac{h}{2}\right)$$

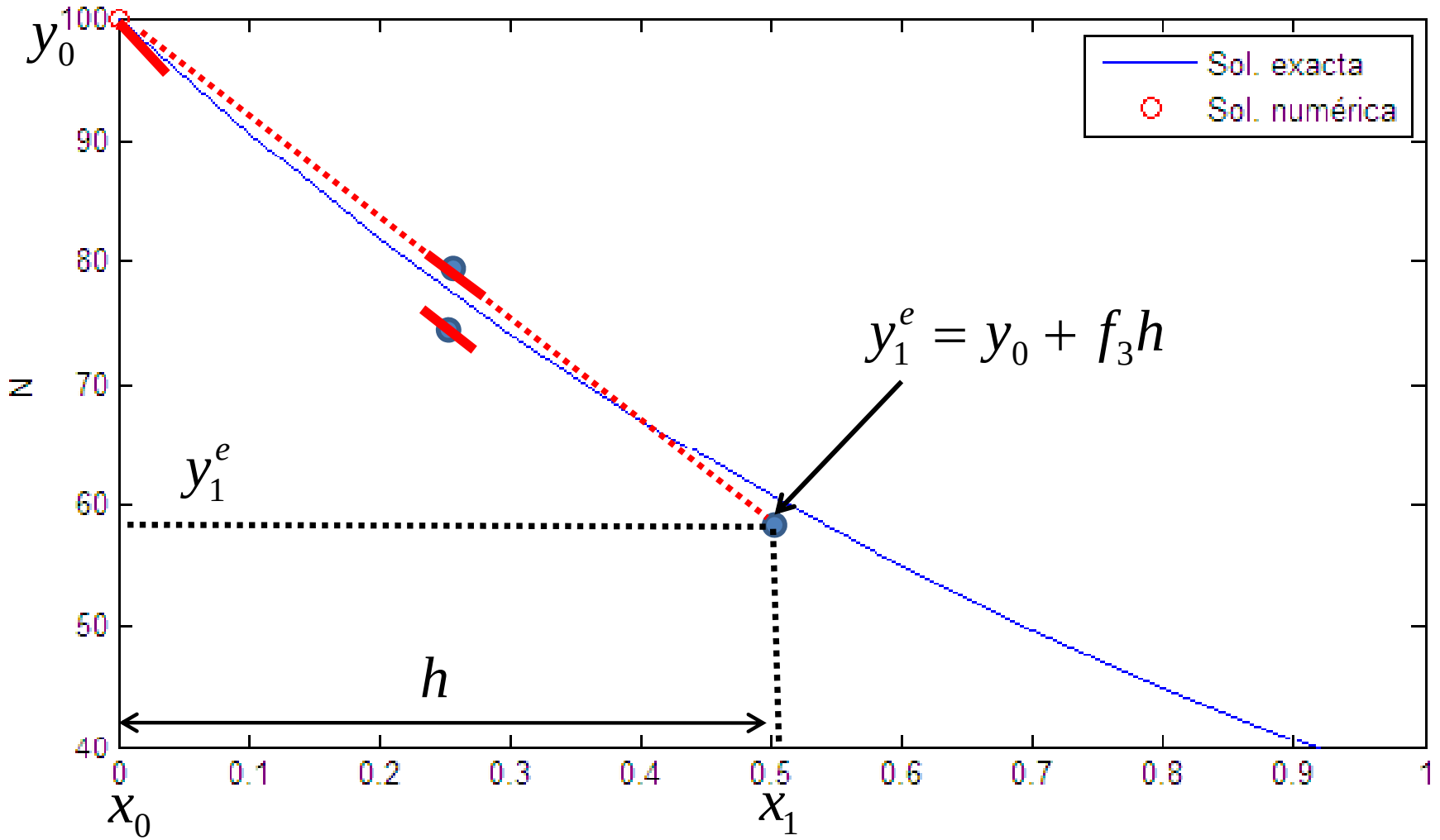


Runge-Kutta de 4º orden

Usar la derivada f_3 para calcular una primera estimación del punto final x_{j+1}, y_{j+1}.

$$x_1 = x_0 + h$$

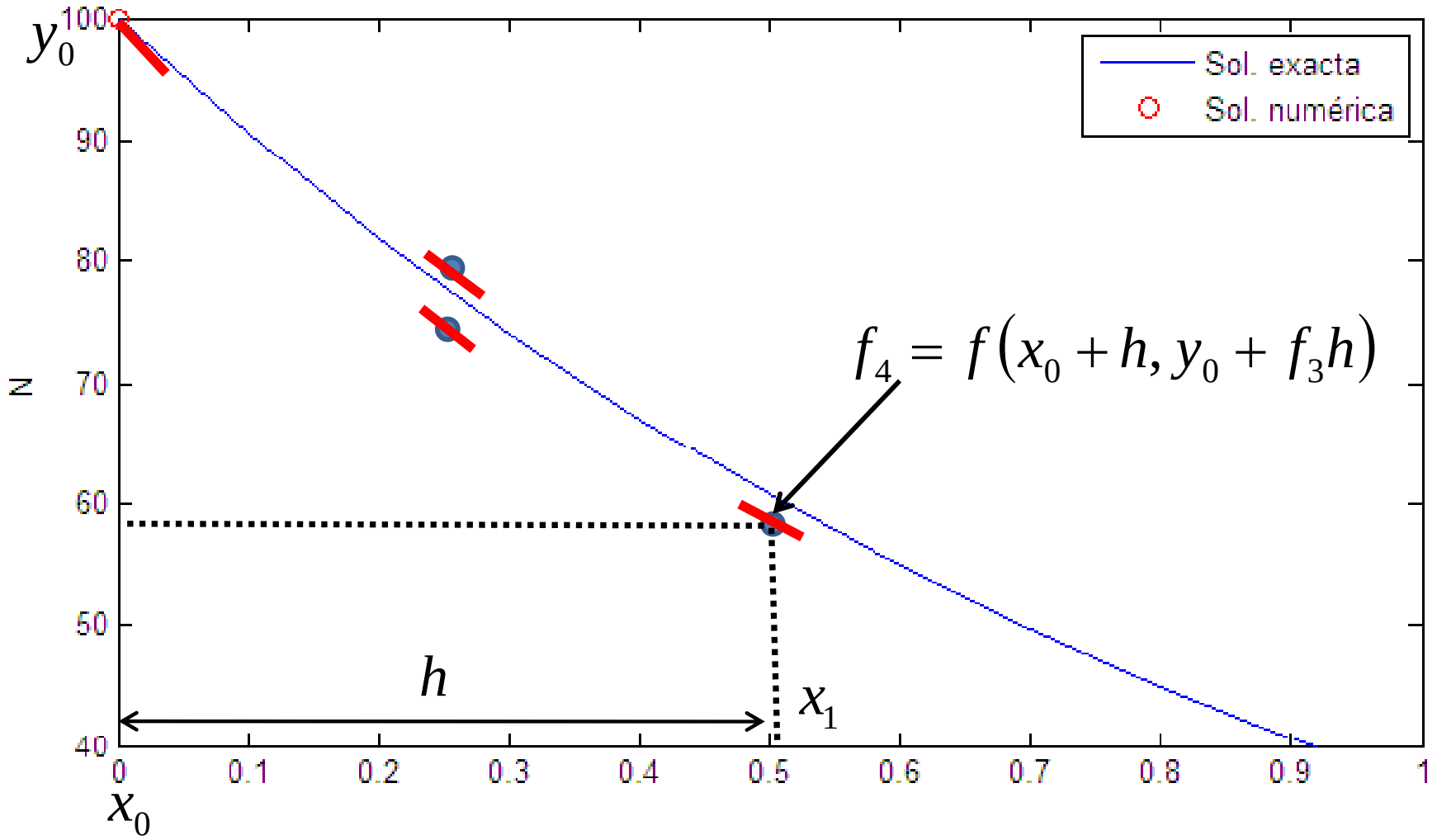
$$y_1^e = y_0 + f_3 h$$



Runge-Kutta de 4º orden

Calcular la derivada f_4 en la estimación del punto final.

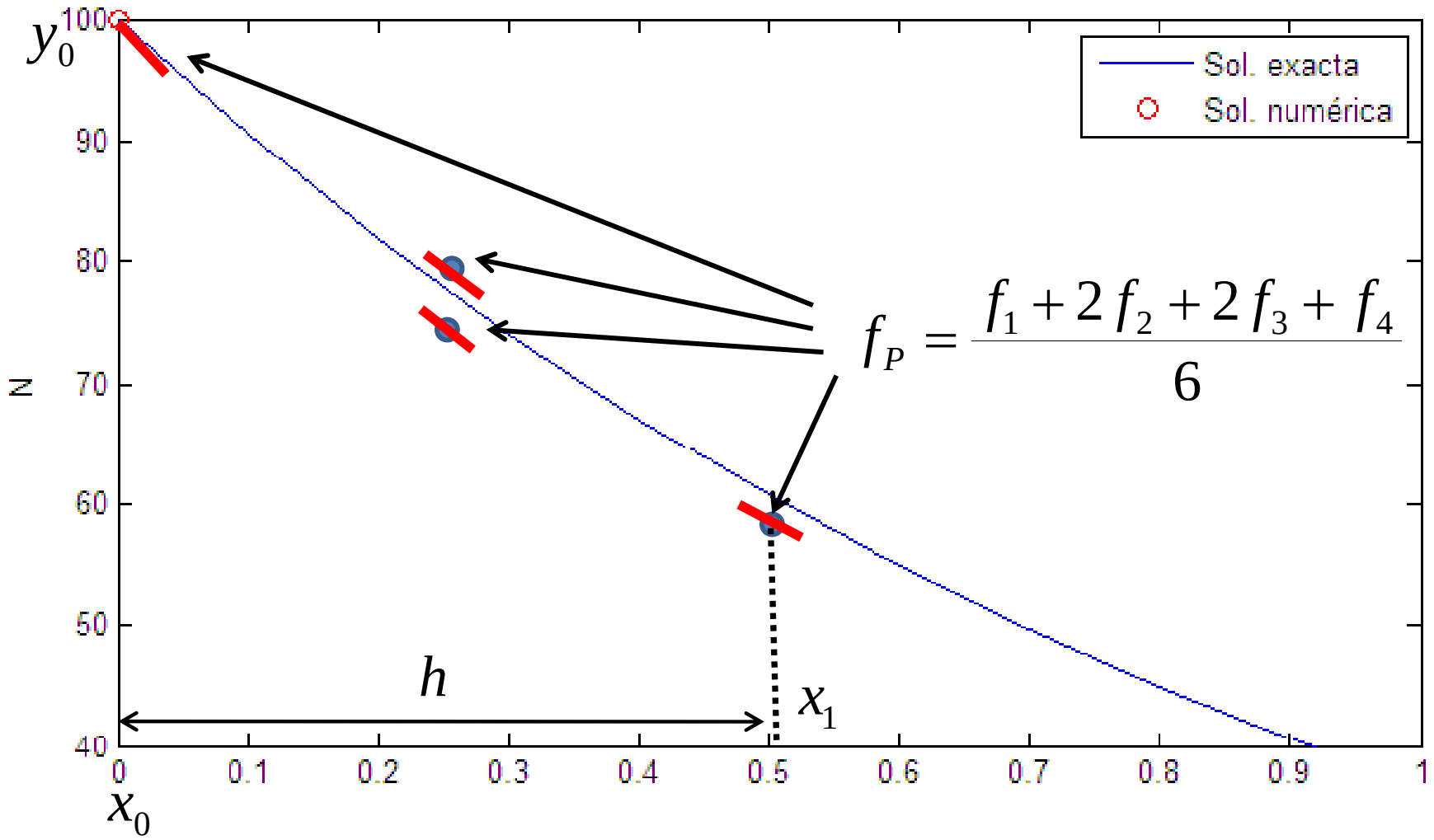
$$f_4 = f\left(x_j + h, y_j + f_3 h\right)$$



Runge-Kutta de 4º orden

Hacer una estimación de la derivada promediando f_1, f_2, f_3 y f_4 como:

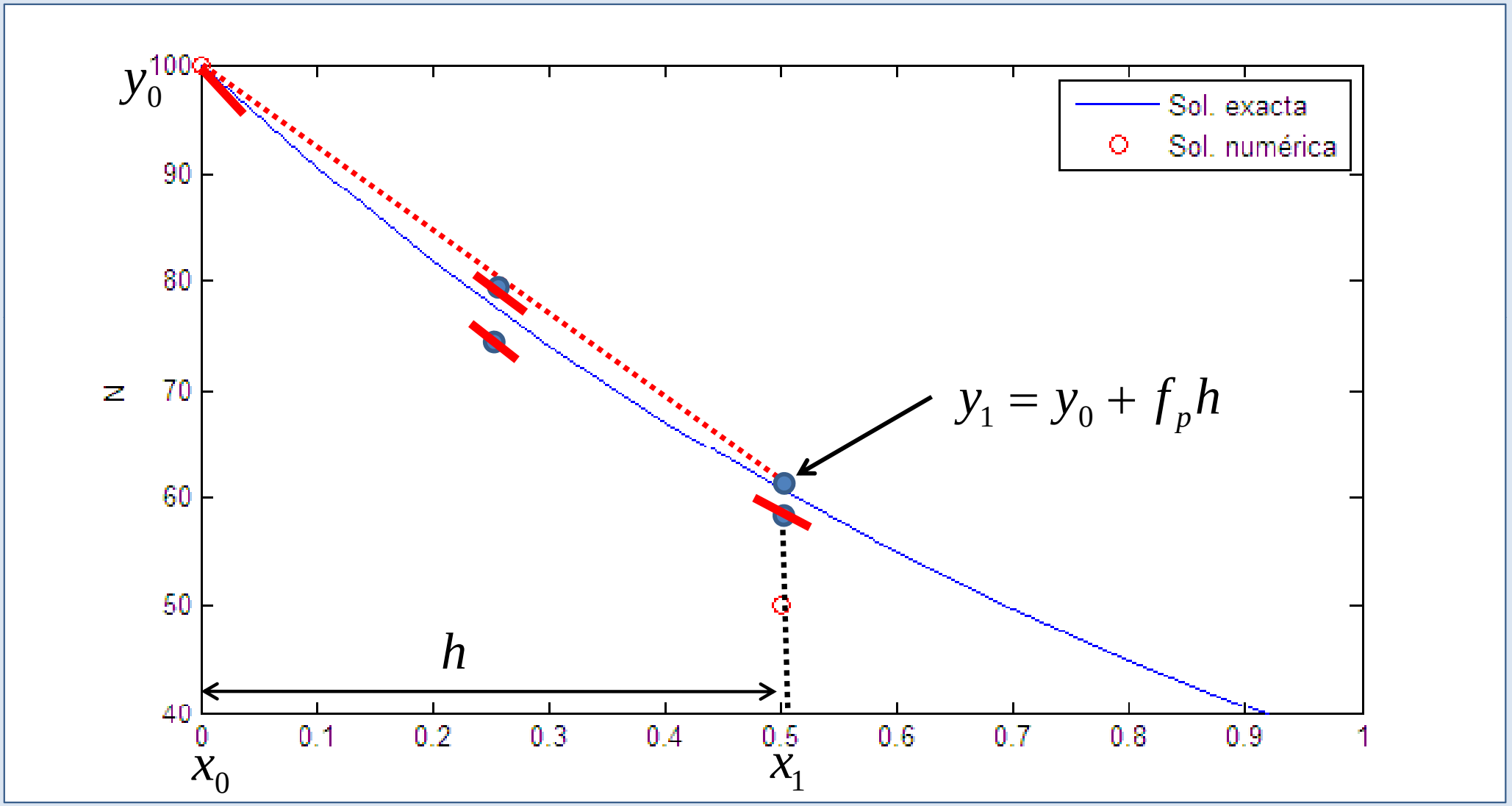
$$f_P = \frac{f_1 + 2f_2 + 2f_3 + f_4}{6}$$



Runge-Kutta de 4º orden

Usar f_p para calcular el punto final

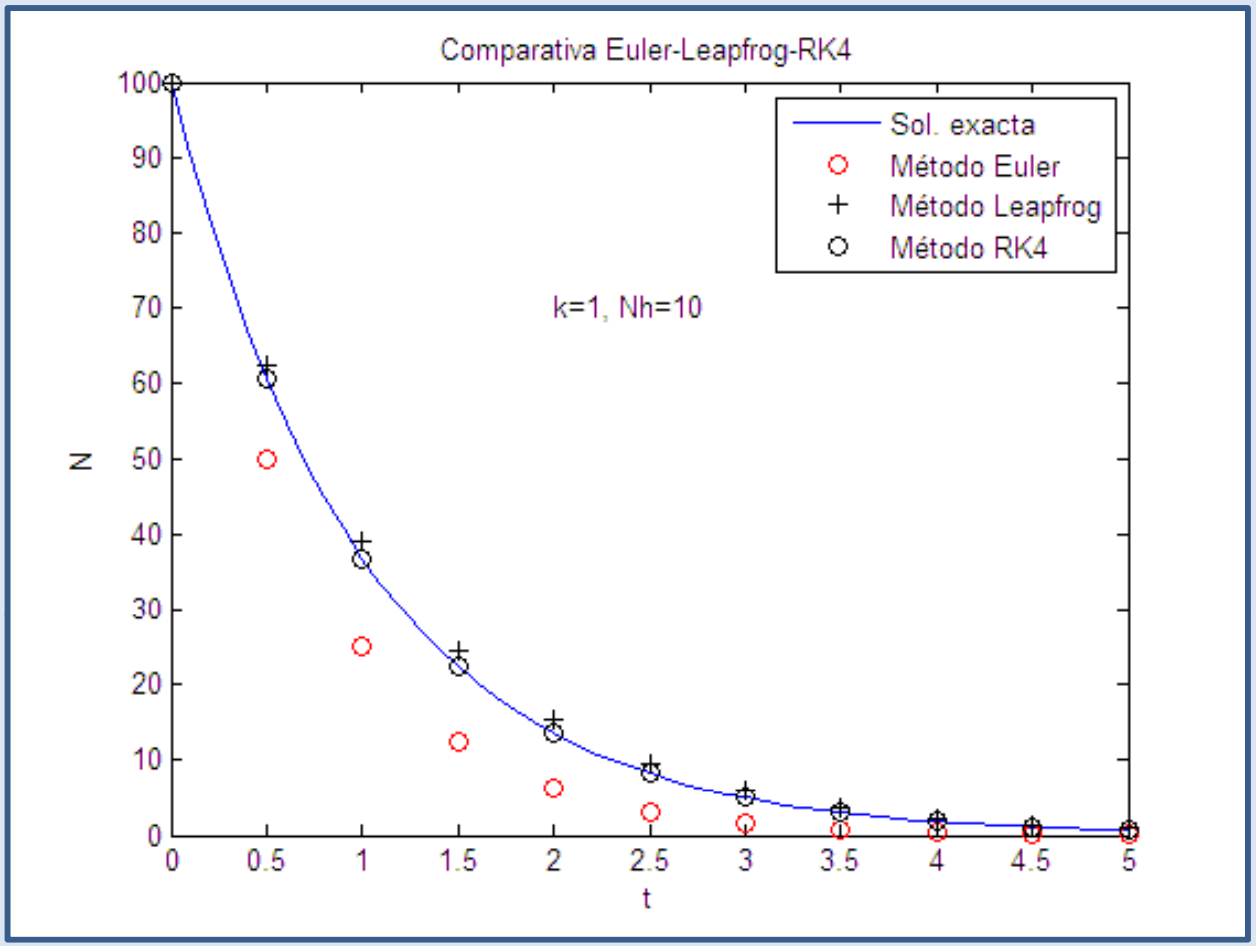
$$y_{j+1} = y_j + hf_p$$



Método RK4. Implementación en Matlab.

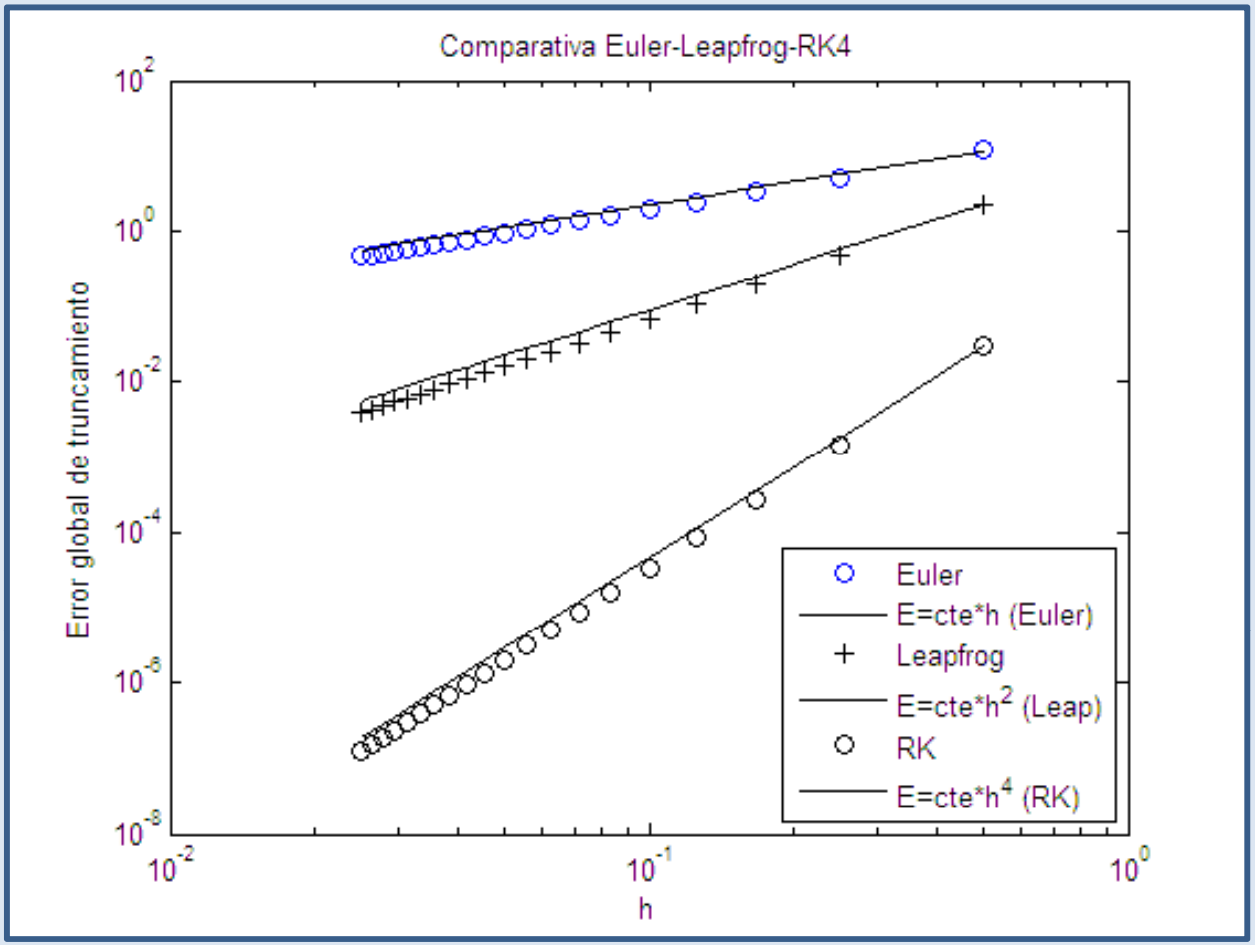
```
27      function [x,y]=fRK4(fun,x0,xf,y0,Nh)
28 -    h=(xf-x0)/Nh;
29 -    x=zeros(1,Nh+1);
30 -    y=zeros(1,Nh+1);
31 -    x(1)=x0;
32 -    y(1)=y0;
33 -    hm=h*0.5;    % La mitad del paso.
34 -    for j=2:1:Nh+1
35 -        xm=x(j-1)+h/2;
36          % Primera evaluación de f(x,y)
37 -        f1=fun(x(j-1),y(j-1));
38 -        ym1=y(j-1)+hm*f1;
39          % Segunda evaluación de f(x,y)
40 -        f2 = fun(xm,ym1);
41 -        ym2=y(j-1)+hm*f2;
42          % Tercera evaluación de f(x,y)
43 -        f3=fun(xm,ym2);
44 -        yj=y(j-1)+h*f3;
45          % Cuarta evaluación de f(x,y)
46 -        x(j)=x(j-1)+h;
47 -        f4=fun(x(j),yj);
48          % El valor final de y(j).
49 -        y(j)=y(j-1)+h*(f1/6+f2/3+f3/3+f4/6);
50 -    end
51 -    end
```

Comparativa Euler-Leapfrog-RK4



- Hemos conseguido mucha más precisión que con los dos métidos anteriores usando el mismo paso.
- El precio que hemos pagado es que hacemos cuatro evaluaciones de f(x,y) en cada paso.

Error del método RK4



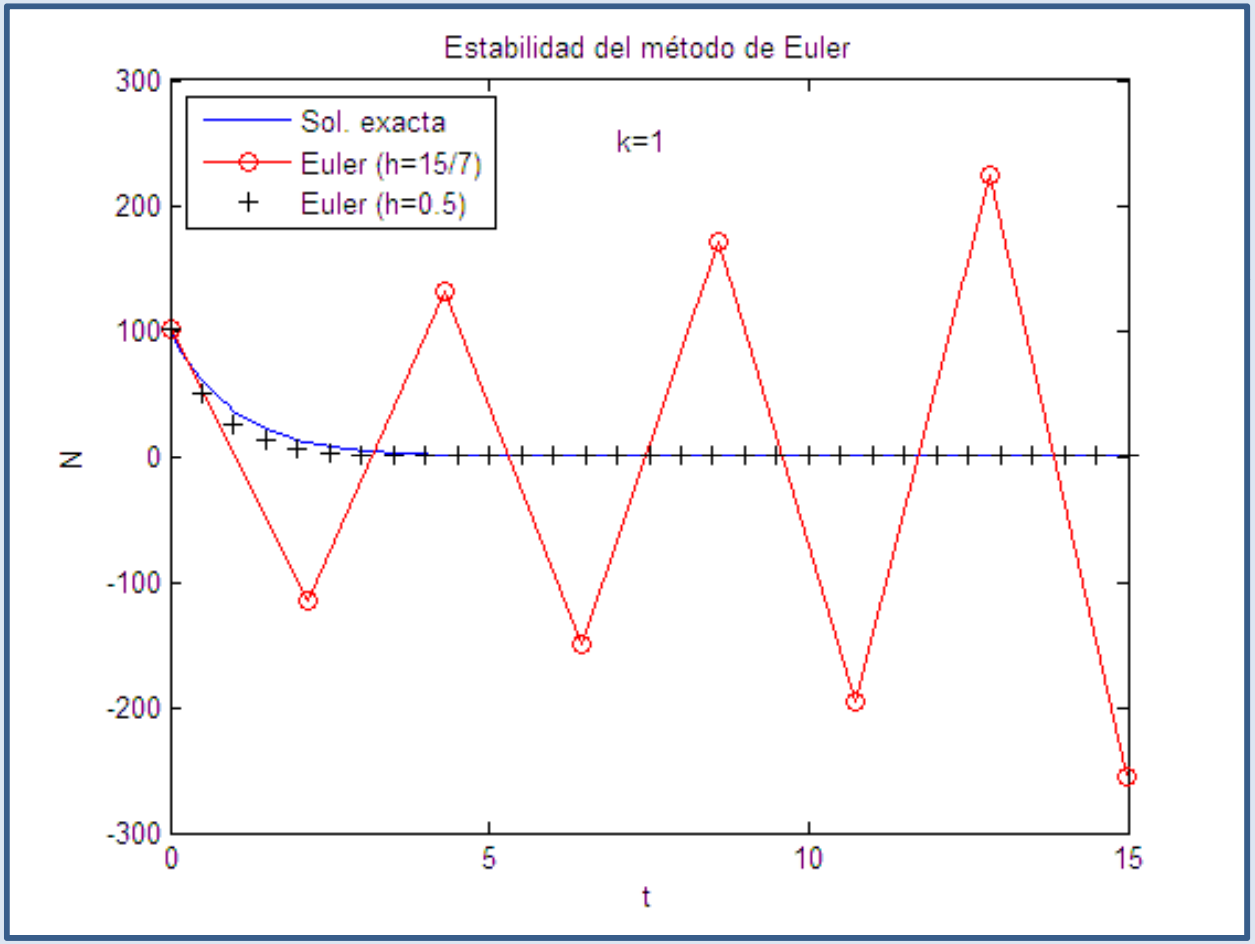
➢En el método RK de cuarto orden el error depende de h^4, como corresponde a un **método de cuarto orden**..

➢Si reducimos el paso a la mitad, el error se reduce a la dieciseisava parte.

Estabilidad

- Volvamos al método de Euler. Supongamos que quiero resolver el problema para un tiempo t más grande.
- Empiezo probando con el método de Euler y hago el paso h más grande para tener que hacer menos cálculos.

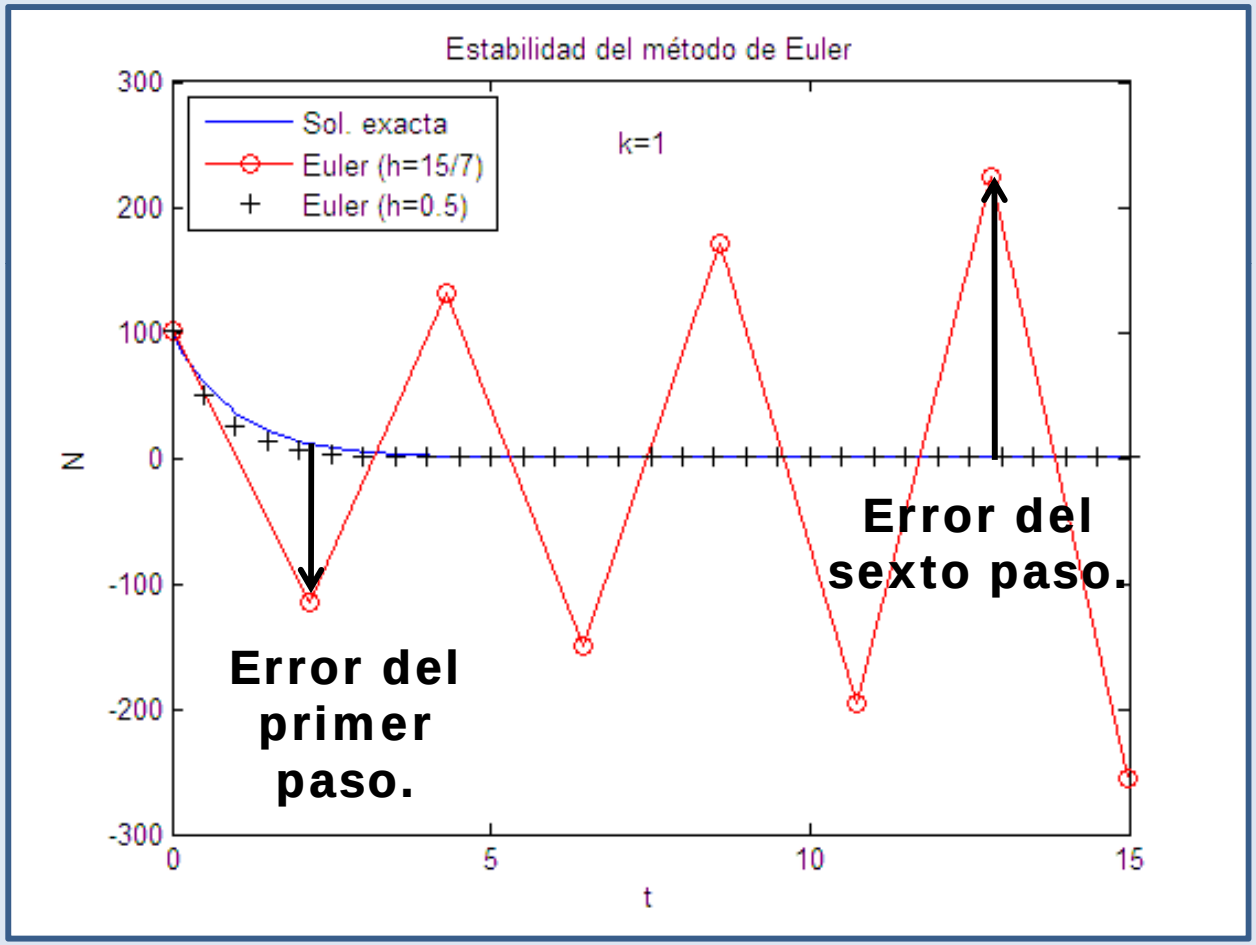
Si hago h>1 la solución numérica que obtengo es absurda.



Origen de la inestabilidad

En el ejemplo con h>1 el error que cometo en el primer paso crece en cada iteración del método.

Para que un método sea estable el error que cometo en cada paso debe atenuarse en las siguientes iteraciones.



Explicación para el método de Euler

- En el ejemplo que hemos puesto:

$$\frac{dy}{dx} = -ky \qquad k > 0$$

- De acuerdo con el método de Euler:

$$y_n = y_{n-1} + f\left(x_{n-1}, y_{n-1}\right)h = y_{n-1} - ky_{n-1}h = \left(1-kh\right)y_{n-1}$$

- Deshaciendo iteraciones hasta el punto inicial (x_0,y_0):

$$y_n = (1-kh)y_{n-1} = (1-kh)^2 y_{n-1} = \ldots = (1-kh)^n y_0$$

- Para que y_n tienda a cero, que es el comportamiento correcto:

$$|1-kh| < 1 \quad \Rightarrow \quad h < \frac{2}{k}$$

Generalización

- Si escribimos la ecuación diferencial general:

$$\frac{dy}{dx} = f(x, y; k)$$

- La condición de estabilidad del método de Euler puede escribirse como:

$$|1 - kh| < 1 \quad \Rightarrow \quad \left| 1 + \frac{f(x, y; k)}{y(x)} h \right| < 1$$

- Para cualesquiera valores de f(x,y;h) y la solución del problema y(x).

CONCLUSIÓN: Como en general, no conoceremos f(x,y;k)/y(x) de antemano porque no conocemos y(x), **no podemos decir cuál es el mayor paso h que es conveniente utilizar.**

Control del paso

- El mismo problema tienen los otros algoritmos que hemos utilizado.

- La solución es cambiar el paso h conforme resolvemos el problema para mantener el error bajo control

- A eso se le llama **control del paso**.

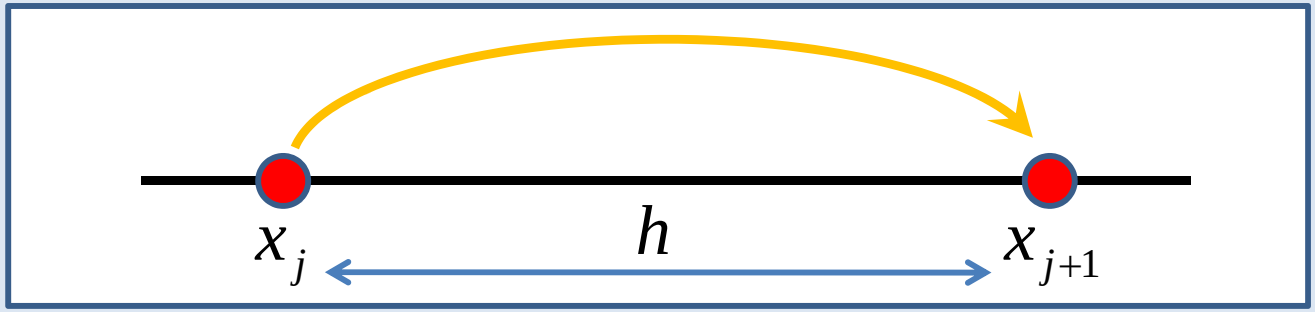
En cualquier cálculo serio hay que usar control del paso.

Las rutinas que hemos presentado como ejemplo no tienen control de paso. Por esa razón, en cualquier cálculo serio hay que usar las funciones de Matlab, que sí incorporan control de paso.

Control del paso por “step doubling” (I)

Suponemos que estamos en x_j y quiero avanzar la solución a $x_{j+1}=x_j+h$ usando el Método de Runge-Kutta de 4º orden. Además **quiero que el error de truncamiento sea menor que un valor Δ.**

- Primero avanzamos la solución en un solo paso.



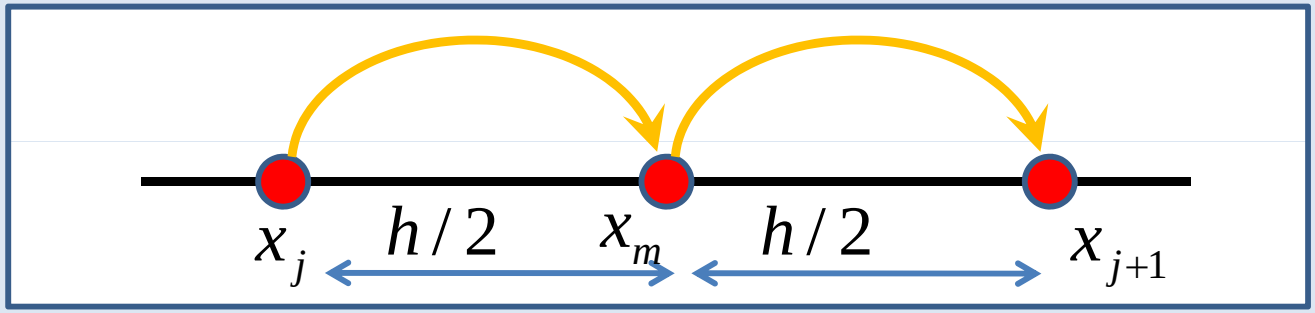
- Como el método que estoy usando es de cuarto orden, sé que:

$$y_{exact}(x_{j+1}) - y_{1p}(x_{j+1}) = Ch^5 + O(h^6)$$

Solución exacta	Solución numérica aproximada	Error de truncamiento de 1 paso	Términos de orden superior

Control del paso por “step doubling” (II)

- Ahora avanzo la solución en dos pasos de anchura h/2.



- Como el método que estoy usando es de cuarto orden, sé que:

$$y_{exact}(x_{j+1}) - y_{2p}(x_{j+1}) = 2C\left(\frac{h}{2}\right)^5 + O(h^6)$$

Solución exacta	Solución numérica aproximada	Error de truncamiento de 2 pasos	Términos de orden superior

Control del paso por “step doubling” (III)

Estimamos el error de truncamiento $e(x_{i+1})$ como:

$$e(x_{j+1}) \approx y_{1p}(x_{j+1}) - y_{2p}(x_{j+1})$$

- Error de truncamiento
- Solución numérica aproximada (1 paso)
- Solución numérica aproximada (2 pasos)

- Si $e(x_{j+1}) > \Delta$ tenemos que reducir el tamaño del paso h para mantener el error de truncamiento dentro del límite requerido.

  Podemos ayudarnos de que sabemos que si dividimos el paso por dos, el error de truncamiento se divide por 16.

- Si $e(x_{j+1}) < \Delta$ podemos aumentar el tamaño del paso h para avanzar más rápidamente y acabar el cálculo antes..

  Normalmente las rutinas son más conservativas a la hora de aumentar el paso que reducirlo.

Métodos explícitos en Matlab.

Matlab tiene dos funciones para el método de Runge-Kutta.

[x,y]=ode45(odefun,xspan,y0,options)

[x,y]=ode23(odefun,xspan,y0,options)

ode[orden del método][orden en el que se evalúa el error]

Parámetros de entrada:
odefun: la función f(x,y,z) en forma de **función anónima**.
xspan: un vector [x0 xf] con los valores inicial y final de x
y0: el valor inicial de la variable dependiente y.
options: estructura con parámetros adicionales

Parámetros de salida:
x: vector con los valores de la variable independiente x
y: vector con los valores de la variable dependiente y.

Estas funciones sólo resuelven ecuaciones del tipo: $\frac{dy}{dx} = f(x, y; k)$

Cómo modificar la estructura *options*

Para modificar la estructura *options* se usa el comando *odeset*:

options=odeset('Campo1',Valor1,'Campo2',Valor2,...)

Parametros de entrada:
Campo1,Campo2,...: Nombre de los campos de la estructura *options* que queremos modificar.
Valor1,Valor2,...: Valor que queremos del correspondiente campo.

	InitialStep	MaxStep
Significado	Valor de h para el primer paso.	Máximo valor del paso h que aceptamos.
Rango	Real positivo	Real positivo
Valor por defecto	Variable	$(x_f - x_0)/10$

Parámetros que controlan el error.

En Matlab, el máximo error de truncamiento en un paso Δ se especifica cómo:

$$\Delta = Max\left[AbsTol, RelTol \times \left| y_j \right|\right]$$

RelTol: Controla la cifra significativa de la solución que está afectada de error. Su valor por defecto es 10^{-3}, lo que indica que es la tercera cifra significativa de la solución la que está afectada de error.

AbsTol: Evita que la expresión anterior conduzca a error si |yj| se hace muy pequeño. Por ejemplo, si $|y_j|=0 \rightarrow$ *RelTol* x $|y_j|=0$, lo que indicaría que o queremos error de truncamiento (algo imposible) . Su valor por defecto es 10^{-6} pero siempre ha de ser bastante mayor que ***eps***.

RelTol* y *AbsTol* son campos de la estructura *options

Ecuaciones de orden 2 o superior.

Los métodos que hemos expuesto sólo resuelven ecuaciones del tipo:

$$\frac{dy}{dx} = f(x, y; k)$$

¿Que hacer entonces si tengo una ecuación de 2º orden o superior? Dijimos que resolveríamos:

$$B(x, y) = A_1(x, y)\frac{dy}{dx} + A_2(x, y)\frac{d^2y}{dx^2} + \ldots + A_n(x, y)\frac{d^ny}{dx^n}$$

Las ecuaciones diferenciales de orden 2º o superior las podemos resolver convirtiéndolas en sistemas de ecuaciones diferenciales de primer orden.

Ejemplo

Para una ecuación de segundo orden:

$$B(x,y) = A_1(x,y)\frac{dy}{dx} + A_2(x,y)\frac{d^2y}{dx^2}$$

Hacemos el cambio de variable:

$$z = \frac{dy}{dx}$$

Nos queda el sistema de dos ecuaciones de primer orden acopladas:

$$B(x,y) = A_1(x,y)z + A_2(x,y)\frac{dz}{dx} \Rightarrow \begin{cases} \frac{dz}{dx} = \frac{B(x,y) - A_1(x,y)z}{A_2(x,y)} \\ \frac{dy}{dx} = z \end{cases}$$

Bibliografía del Tema.

Numerical Recipes,
http://www.nr.com/oldverswitcher.html

A. Quarterioni, F. Salieri, Cálculo científico con Matlab y Octave (capítulo 7) , Springer-Verlag (2006), ISBN 10 88-470-0503-5

Cleve Moler. Numerical computing with Matlab.
http://www.mathworks.com/moler/chapters.html

Abramowitz and Stegun. Handbook of mathematical functions.
http://www.nrbook.com/abramowitz_and_stegun/