

# Mètodes Numèrics per Equacions Diferencials Ordinàries

Part I: Mètodes explícits d'un pas

Lluís Alsedà

Departament de Matemàtiques  
Universitat Autònoma de Barcelona

<http://www.mat.uab.cat/~alseda>

5 d'octubre de 2025 (versió 2.0)

**UAB** Universitat Autònoma  
de Barcelona

**Departament  
de Matemàtiques**



Subjecte a una llicència *Creative Commons de Reconeixement-NoComercial-CompartirIgual 4.0 Internacional* (<http://creativecommons.org/licenses/by-nc-sa/4.0/>)

Mètodes numèrics per problemes de valor inicial .....	▶ 1
Anàlisi dels mètodes explícits d'un pas .....	▶ 16
Error local de truncament .....	▶ 17
Error global de truncament, Ordre i Consistència .....	▶ 19
Convergència .....	▶ 23
Mètodes de Runge-Kutta .....	▶ 38
La definició .....	▶ 38
Exemples senzills .....	▶ 41
Mètodes RK4 .....	▶ 47
Sobre l'ordre dels mètodes de Runge-Kutta .....	▶ 58
Pas adaptatiu .....	▶ 59
Estratègies generals de pas adaptatiu .....	▶ 59
Mètodes de Runge-Kutta entrelaçats amb pas adaptatiu .....	▶ 70
El mètode Runge-Kutta-Fehlberg-7(8) en pseudocodi .....	▶ 78
Una implementació del mètode RKF7(8) en dimensió 1 .....	▶ 81
Un exemple complet i comentat d'integració d'un PVI en <b>C</b> .....	▶ 89

Recordem que

$$\begin{cases} \dot{x} = f(t, x), \\ x(t_0) = x^*, \end{cases}$$

s'anomena *problema de Cauchy* o *problema de valor inicial (PVI)*.

### El marc de treball: notació base

L'aproximació numèrica de la solució d'un PVI consisteix a fixar, en primer lloc, un  $T \in \mathbb{R}$  positiu que determina l'*interval d'integració* (o *d'aproximació*)  $I = [t_0, t_0 + T]$ . Per altra banda, cal fixar el *pas de discretització* o *pas d'integració*  $h > 0$ . Aquests dos *paràmetres del mètode* determinen els *nodes de discretització*  $t_k = t_0 + k \cdot h$  amb  $k = 1, 2, \dots, L_h$  i  $L_h = \lfloor \frac{T}{h} \rfloor$  (és a dir,  $L_h$  és l'enter més gran tal que  $hL_h \leq T$ ). Finalment, l'*aproximació numèrica de la solució d'un PVI* consisteix a calcular, a partir del punt  $(t_0, x_0 = x^*)$ , aproximacions  $x_k$  dels vertaders valors de la solució  $x(t_k)$  per  $k = 1, 2, \dots, L_h$ .

### Que és un mètode numèric per problemes de valor inicial?

És una *equació en diferències* que permet calcular la successió d'aproximacions  $\{x_k\}_{k \in \mathbb{N}}$  dels vertaders valors de la solució  $\{x(t_k)\}_{k \in \mathbb{N}}$  a partir d'un cert nombre d'aproximacions anteriors *consecutives*  $x_{k-1}, x_{k-2}, x_{k-3}, \dots, x_{k-s}$ .

El nombre  $s \geq 1$  s'anomena *nombre de passos del mètode*.

### Mètodes d'un pas i mètodes multi-pas

Un mètode numèric per a l'aproximació d'un problema de valor inicial s'anomena *mètode d'un pas* quan el nombre de passos  $s$  del mètode és 1.

Quan  $s > 1$  el mètode s'anomena *multi-pas*.

### Mètodes explícits i implícits

Es diu que un mètode és *implícit* quan cada aproximació  $x_k$  depèn implícitament d'ella mateixa.

El mètodes que no són implícits s'anomenen *explícits*.

### Observació

Si llegim atentament la definició del nombre de passos d'un mètode veurem que solament es tenen en compte les aproximacions *anteriors* a  $x_k$ . Per tant, quan les aproximacions  $x_k$  depenen implícitament d'elles mateixes, no s'incrementa el nombre de passos del mètode.

En particular, hi ha mètodes d'un pas explícits i implícits.

# Mètodes numèrics per problemes de valor inicial

Exemples de mètodes d'un pas

## Mètode d'Euler

$$x_{k+1} = x_k + hf(t_k, x_k).$$

## Mètode d'Euler endarrere (implícit)

$$x_{k+1} = x_k + hf(t_{k+1}, x_{k+1}).$$

## Mètode d'Euler modificat (Runge-Kutta-2)

$$x_{k+1} = x_k + hf\left(t_k + \frac{h}{2}, x_k + \frac{h}{2}f(t_k, x_k)\right).$$

## Mètode de Taylor d'ordre 2

$$x_{k+1} = x_k + hf(t_k, x_k) + \frac{h^2}{2}\left(f_t(t_k, x_k) + f_x(t_k, x_k)f(t_k, x_k)\right).$$

## Mètode Trapezoidal o de Crank-Nicolson (implícit)

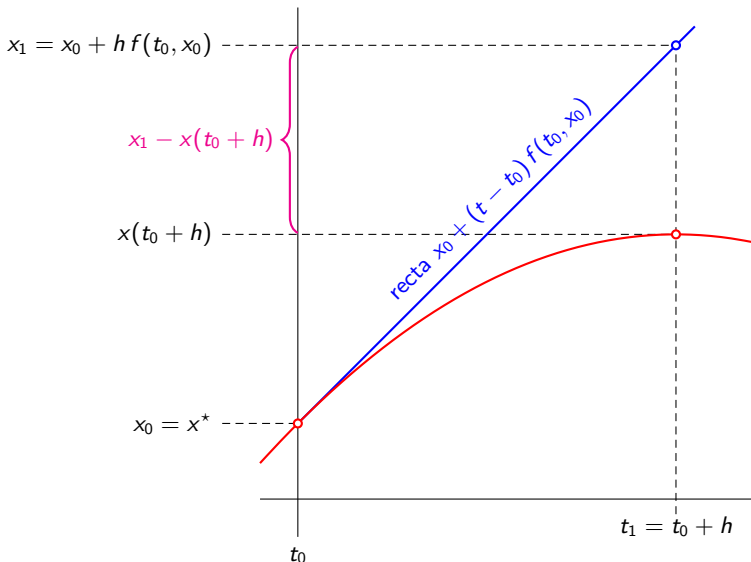
$$x_{k+1} = x_k + \frac{h}{2}(f(t_k, x_k) + f(t_{k+1}, x_{k+1})).$$

## Mètode de Heun (Runge-Kutta-2)

$$x_{k+1} = x_k + \frac{h}{2}\left(f(t_k, x_k) + f(t_k + h, x_k + hf(t_k, x_k))\right).$$

# Mètodes numèrics per problemes de valor inicial

## Interpretació gràfica del Mètode d'Euler



# Mètodes numèrics per problemes de valor inicial

L'exemple més senzill possible. Observem que l'error augmenta exponencialment

Considerem el problema de Cauchy:

$$\dot{x} = x \quad \text{amb} \quad x(0) = 1$$

que té  $x(t) = e^t$  com a solució.

El mètode d'Euler per a aproximar aquesta solució és

$$x_{k+1} = x_k + h f(t_k, x_k) = x_k + h x_k.$$

Si prenem  $h = 1$  com a pas, tenim  $x_{k+1} = x_k + h x_k = 2x_k$ .

La taula de la dreta compara els 10 primers iterats  $x_k$  del mètode d'Euler per aquest problema amb els corresponents valors exactes de la solució  $x(t_k) = x(k) = e^k$ .

$t_k$	$x_k$	$x(t_k) = e^{t_k}$
0	1	1
1	2	2.71828182845905
2	4	7.38905609893065
3	8	20.0855369231877
4	16	54.5981500331442
5	32	148.413159102577
6	64	403.428793492735
7	128	1096.63315842846
8	256	2980.95798704173
9	512	8103.08392757538
10	1024	22026.4657948067

# Mètodes numèrics per problemes de valor inicial

L'exemple més senzill possible però ens hi esforcem una mica més

Considerem el problema de Cauchy anterior:

$$\dot{x} = x \quad \text{amb} \quad x(0) = 1,$$

però ara, a més del mètode d'Euler (mètode de Taylor d'ordre 1), usarem el mètode de Taylor d'ordre 2:

$$\begin{aligned} x_{k+1} &= x_k + h f(t_k, x_k) + \frac{h^2}{2} \left( f_t(t_k, x_k) + f_x(t_k, x_k) f(t_k, x_k) \right) = \\ &= x_k + h x_k + \frac{h^2}{2} x_k = x_k \left( 1 + h + \frac{h^2}{2} \right). \end{aligned}$$

A més, en comptes de prendre  $h = 1$  ara prendrem  $h = \frac{1}{2}$ . Llavors,

$$\begin{aligned} x_{k+1} &= x_k + h x_k = \frac{3}{2} x_k && \text{(pel mètode d'Euler), i} \\ x_{k+1} &= x_k \left( 1 + h + \frac{h^2}{2} \right) = \frac{13}{8} x_k && \text{(pel mètode de Taylor d'ordre 2).} \end{aligned}$$

La següent diapositiva mostra una taula d'iterats d'ambdós mètodes. La columna etiquetada **Euler** mostra  $x(t_k) - x_k^{\text{Euler}}$ , mentre la columna etiquetada **Taylor** mostra  $x(t_k) - x_k^{\text{Taylor2}}$ .

# Mètodes numèrics per problemes de valor inicial

L'exemple més senzill possible però ens hi esforcem una mica més

$k$	$t_k$	$x(t_k) = e^{t_k}$	$x_k^{\text{Euler}}$	Euler	$x_k^{\text{Taylor}_2}$	Taylor
0	0	1	1	0	1	0
1	0.5	1.64872127070013	1.5	0.15	1.625	0.02
2	1.0	2.71828182845905	2.25	0.47	2.640625	0.08
3	1.5	4.48168907033806	3.375	1.11	4.291015625	0.19
4	2.0	7.38905609893065	5.0625	2.33	6.972900390625	0.42
5	2.5	12.1824939607035	7.59375	4.59	11.3309631347656	0.85
6	3.0	20.0855369231877	11.390625	8.69	18.4128150939941	1.67
7	3.5	33.1154519586923	17.0859375	16.03	29.9208245277405	3.19
8	4.0	54.5981500331442	25.62890625	28.97	48.6213398575783	5.98
9	4.5	90.0171313005218	38.443359375	51.57	79.0096772685647	11.01
10	5.0	148.413159102577	57.6650390625	90.75	128.390725561418	20.02
11	5.5	244.69193226422	86.49755859375	158.19	208.634929037304	36.06
12	6.0	403.428793492735	129.746337890625	273.68	339.031759685618	64.40
13	6.5	665.141633044362	194.619506835938	470.52	550.92660948913	114.22
14	7.0	1096.63315842846	291.929260253906	804.70	895.255740419836	201.38
15	7.5	1808.04241445606	437.893890380859	1370.15	1454.79057818223	353.25
16	8.0	2980.95798704173	656.840835571289	2324.12	2364.03468954613	616.92
17	8.5	4914.76884029913	985.261253356934	3929.51	3841.55637051246	1073.21
18	9.0	8103.08392757538	1477.8918800354	6625.19	6242.52910208275	1860.55
19	9.5	13359.7268296619	2216.8378200531	11142.89	10144.1097908845	3215.62
20	10.0	22026.4657948067	3325.25673007965	18701.21	16484.1784101873	5542.29

## Observacions

- *El mètode de Taylor d'ordre 2 té menys error que el mètode d'Euler* (com ha de ser ja que estem usant una millor aproximació a l'EDO).  
La cosa no és per “tirar coets” però és que estem aproximant l'exponencial!!
- Si comparem les aproximacions obtingudes amb el mètode d'Euler amb pas  $h = \frac{1}{2}$  als nodes  $t_k \in \{0, 1, 2, \dots, 10\}$ , amb les de la taula de la Pàgina 6 obtingudes amb pas  $h = 1$ ; veiem que les aproximacions obtingudes amb pas  $h = \frac{1}{2}$  són millors.  
Per tant, *sembla* que (si oblidem els errors d'arrodoniment) la precisió dels mètodes té una relació inversa amb la mida del pas.

Estem tractant amb tres objectes relacionats però totalment diferents:

- la vertadera solució  $x(t)$  d'un PVI,
- un mètode numèric *teòric* per a aproximar  $x(t)$  als nodes (temps) de discretització, i finalment,
- la *implementació* del mètode numèric que dona *una aproximació (amb errors d'arrodoniment) de l'aproximació* donada pel mètode numèric a la solució  $x(t)$  als nodes de discretització.

# Mètodes numèrics per problemes de valor inicial

Una reflexió/avís sobre els errors per iteració del mètode: a quin joc estem jugant

Més precisament:

- ① Tenim un PVI 
$$\begin{cases} \dot{x} = f(t, x(t)), \\ x(t_0) = x^*, \end{cases}$$

del qual suposem que té una solució única  $x(t)$  definida a tot l'interval d'integració  $I = [t_0, t_0 + T]$ .

Aquesta solució  $x(t)$ , que suposem incalculable, genera la successió

$$\{x(t_k)\}_{k=0,1,\dots,L_h}$$

de valors (discretitzats) de la solució als nodes d'integració.

- ② Tenim un mètode numèric 
$$\begin{cases} x_{k+1} = x_k + h\Phi(t_k, x_k, h), \\ x_0 = x^*, \end{cases}$$

que genera la successió  $\{x_k\}_{k=0,1,\dots,L_h}$  de valors aproximats de la successió  $\{x(t_k)\}_{k=0,1,\dots,L_h}$ .

# Mètodes numèrics per problemes de valor inicial

Una reflexió/avís sobre els errors per iteració del mètode: a quin joc estem jugant

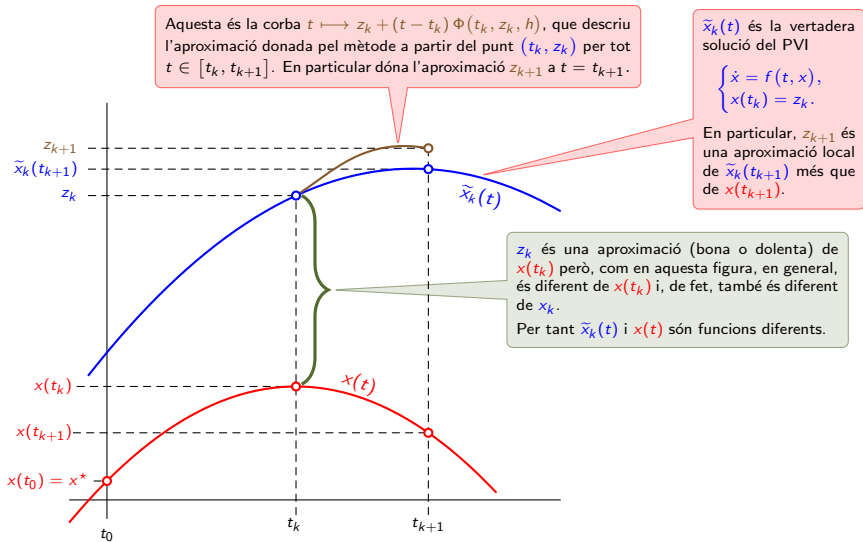
- 3 I, per acabar, tenim una implementació del mètode numèric anterior:

$$\begin{cases} z_{k+1} = z_k + h \Phi(t_k, z_k, h) + h \delta_{k+1}, \\ z_0 = x_0 + \delta_0 = x^* + \delta_0, \end{cases}$$

que genera la successió  $\{z_k\}_{k=0,1,\dots,L_h}$  de valors aproximats de la successió  $\{x_k\}_{k=0,1,\dots,L_h}$  amb errors d'arrodoniment.

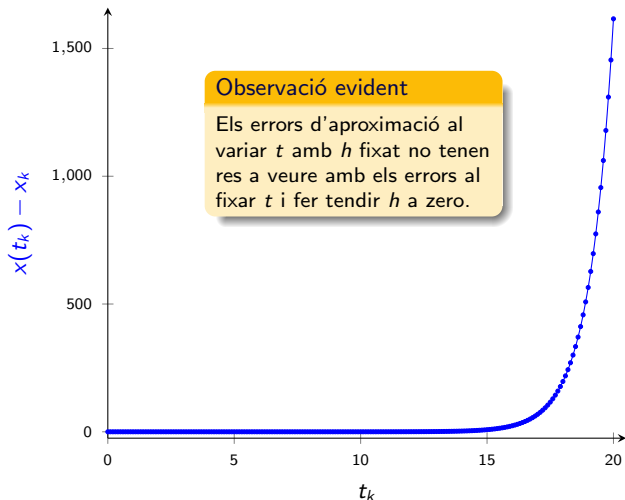
# Mètodes numèrics per problemes de valor inicial

Una reflexió/avís sobre els errors per iteració del mètode: a quin joc estem jugant



# Mètodes numèrics per problemes de valor inicial

Una reflexió/avís sobre els errors per iteració del mètode: a quin joc estem jugant; ho visualitzem a partir de l'exemple "més senzill impossible" de la Pàgina 7



La gràfica de l'esquerra mostra  $x(t_k) - x_k$  quan  $h = 10^{-3}$ . És a dir, mostra *l'error en l'aproximació d' $x(t_k)$  per  $x_k$  en funció de  $t_k$ , amb un pas fix.*

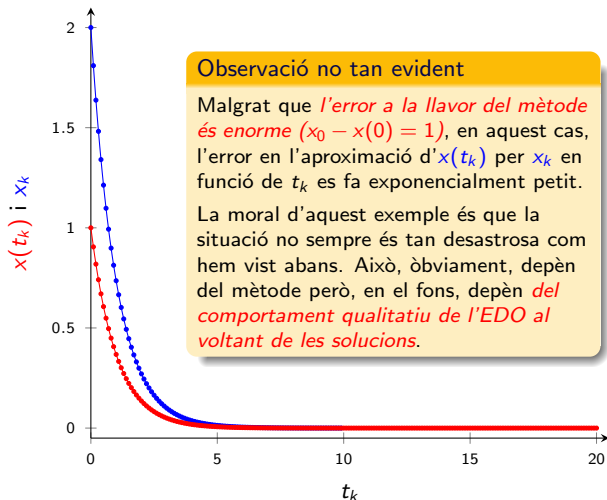
A la figura,  $x(t) = e^t$  és la solució del problema de Cauchy

$\dot{x} = x$  amb  $x(0) = 1$ ,  
i  $x_k$  és l'aproximació d' $x(t_k)$  donada pel mètode de Taylor d'ordre 2 amb pas  $h$ :

$$\begin{cases} x_{k+1} = x_k \left( 1 + h + \frac{h^2}{2} \right), \\ x_0 = x(0) = 1. \end{cases}$$

# Mètodes numèrics per problemes de valor inicial

Una reflexió/avís sobre els errors per iteració del mètode: l'altra cara de la moneda



La gràfica de l'esquerra mostra  $x(t_k)$  en vermell i  $x_k$  en blau, també calculats amb pas  $h = 10^{-3}$ .

$x(t) = e^{-t}$  és la solució del problema de Cauchy  $\dot{x} = -x$  amb  $x(0) = 1$ , i  $x_k$  és l'aproximació d' $x(t_k)$  donada pel mètode de Taylor d'ordre 2 amb pas  $h$ , que en aquest cas és:

$$\begin{cases} x_{k+1} = x_k \left( 1 - h + \frac{h^2}{2} \right), \\ x_0 = 2 \neq x(0) = 1. \end{cases}$$

L'objectiu de la segona part d'aquestes notes és el d'introduir i estudiar les nocions d'error local i global de truncament, ordre, consistència i convergència.

Tota aquesta "artilleria" apunta al control dels errors d'aproximació dels mètodes numèrics a la vertadera solució en instants de temps fixats *quan varia el pas  $h$* .

Més concretament, ens situem en el marc de l'exemple de les Pàgines 7 i 8, amb la intenció de donar suport teòric a la segona afirmació de la Pàgina 9. O més ben dit, volem determinar en quines condicions aquesta afirmació és certa.

Observem que aquest anàlisi és totalment transversal al de la Pàgina 13, i als exemples de les Pàgines 14 i 15.

# Anàlisi dels mètodes explícits d'un pas

## Error local de truncament

Els mètodes explícits d'un pas, usualment, es poden escriure com

$$x_{k+1} = x_k + h \Phi(t_k, x_k, h),$$

on la funció  $\Phi$  s'anomena *funció d'increment del mètode*. Òbviament, és la funció d'increment la que determina el mètode.

Usant el Teorema de Taylor, el vertader valor de la solució del PVI al punt  $t_k + h$  es pot escriure en termes de la funció d'increment del mètode:

$$\begin{aligned}x(t_k + h) &= x(t_k) + h \Delta(t_k, x(t_k), h) \\ &= x(t_k) + h \Phi(t_k, x(t_k), h) + h \tau(t_k, h, \Phi)\end{aligned}$$

amb

$$\begin{aligned}\tau(t_k, h, \Phi) &= \Delta(t_k, x(t_k), h) - \Phi(t_k, x(t_k), h) \\ &= \frac{x(t_{k+1}) - (x(t_k) + h \Phi(t_k, x(t_k), h))}{h}.\end{aligned}$$

### Definició (Error local de truncament)

La quantitat  $\tau(t_k, h, \Phi)$  s'anomena *error local de truncament (o de discretització) del mètode  $\Phi$  al node  $t_k$* .

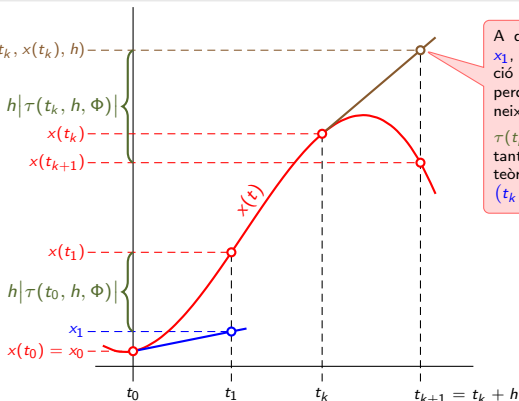
# Anàlisi dels mètodes explícits d'un pas

## Entenent l'error local de truncament

### És a dir

l'error local de truncament és una *mesura teòrica de l'error relatiu a  $h$*  en aproximar  $x(t_{k+1}) = x(t_k + h)$  pel mètode

$x(t_k) + h\Phi(t_k, x(t_k), h)$ , prenent  $(t_k, x(t_k))$  com a punt (certament desconegut) de partida.



A diferència del punt  $x_1$ , aquesta aproximació no es calcula perquè, de fet, no coneixem el punt  $x(t_k)$ .

$\tau(t_k, h, \Phi)$  és, per tant, una mesura teòrica de l'error a  $(t_k, x(t_k))$ .

$$\tau(t_0, h, \Phi) = \frac{x(t_0 + h) - (x(t_0) + h\Phi(t_0, x(t_0), h))}{h} = \frac{x(t_1) - (x_0 + h\Phi(t_0, x_0, h))}{h} = \frac{x(t_1) - x_1}{h}.$$

$$\tau(t_k, h, \Phi) = \frac{x(t_{k+1}) - (x(t_k) + h\Phi(t_k, x(t_k), h))}{h}.$$

### Definició (Error global de truncament)

La quantitat

$$\tau(h, \Phi) := \max_{k=0,1,\dots,L_h-1} |\tau(t_k, h, \Phi)|$$

s'anomena *error global de truncament (o de discretització) del mètode  $\Phi$* .

### Definició (Consistència)

Un mètode explícit d'un pas  $\Phi$  s'anomena *consistent* quan l'error global de truncament tendeix a zero amb  $h$ :

$$\lim_{h \rightarrow 0} \tau(h, \Phi) = 0.$$

### Observació: No volem mètodes que no siguin consistents

En un mètode *no* consistent, encara que coneguem  $x(t_k)$ , els errors  $|\tau(t_k, h, \Phi)|$  (d'aproximació de  $x(t_k + h)$  per  $x(t_k) + h\Phi(t_k, x(t_k), h)$ ) es mantenen *desastrosament* afitats quan  $h \rightarrow 0$ .

La noció d'*ordre d'un mètode* parametriza el comportament infinitesimal (és a dir la velocitat de convergència a zero) de l'error local de discretització respecte d' $h$ , quan el mètode és consistent.

### Ordre

Un mètode explícit d'un pas  $\Phi$  té *ordre*  $p$  si

$$\tau(h, \Phi) = \mathcal{O}(h^p).$$

# Anàlisi dels mètodes explícits d'un pas

## Exemples de càlcul de l'orde

Començarem usant el Teorema de Taylor per a obtenir una aproximació de la solució d'una EDO<sup>1</sup>. En particular això ens donarà una expressió convenient de  $\Delta(t, x(t), h)$ .

$$\begin{aligned}x(t+h) &= x(t) + hx'(t) + \frac{h^2}{2}x''(t) + \mathcal{O}(h^3) \\ &= x(t) + hf(t, x(t)) + \frac{h^2}{2} \frac{d}{dt}f(t, x(t)) + \mathcal{O}(h^3) \\ &= x(t) + hf(t, x(t)) + \frac{h^2}{2} \left( f_t(t, x(t)) + f_x(t, x(t))x'(t) \right) + \mathcal{O}(h^3) \\ &= x(t) + hf(t, x(t)) + \frac{h^2}{2} \left( f_t(t, x(t)) + f_x(t, x(t))f(t, x(t)) \right) + \mathcal{O}(h^3)\end{aligned}$$

Per tant,

$$\Delta(t, x(t), h) = f(t, x(t)) + \frac{h}{2} \left( f_t(t, x(t)) + f_x(t, x(t))f(t, x(t)) \right) + \mathcal{O}(h^2).$$

---

<sup>1</sup>Aquest és un càlcul típic al món dels mètodes numèrics d'EDOs.

Ara, l'error local de discretització del mètode d'Euler és

$$\begin{aligned}\tau(t, h, \Phi) &= \Delta(t, x(t), h) - \Phi(t, x(t), h) \\ &= \Delta(t, x(t), h) - f(t, x(t)) \\ &= \frac{h}{2} \left( f_t(t, x(t)) + f_x(t, x(t)) f(t, x(t)) \right) + \mathcal{O}(h^2) \\ &= \mathcal{O}(h).\end{aligned}$$

Pel mètode de Taylor d'ordre 2 tenim

$$\Phi(t, x(t), h) = f(t, x(t)) + \frac{h}{2} \left( f_t(t, x(t)) + f_x(t, x(t)) f(t, x(t)) \right)$$

i, tal com era d'esperar,

$$\tau(t, h, \Phi) = \Delta(t, x(t), h) - \Phi(t, x(t), h) = \mathcal{O}(h^2).$$

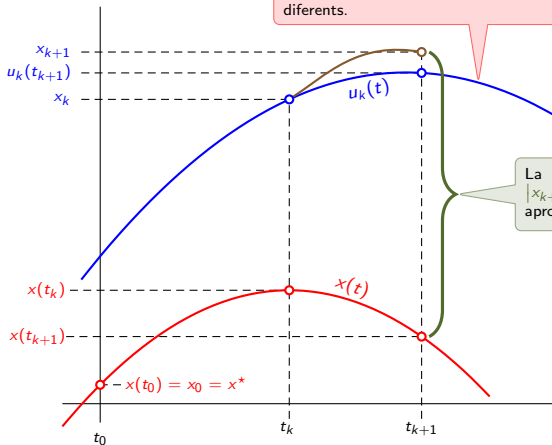
# Mètodes numèrics per problemes de valor inicial

## Introducció al concepte de Convergència

Aquí  $u_k(t)$  denota la verdadera solució del PVI

$$\dot{x} = f(t, x) \quad \text{amb} \quad x(t_k) = x_k.$$

Com s'ha dit abans, en general,  $u_k(t)$  i  $x(t)$  són funcions diferents.



La noció de **convergència** pretén controlar  $|x_{k+1} - x(t_{k+1})|$ , l'error absolut comès en aproximar  $x(t_{k+1})$  per  $x_{k+1}$ .

# Anàlisi dels mètodes explícits d'un pas

Sobre la mesura d'errors a punts arbitraris de l'interval d'integració

Necessitem poder mesurar errors a punts arbitraris  $t$  de l'interval d'integració  $[t_0, t_0 + T]$ .

El problema és que els algorismes de que disposem solament ens permeten calcular aproximacions a la solució del PVI en els nodes de discretització  $t_k$  i, en general,  $t$  no ho serà. Simultàniament volem poder jugar amb el pas  $h$ , fent-lo més petit si convé.

La solució d'aquest problema és òbvia: *si Mahoma no va a la muntanya, la muntanya ha d'anar a Mahoma.*

Això es fa prenent, per a cada  $m \in \mathbb{N}$ , un pas

$$h_m = \frac{t - t_0}{m},$$

i definint els nodes de discretització  $t_k := t_0 + kh_m$  per  $k = 0, 1, 2, \dots, m$ .

*Així,  $t = t_m$  si que és un node de discretització.*

# Anàlisi dels mètodes explícits d'un pas

## Definició de Convergència

### Notació

Per a cada  $t \in [t_0, t_0 + T]$  i per a cada  $m \in \mathbb{N}$  denotem

$$h_m := \frac{t - t_0}{m},$$

i definim els nodes de discretització

$$t_k := t_0 + kh_m \quad \text{per a tot} \quad k \in \{0, 1, 2, \dots, L_{h_m}\}.$$

En particular,  $t = t_m$ .

En aquest context, per a tot  $k \in \{0, 1, 2, \dots, L_{h_m}\}$ , denotarem per  $u(t_k, h_m)$  l'aproximació  $x_k$  d' $x(t_k)$  obtinguda amb pas  $h = h_m$ .

Més concretament:

$$\begin{aligned} u(t_0, h_m) &:= x_0 = x^*, \\ u(t_{k+1}, h_m) &:= u(t_k, h_m) + h_m \Phi(t_k, u(t_k, h_m), h_m), \quad \text{i} \\ u(t, h_m) &= u(t_m, h_m) = x_m. \end{aligned}$$

### Definició (Convergència)

Un mètode explícit d'un pas  $\Phi$  s'anomena *convergent* si

$$\lim_{m \rightarrow \infty} u(t, h_m) - x(t) = 0$$

per tot  $t \in [t_0, t_0 + T]$ , i *convergent d'ordre  $p$*  si

$$u(t, h_m) - x(t) = \mathcal{O}(h_m^p).$$

Un mètode és *convergent* si

per tot  $t \in [t_0, t_0 + T]$ , l'error absolut en aproximar  $x(t)$  (el vertader valor de la solució al punt  $t$ ) per  $u(t, h_m) = u(t_m, h_m)$  és infinitesimal amb el pas  $h_m$ .

És a dir, *disminuint el pas tant com calgui puc obtenir aproximacions a la solució tan bones com vulgui.*

**En conclusió:** *un mètode que no sigui convergent (amb errors absoluts sistemàtics independentment del pas  $h$ ) no ens serveix de res.*

La condició que un mètode sigui convergent és difícil de comprovar. Afortunadament la condició de consistència és molt més senzilla de comprovar (encara que és més feble), i tenim el següent teorema que mostra que *consistència amb una condició addicional d'estabilitat del mètode implica convergència.*

# Anàlisi dels mètodes explícits d'un pas

Estabilitat + Consistència  $\implies$  Convergència

## Teorema (Estabilitat (Lipschitz) + Consistència $\implies$ Convergència)

*Sigui  $\dot{x} = f(t, x(t))$  amb  $x(t_0) = x^*$  un PVI. Considerem un mètode explícit d'un pas tal que la funció,  $\Phi$ , d'increment del mètode és Lipschitz contínua respecte de la segona variable, amb constant  $K$  independent de  $h$  i de  $t \in (t_0, t_0 + T]$ . És a dir, existeixen  $h^* > 0$  i  $K > 0$  tals que, per a cada  $h \in (0, h^*]$  i  $t \in (t_0, t_0 + T]$ ,*

$$|\Phi(t, x, h) - \Phi(t, y, h)| \leq K |x - y|.$$

*Llavors,*

$$|x(t) - u(t, h_m)| \leq \tau(h_m, \Phi) \frac{e^{K(t-t_0)} - 1}{K},$$

*per a tot  $t \in (t_0, t_0 + T]$ .*

*Per tant, si el mètode és consistent aleshores és convergent.*

*Encara més, si el mètode té ordre  $p$  llavors és convergent d'ordre  $p$ .*

## Observació

Una noció relacionada amb la de convergència és la d'*estabilitat zero*, que controla els errors d'arrodoniment en l'aplicació d'un mètode afitant la successió

$$\{|x_k - z_k|\}_{k=0,1,\dots,L_h},$$

on  $\{x_k\}_{k=0,1,\dots,L_h}$  és la successió d'iterats generats pel mètode (teòric):

$$\begin{cases} x_{k+1} = x_k + h\Phi(t_k, x_k, h), \\ x_0 = x^*, \end{cases}$$

i  $\{z_k\}_{k=0,1,\dots,L_h}$  és la successió d'iterats (amb errors d'arrodoniment) generats per la implementació del mètode:

$$\begin{cases} z_{k+1} = z_k + h\Phi(t_k, z_k, h) + h\delta_{k+1}, \\ z_0 = x_0 + \delta_0 = x^* + \delta_0. \end{cases}$$

En general, hi ha dues maneres de millorar la precisió:

- 1 Reduir el pas de temps  $h$  augmentant la quantitat de passos necessaris per a arribar a un temps  $t$ .
- 2 Augmentar l'ordre de convergència (tot i que no és raonable augmentar-lo indiscriminadament).

La segona via l'explorarem més endavant deduint mètodes (de Runge-Kutta i de Taylor) d'ordre elevat.

Els avantatges i els contres de la primera alternativa es discuteixen a les tres diapositives següents.

# Anàlisi dels mètodes explícits d'un pas

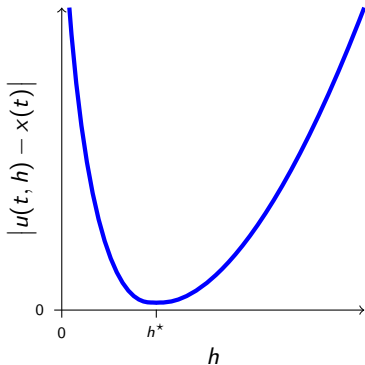
## Sobre la precisió dels mètodes

La gràfica (qualitativa) de la figura de la dreta, mostra el comportament típic de l'error absolut  $|u(t, h) - x(t)|$  per un  $t$  fixat, en funció del pas  $h$ .

Per passos grans, la velocitat amb que es calcula  $u(t, h)$  és gran (ja que calen poques iteracions) però l'error de truncament també és gran.

Per passos petits, calen moltes iteracions del mètode per arribar a obtenir  $u(t, h)$ . Per tant, la computació d' $u(t, h)$  és lenta i, a més, l'error és gran degut a l'acumulació dels errors d'arrodoniment.

Usualment hi ha una zona "privilegiada" on es troba el pas òptim  $h^*$ , que permet velocitats acceptables en relació als errors absoluts d'aproximació. En particular, el pas òptim  $h^*$ , determina l'error mínim.



# Anàlisi dels mètodes explícits d'un pas

Sobre la precisió dels mètodes: La disquisició de la pàgina anterior a la llum d'un exemple

Considerem el PVI

$$\begin{cases} \dot{x} = -200tx^2, \\ x(-1) = \frac{1}{101}, \end{cases}$$

que té  $x(t) = \frac{1}{1+100t^2}$  per solució.

Volem calcular  $|u(0, h_m) - x(0)| = |u(0, h_m) - 1|$  a partir de  $t_0 = -1$  amb el mètode de Heun (aplicat reiteradament):

$$\begin{aligned} x_{k+1} &= x_k + \frac{h_m}{2} \left( f(t_k, x_k) + f(t_k + h_m, x_k + h_m f(t_k, x_k)) \right) \\ &= x_k - 100h_m \left( t_k x_k^2 + (t_k + h_m) \left( x_k - 200h_m t_k x_k^2 \right)^2 \right), \text{ amb} \end{aligned}$$

$$x_0 = x(t_0) = x(-1) = \frac{1}{101},$$

amb aritmètica de 7 dígits (que és la que té el **C** per variables float) i diferents passos  $h_m$ .

# Anàlisi dels mètodes explícits d'un pas

Sobre la precisió dels mètodes: La disquisició de la pàgina anterior a la llum d'un exemple

Com a resultat d'aquest exercici obtenim la taula següent:

$h_m$	$10^{-2}$	$5 \cdot 10^{-3}$	$10^{-3}$	$5 \cdot 10^{-4}$	$10^{-4}$	$5 \cdot 10^{-5}$	$10^{-5}$
$ u(0, h_m) - x(0) $	$6.3 \cdot 10^{-2}$	$1.7 \cdot 10^{-2}$	$6.5 \cdot 10^{-4}$	$2 \cdot 10^{-5}$	$8.5 \cdot 10^{-4}$	$1.9 \cdot 10^{-3}$	$8 \cdot 10^{-3}$

## Comentaris il·lustratius a la disquisició de la Pàgina 31

- Per passos grans ( $10^{-2}$  i  $5 \cdot 10^{-3}$ ) el mètode fa poques iteracions i, per tant, no hi pot haver molt error d'arrodoniment. És a dir, l'error es degut bàsicament als errors de truncament del mètode.
- Ben al contrari, per passos petits (per exemple  $h_m = 10^{-5}$ ), el mètode fa moltes iteracions (per  $h_m = 10^{-5}$ ,  $m = 10^5$  i el mètode s'itera  $10^5$  vegades) i l'error s'incrementa enormement degut als errors d'arrodoniment. Els errors de truncament del mètode quan es fan moltes iteracions òbviament poden ser grans però també poden ser menyspreables com mostra l'exemple de la Pagina 15.

# Anàlisi dels mètodes explícits d'un pas

Sobre la precisió dels mètodes: La disquisició de la pàgina anterior a la llum d'un exemple

## Més comentaris il·lustratius a la disquisició de la Pàgina 31

- Observem que, efectivament, hi ha un pas òptim. En aquest cas és  $5 \cdot 10^{-4}$ .
- Desafortunadament, en general no és possible fer l'anàlisi de la taula ja que no coneixem la solució (recordem que tot aquest esforç és precisament per a aproximar-la numèricament degut a que no la coneixem).

## Un exercici d'entrenament (amb esperit purament esportiu)

Recalculer la taula de la diapositiva anterior.

**Nota I:** És un bon exercici de seguiment no local d'una solució.

**Nota II:** Es gairebé impossible que la taula surti quantitativament com la de la diapositiva anterior però, certament, hauria de sortir similar qualitativament.

## Demostració del Teorema de Convergència

Sigui  $M^* \in \mathbb{N}$  tal que  $t - t_0 \leq M^* h^*$ , i prenem  $m \geq M^*$ . Llavors tenim  $h_m \in (0, h^*]$ , i podem usar la notació de la Pàgina 25. En particular,  $t = t_m$ .

Com a la definició d'error local de truncament, escrivim:

$$\begin{cases} x(t_{k+1}) = x(t_k + h_m) = x(t_k) + h_m \Phi(t_k, x(t_k), h_m) + h_m \tau(t_k, h_m, \Phi) \\ x(t_0) = x_0 = u(t_0, h_m), \end{cases}$$

Llavors, per a cada  $k \in \{0, 1, \dots, m-1\}$  tenim

$$\begin{aligned} & |x(t_{k+1}) - u(t_{k+1}, h_m)| \\ & \leq |x(t_k) - u(t_k, h_m)| + h_m |\tau(t_k, h_m, \Phi)| + \\ & \quad h_m |\Phi(t_k, x(t_k), h_m) - \Phi(t_k, u(t_k, h_m), h_m)| \\ & \leq (1 + h_m K) |x(t_k) - u(t_k, h_m)| + h_m \tau(h_m, \Phi). \end{aligned}$$

### Demostració del Teorema de Convergència

Per tant,

$$\begin{aligned} |x(t_1) - u(t_1, h_m)| &\leq (1 + h_m K) |x(t_0) - u(t_0, h_m)| + h_m \tau(h_m, \Phi) \\ &= h_m \tau(h_m, \Phi), \end{aligned}$$

$$\begin{aligned} |x(t_2) - u(t_2, h_m)| &\leq (1 + h_m K) |x(t_1) - u(t_1, h_m)| + h_m \tau(h_m, \Phi) \\ &\leq h_m \tau(h_m, \Phi) (1 + (1 + h_m K)), \end{aligned}$$

$$\begin{aligned} |x(t_3) - u(t_3, h_m)| &\leq (1 + h_m K) |x(t_2) - u(t_2, h_m)| + h_m \tau(h_m, \Phi) \\ &\leq h_m \tau(h_m, \Phi) (1 + (1 + h_m K) + (1 + h_m K)^2), \end{aligned}$$

...

$$\begin{aligned} |x(t_m) - u(t_m, h_m)| &\leq (1 + h_m K) |x(t_{m-1}) - u(t_{m-1}, h_m)| + h_m \tau(h_m, \Phi) \\ &\leq h_m \tau(h_m, \Phi) \sum_{k=0}^{m-1} (1 + h_m K)^k \\ &= h_m \tau(h_m, \Phi) \frac{(1 + h_m K)^m - 1}{(1 + h_m K) - 1} = \tau(h_m, \Phi) \frac{(1 + h_m K)^m - 1}{K}. \end{aligned}$$

## Demostració del Teorema de Convergència

Ara cal afitar  $(1 + h_m K)^m$ . Tenim,

$$(1 + h_m K)^m = (1 + h_m K)^{\frac{t-t_0}{h_m}} = \left( (1 + h_m K)^{\frac{1}{h_m}} \right)^{t-t_0} \leq e^{K(t-t_0)}$$

ja que  $\lim_{h \searrow 0} (1 + hK)^{\frac{1}{h}} = e^K = \sup_{h \in (0, h^*]} (1 + hK)^{\frac{1}{h}}$ .

Resumint,

$$|x(t_m) - u(t_m, h_m)| \leq \tau(h_m, \Phi) \frac{(1 + h_m K)^m - 1}{K} \leq \tau(h_m, \Phi) \frac{e^{K(t-t_0)} - 1}{K}.$$



Aquesta part del curs es dedicarà a l'estudi sistemàtic dels mètodes de Runge-Kutta (RK) amb l'objectiu de poder construir mètodes amb ordre de convergència (arbitràriament) elevat<sup>2</sup>.

Els mètodes de Runge-Kutta són mètodes *interpolatoris* que, per a obtenir l'estimació  $x_{k+1}$  d' $x(t_{k+1})$ , usen avaluacions de la funció  $f$  a diversos nodes d'interpolació a l'interval  $[t_k, t_{k+1} = t_k + h]$ .

Formalment són mètodes *d'un pas* i poden ser explícits o implícits.

En el que segueix, ens concentrarem principalment en els mètodes RK explícits.

---

<sup>2</sup>Recordem que aquesta línia de treball no és la única manera de aconseguir mètodes amb ordre arbitràriament elevat. També hi ha la via dels mètodes de Taylor, que tractarem més endavant.

# Mètodes de Runge-Kutta

## Definició

Un *mètode de Runge-Kutta d' $m$  etapes* es defineix com

$$x_{k+1} := x_k + h \sum_{i=1}^m c_i \kappa_i(t_k, x_k),$$

on

$$\kappa_1(t_k, x_k) = f(t_k, x_k),$$

$$\kappa_2(t_k, x_k) = f(t_k + \alpha_2 h, x_k + h \beta_{2,1} \kappa_1(t_k, x_k)),$$

$$\kappa_3(t_k, x_k) = f(t_k + \alpha_3 h, x_k + h \beta_{3,1} \kappa_1(t_k, x_k) + h \beta_{3,2} \kappa_2(t_k, x_k)),$$

$$\vdots \qquad \qquad \qquad \vdots$$

$$\kappa_m(t_k, x_k) = f\left(t_k + \alpha_m h, x_k + h \sum_{j=1}^{m-1} \beta_{m,j} \kappa_j(t_k, x_k)\right).$$

Els paràmetres del mètode són: el nombre d'etapes  $m$  els coeficients  $c_1, c_2, \dots, c_m$ , els coeficients  $\alpha_2, \alpha_3, \dots, \alpha_m$  i els coeficients  $\beta_{i,j}$  per  $i \in \{2, 3, \dots, m\}$  i  $j \in \{1, 2, \dots, i-1\}$ .

# Mètodes Runge-Kutta

Definició: Taula de Butcher

Aquests coeficients, usualment, es presenten en forma compacta en una *taula de Butcher*:

0					
$\alpha_2$	$\beta_{2,1}$				
$\alpha_3$	$\beta_{3,1}$	$\beta_{3,2}$			
$\vdots$	$\vdots$	$\vdots$	$\ddots$		
$\alpha_m$	$\beta_{m,1}$	$\beta_{m,2}$	$\cdots$	$\beta_{m,m-1}$	
<hr/>					
	$c_1$	$c_2$	$\cdots$	$c_{m-1}$	$c_m$

## Observació – Recordem

Els mètodes de Runge-Kutta són mètodes *interpolatoris* que usen avaluacions de la funció  $f$  a  $m$  nodes d'interpolació a l'interval  $[t_k, t_{k+1} = t_k + h]$ . Formalment són mètodes *d'un pas*.

# Mètodes Runge-Kutta

Exemples: una etapa ( $m = 1$ )

El mètode de Runge-Kutta d'una etapa és

$$x_{k+1} = x_k + hc_1 \kappa_1(t_k, x_k) = x_k + hc_1 f(t_k, x_k),$$

on cal determinar la constant  $c_1$ . Òbviament determinarem  $c_1$  amb l'objectiu de maximitzar l'ordre del mètode.

De la Pàgina 21 (Taylor) tenim

$$\Delta(t, x(t), h) = f(t, x(t)) + \mathcal{O}(h).$$

Llavors, repetint el càlcul de la Pàgina 22,

$$\tau(t, h, \Phi) = \Delta(t, x(t), h) - \Phi(t, x(t), h) = (1 - c_1)f(t, x(t)) + \mathcal{O}(h).$$

Per tant,  $c_1 = 1$  i el mètode de Runge-Kutta d'una etapa acaba essent el mètode d'Euler i tenint ordre 1.

# Mètodes Runge-Kutta

Exemples: dues etapes ( $m = 2$ )

En aquest cas el mètode és

$$\begin{aligned}x_{k+1} &= x_k + h(c_1 \kappa_1(t_k, x_k) + c_2 \kappa_2(t_k, x_k)) \\ &= x_k + h\left(c_1 f(t_k, x_k) + c_2 f(t_k + \alpha_2 h, x_k + h \beta_{2,1} f(t_k, x_k))\right).\end{aligned}$$

Una altra vegada de la Pàgina 21 tenim,

$$\Delta(t, x(t), h) = f(t, x(t)) + \frac{h}{2} \left( f_t(t, x(t)) + f_x(t, x(t)) f(t, x(t)) \right) + \mathcal{O}(h^2).$$

D'altra banda,

$$\begin{aligned}\kappa_2(t, x(t)) &= f(t + \alpha_2 h, x(t) + h \beta_{2,1} f(t, x(t))) = f(t, x(t)) + \\ &h \alpha_2 f_t(t, x(t)) + h \beta_{2,1} f(t, x(t)) f_x(t, x(t)) + \mathcal{O}(h^2).\end{aligned}$$

# Mètodes Runge-Kutta

Exemples: dues etapes ( $m = 2$ )

Per tant,

$$\begin{aligned}\Phi(t, x(t), h) &= c_1 f(t, x(t)) + \\ &\quad c_2 f(t + \alpha_2 h, x(t) + h \beta_{2,1} f(t, x(t))) \\ &= (c_1 + c_2) f(t, x(t)) + \\ &\quad h \left( c_2 \alpha_2 f_t(t, x(t)) + c_2 \beta_{2,1} f(t, x(t)) f_x(t, x(t)) \right) + \\ &\quad \mathcal{O}(h^2),\end{aligned}$$

i

$$\begin{aligned}\tau(t, h, \Phi) &= \Phi(t, x(t), h) - \Delta(t, x(t), h) \\ &= (c_1 + c_2 - 1) f(t, x(t)) + h \left( c_2 \alpha_2 - \frac{1}{2} \right) f_t(t, x(t)) + \\ &\quad h \left( c_2 \beta_{2,1} - \frac{1}{2} \right) f(t, x(t)) f_x(t, x(t)) + \\ &\quad \mathcal{O}(h^2).\end{aligned}$$

# Mètodes Runge-Kutta

Exemples: dues etapes ( $m = 2$ )

El mètode de Runge-Kutta de dues etapes té ordre 2 si i només si

$$c_1 + c_2 = 1, \quad c_2 \alpha_2 = \frac{1}{2} \quad \text{i} \quad c_2 \beta_{2,1} = \frac{1}{2}.$$

El sistema té quatre incògnites i tres equacions. Per a resoldre'l cal afegir una condició adicional.

- En primer lloc imposarem  $\alpha_2 = 1$ . Això dona la taula de Butcher següent (on  $c_1 = c_2 = \frac{1}{2}$  i  $\beta_{2,1} = 1$ ):

$$\begin{array}{c|c} 0 & \\ \hline 1 & 1 \\ \hline & \frac{1}{2} \quad \frac{1}{2} \end{array}$$

Per tant, finalment, ens queda el mètode de Heun:

$$x_{k+1} = x_k + \frac{h}{2} \left( f(t_k, x_k) + f(t_k + h, x_k + hf(t_k, x_k)) \right)$$

En particular, acabem de demostrar que el mètode de Heun té ordre 2.

- Ara imposarem  $\alpha_2 = \frac{1}{2}$ . Això dona la taula de Butcher següent (on  $c_1 = 0$ ,  $c_2 = 1$  i  $\beta_{2,1} = \frac{1}{2}$ ):

$$\begin{array}{c|c} 0 & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \\ \hline & 0 \quad 1 \end{array}$$

En aquest cas obtenim el mètode d'Euler modificat o Runge-Kutta-2:

$$x_{k+1} = x_k + h f\left(t_k + \frac{h}{2}, x_k + \frac{h}{2} f(t_k, x_k)\right)$$

que també té ordre 2.

## Exercici llarguet i pesatet

Deduir un mètode de Runge-Kutta de 3 etapes i mostrar que té ordre 3.

# Mètodes de Runge-Kutta amb 4 etapes

## El mètode Runge-Kutta-4 Clàssic

El mètode de Runge-Kutta de 4 etapes és

$$x_{k+1} = x_k + h \sum_{i=1}^4 c_i \kappa_i(t_k, x_k),$$

$$\kappa_1(t_k, x_k) = f(t_k, x_k),$$

$$\kappa_2(t_k, x_k) = f(t_k + \alpha_2 h, x_k + h \beta_{2,1} \kappa_1(t_k, x_k)),$$

$$\kappa_3(t_k, x_k) = f(t_k + \alpha_3 h, x_k + h \beta_{3,1} \kappa_1(t_k, x_k) + h \beta_{3,2} \kappa_2(t_k, x_k)),$$

$$\kappa_4(t_k, x_k) = f\left(t_k + \alpha_4 h, x_k + h \sum_{j=1}^3 \beta_{4,j} \kappa_j(t_k, x_k)\right),$$

que té paràmetres  $c_1, c_2, c_3, c_4, \alpha_2, \alpha_3, \alpha_4$  i  $\beta_{i,j}$  amb  $i \in \{2, 3, 4\}$  i  $j \in \{1, 2, \dots, i-1\}$ .

# Mètodes de Runge-Kutta amb 4 etapes

## El mètode Runge-Kutta-4 Clàssic

De manera similar als mètodes Runge-Kutta de dues etapes (però amb més paciència) obtenim el sistema

$$\left\{ \begin{array}{l} c_1 + c_2 + c_3 + c_4 \\ \beta_{2,1} \\ \beta_{3,1} + \beta_{3,2} \\ c_2\alpha_2 + c_3\alpha_3 + c_4\alpha_4 \\ c_2\alpha_2^2 + c_3\alpha_3^2 + c_4\alpha_4^2 \\ c_2\alpha_2^3 + c_3\alpha_3^3 + c_4\alpha_4^3 \\ c_3\alpha_2\beta_{3,2} + c_4(\alpha_2\beta_{4,2} + \alpha_3\beta_{4,3}) \\ c_3\alpha_2\alpha_3\beta_{3,2} + c_4\alpha_4(\alpha_2\beta_{4,2} + \alpha_3\beta_{4,3}) \\ c_3\alpha_2^2\beta_{3,2} + c_4(\alpha_2^2\beta_{4,2} + \alpha_3^2\beta_{4,3}) \\ c_4\alpha_2\beta_{3,2}\beta_{4,3} \end{array} \right. \begin{array}{l} = 1, \\ = \alpha_2, \\ = \alpha_3, \\ = \frac{1}{2}, \\ = \frac{1}{3}, \\ = \frac{1}{4}, \\ = \frac{1}{6}, \\ = \frac{1}{8}, \\ = \frac{1}{12}, \\ = \frac{1}{24}. \end{array}$$

# Mètodes de Runge-Kutta amb 4 etapes

## El mètode Runge-Kutta-4 Clàssic

El sistema anterior té tretze incògnites en onze equacions. Per tant cal imposar dues condicions més. Les més útils, que donen l'anomenat *Mètode Runge-Kutta-4 Clàssic*, són

$$\alpha_2 = \frac{1}{2} \quad \text{i} \quad \beta_{3,1} = 0.$$

Resolent el sistema amb aquestes condicions addicionals obtenim la següent taula de Butcher:

0					0				
$\alpha_2$	$\beta_{2,1}$				$\frac{1}{2}$	$\frac{1}{2}$			
$\alpha_3$	$\beta_{3,1}$	$\beta_{3,2}$			$\frac{1}{2}$	0	$\frac{1}{2}$		
$\alpha_4$	$\beta_{4,1}$	$\beta_{4,2}$	$\beta_{4,3}$		1	0	0	1	
	$c_1$	$c_2$	$c_3$	$c_4$		$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$

# Mètodes de Runge-Kutta amb 4 etapes

Finalment el mètode Runge-Kutta-4 Clàssic és

$$x_{k+1} = x_k + \frac{h}{6} \left( \kappa_1(t_k, x_k) + 2\kappa_2(t_k, x_k) + 2\kappa_3(t_k, x_k) + \kappa_4(t_k, x_k) \right)$$

amb

$$\kappa_1(t_k, x_k) = f(t_k, x_k),$$

$$\kappa_2(t_k, x_k) = f\left(t_k + \frac{h}{2}, x_k + \frac{h}{2}f(t_k, x_k)\right)$$

$$\kappa_3(t_k, x_k) = f\left(t_k + \frac{h}{2}, x_k + \frac{h}{2}\kappa_2(t_k, x_k)\right),$$

$$\kappa_4(t_k, x_k) = f\left(t_k + h, x_k + h\kappa_3(t_k, x_k)\right).$$

## Observació

El fet que hi hagi alguns coeficients  $\beta$  igual a zero simplifica les fórmules i, per tant, redueix els *errors d'arrodoniment del mètode*.

# Mètodes de Runge-Kutta amb 4 etapes

## Altres mètodes RK-4

Observeu que, jugant amb les dues condicions addicionals que cal afegir al sistema d'equacions anterior, es poden construir altres mètodes RK-4. Un dels més anomenats és el *mètode Runge-Kutta-3/8*.

La taula de Butcher d'aquest mètode és:

0				
$\frac{1}{3}$	$\frac{1}{3}$			
$\frac{2}{3}$	$-\frac{1}{3}$	1		
1	1	-1	1	
	$\frac{1}{8}$	$\frac{3}{8}$	$\frac{3}{8}$	$\frac{1}{8}$

# Mètodes de Runge-Kutta amb 4 etapes

## Interpretació geomètrica del mètode RK-4 Clàssic

La successió  $\{x(t_k)\}_{k \in \mathbb{Z}^+}$ , de valors discretitzats de la solució  $x(t)$  d'un PVI als nodes de discretització  $\{t_k\}_{k \in \mathbb{Z}^+}$ , es pot obtenir per integració directa de l'EDO:

$$x(t_{k+1}) - x(t_k) = \int_{t_k}^{t_{k+1}} f(s, x(s)) ds.$$

Si apliquem la regla de Simpson per a aproximar la integral de la darrera equació obtenim

$$\int_{t_k}^{t_{k+1}} f(s, x(s)) ds \approx \frac{h}{6} \left( f(t_k, x(t_k)) + 4f\left(t_k + \frac{h}{2}, x\left(t_k + \frac{h}{2}\right)\right) + f(t_{k+1}, x(t_{k+1})) \right).$$

# Mètodes de Runge-Kutta amb 4 etapes

## Interpretació geomètrica del mètode RK-4 Clàssic

Ara volem relacionar els termes de l'expressió anterior amb els del mètode RK-4 clàssic. Tenim,

$$\kappa_1(t_k, x(t_k)) = f(t_k, x(t_k)),$$

i

$$\begin{aligned}\kappa_4(t_k, x(t_k)) &= f(t_k + h, x(t_k) + h \kappa_3(t_k, x(t_k))) \\ &\approx f(t_{k+1}, x(t_{k+1})),\end{aligned}$$

on la segona equació se segueix de la continuïtat d' $f$  i de

$$\begin{aligned}x(t_{k+1}) &= x(t_k + h) = x(t_k) + h \Delta(t_k, x(t_k)) \\ &\approx x(t_k) + h \kappa_3(t_k, x(t_k)),\end{aligned}$$

que és certa per  $h$  prou petit per la continuïtat de  $f$ ,  $\Delta$  i  $\kappa_3$ .

# Mètodes de Runge-Kutta amb 4 etapes

## Interpretació geomètrica del mètode RK-4 Clàssic

Finalment, el segon terme de la darrera expressió de la Pàgina 52,  $f\left(t_k + \frac{h}{2}, x\left(t_k + \frac{h}{2}\right)\right)$  és el pendent de la recta tangent a  $x(t)$  en el punt  $\left(t_k + \frac{h}{2}, x\left(t_k + \frac{h}{2}\right)\right)$ . Usant arguments similars als d'abans però per  $\frac{h}{2}$ , podem veure que

$$\kappa_2(t_k, x(t_k)) := f\left(t_k + \frac{h}{2}, x(t_k) + \frac{h}{2}f(t_k, x(t_k))\right), \text{ i}$$

$$\kappa_3(t_k, x(t_k)) := f\left(t_k + \frac{h}{2}, x(t_k) + \frac{h}{2}\kappa_2(t_k, x(t_k))\right)$$

són sengles aproximacions d' $f\left(t_k + \frac{h}{2}, x\left(t_k + \frac{h}{2}\right)\right)$ . Per tant podem prendre l'aproximació següent:

$$f\left(t_k + \frac{h}{2}, x\left(t_k + \frac{h}{2}\right)\right) \approx \frac{\kappa_2(t_k, x(t_k)) + \kappa_3(t_k, x(t_k))}{2}.$$

# Mètodes de Runge-Kutta amb 4 etapes

## Interpretació geomètrica del mètode RK-4 Clàssic

Resumint,

$$\begin{aligned}x(t_{k+1}) - x(t_k) &= \int_{t_k}^{t_{k+1}} f(t, x(s)) ds \\&\approx \frac{h}{6} \left( f(t_k, x(t_k)) + 4f\left(t_k + \frac{h}{2}, x\left(t_k + \frac{h}{2}\right)\right) + f(t_{k+1}, x(t_{k+1})) \right) \\&\approx \frac{h}{6} \left( \kappa_1(t_k, x(t_k)) + 4 \frac{\kappa_2(t_k, x(t_k)) + \kappa_3(t_k, x(t_k))}{2} + \kappa_4(t_k, x(t_k)) \right) \\&\approx \frac{h}{6} \left( \kappa_1(t_k, x(t_k)) + 2\kappa_2(t_k, x(t_k)) + 2\kappa_3(t_k, x(t_k)) + \kappa_4(t_k, x(t_k)) \right)\end{aligned}$$

que, substituint  $x(t_{k+1}) = x_{k+1}$  i  $x(t_k) = x_k$ , dona el mètode RK-4 Clàssic.

# Mètodes de Runge-Kutta amb 4 etapes

## Ordre del mètode RK-4 Clàssic

En conseqüència, l'error local de truncament del mètode RK-4 Clàssic es pot estimar a partir de l'error de la regla de Simpson:

$$\begin{aligned}h_k \tau(t_k, h_k, RK4) &= x(t_{k+1}) - x(t_k) \\ &- \frac{h_k}{6} \left( \kappa_1(t_k, x(t_k)) + 2\kappa_2(t_k, x(t_k)) + 2\kappa_3(t_k, x(t_k)) + \kappa_4(t_k, x(t_k)) \right) \\ &= -\frac{f^{(4)}(\xi, x(\xi))}{90} \left( \frac{h_k}{2} \right)^5 = -\frac{f^{(4)}(\xi, x(\xi))}{2880} h_k^5.\end{aligned}$$

Resumint,

$$\tau(t_k, h_k, RK4) = \mathcal{O}(h_k^4),$$

i *el mètode RK-4 Clàssic és d'ordre 4.*

# Mètodes de Runge-Kutta amb 4 etapes

## Ordre del mètode RK-4 Clàssic

A més, si la funció d'increment del mètode

$$\frac{1}{6} \left( \kappa_1(t, x) + 2\kappa_2(t, x) + 2\kappa_3(t, x) + \kappa_4(t, x) \right)$$

és Lipschitz contínua respecte de la segona variable, *el mètode RK-4 Clàssic és d'ordre de convergència 4.*

### Exercici

En quines condicions la funció d'increment del mètode de RK-4 Clàssic és Lipschitz contínua respecte de la segona variable.

### Observació

Amb el mètode RK-4 Clàssic ja dissenyat és molt més fàcil determinar així el seu ordre. Si estiguéssim deduïnt el mètode (com hem fet per exemple amb l'RK-2), el sistema d'onze equacions que s'obté ja ens diu quins termes del desenvolupament de Taylor de l'error local de truncament del mètode s'anul·len i, per tant, l'ordre.

# Sobre l'ordre dels mètodes de Runge-Kutta

La *barrera Butcher* diu que la quantitat d'etapes  $m$  creix més ràpidament que l'ordre  $p$ . En particular, *per a  $m \geq 5$  etapes, no hi ha cap mètode de Runge-Kutta explícit amb ordre de convergència  $m$*  (el sistema corresponent és irresoluble).

La taula següent, que confirma explícitament la barrera Butcher, mostra l'ordre dels mètodes de Runge-Kutta en funció de les etapes, fins a 11 etapes:

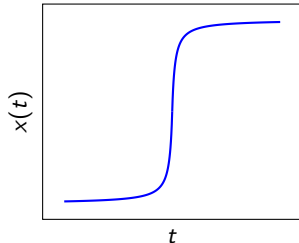
<b>Etapes</b>	1	2	3	4	5	6	7	8	9	11
<b>Ordre</b>	1	2	3	4	4	5	6	6	7	8

Notem que no és eficient usar el mètode RK-5 ja que amb l'RK-4 tenim el mateix ordre de convergència amb més rapidesa. Anàlogament és millor usar l'RK-7 que l'RK-8.

Com hem vist, la derivació i anàlisi dels mètodes numèrics per a PVI's normalment es basa en una mida fixa del pas  $h$  i el comportament de l'error,  $\mathcal{O}(h^p) = Ch^p$ , està garantit pel Teorema de Convergència.

Tot i així, no es coneix (l'ordre de magnitud de) la constant  $C$  que multiplica  $h^p$  a la fita  $\mathcal{O}(h^p)$  i, en conseqüència, en el fons no tenim estimacions fiables de l'error.

D'altra banda, l'estratègia de fixar un pas únic a tot l'interval d'integració  $I = [t_0, t_0 + T]$  no és la més òptima. En efecte, la mateixa solució pot ser “plana” en certs subinterval de temps de l'interval  $I$  i de ràpida variació o “complicada” en d'altres zones.



# Pas adaptatiu

Introducció (extreta parcialment de la Wikipedia)

Una manera de resoldre els dos problemes plantejats és la d'usar *mètodes amb pas adaptatiu*.

Els mètodes adaptatius estan dissenyats per produir una estimació fiable de l'error local de truncament a cada pas del mètode.

Durant la integració, la mida del pas  $h$  s'adapta de manera que l'error estimat es mantingui per sota d'un llindar definit per l'usuari: Si l'error és massa alt, es repeteix una iteració amb una mida de pas  $h$  inferior; si l'error és molt menor, s'augmenta la mida del pas  $h$  per estalviar iteracions. Això resulta en una mida d' $h$  (gairebé) òptima, que estalvia temps de càlcul. A més, l'usuari no ha de dedicar temps a trobar una mida de pas adequada.

En el cas del mètodes de Runge-Kutta l'estimació de l'error local de truncament s'obté usant dos mètodes diferents amb ordres de consistència consecutius. A més, aquests mètodes es poden entrellaçar (és a dir, tenen passos intermedis comuns) de manera que el cost computacional total és equivalent al del mètode amb més etapes. Gràcies a això, l'estimació de l'error té un cost computacional baix o insignificant en comparació amb l'avaluació dels dos mètodes.

Considerem un PVI:

$$\begin{cases} \dot{x} = f(t, x), \\ x(t_0) = x^*, \end{cases}$$

i un mètode numèric explícit d'un pas d'ordre de consistència  $p$ , per a calcular aproximacions  $x_k$  d' $x(t_k)$ :

$$\begin{cases} x_{k+1} = x_k + h \Phi(t_k, x_k, h), \\ x_0 = x(t_0) = x^*. \end{cases}$$

Amb aquest mètode numèric calculem dues solucions aproximades d' $x(t_k + h)$  a partir d' $x(t_k)$ . Una amb pas  $h$ :

$$x_{k+1}^{[1],h} := x(t_k) + h \Phi(t_k, x(t_k), h);$$

i l'altra iterant dues vegades el mètode amb pas  $\frac{h}{2}$ :

$$\begin{aligned} \tilde{x}_{k+1/2} &:= x(t_k) + \frac{h}{2} \Phi\left(t_k, x(t_k), \frac{h}{2}\right), \text{ i} \\ x_{k+1}^{[2],h/2} &:= \tilde{x}_{k+1/2} + \frac{h}{2} \Phi\left(t_k + \frac{h}{2}, \tilde{x}_{k+1/2}, \frac{h}{2}\right). \end{aligned}$$

# Estratègies generals de pas adaptatiu

Veurem que  $x_{k+1}^{[2],h/2}$  és millor aproximació d' $x(t_k + h)$  que  $x_{k+1}^{[1],h}$  (ja que té menor error absolut). Per tant, usarem  $x_{k+1}^{[2],h/2}$  com aproximació d' $x(t_k + h)$ , i  $x_{k+1}^{[2],h/2}$  i  $x_{k+1}^{[1],h}$  per a estimar l'error absolut de  $\left| x(t_k + h) - x_{k+1}^{[2],h/2} \right|$ .

Com que el mètode té ordre  $p$ , tenim

$$\begin{aligned} x(t+h) &= x(t) + h\Phi(t, x(t), h) + h\tau(t, h, \Phi) \\ &= x(t) + h\Phi(t, x(t), h) + C(t)h^{p+1} + \mathcal{O}(h^{p+2}). \end{aligned}$$

Per tant, usant el mètode amb pas  $h$ ,

$$x(t_k + h) = x_{k+1}^{[1],h} + C(t_k)h^{p+1} + \mathcal{O}(h^{p+2}).$$

D'altra banda, en iterar dues vegades el mètode amb pas  $\frac{h}{2}$ , tenim:

$$x\left(t_k + \frac{h}{2}\right) = \tilde{x}_{k+1/2} + C(t_k)\left(\frac{h}{2}\right)^{p+1} + \mathcal{O}(h^{p+2}).$$

Per tant,

$$\begin{aligned}x(t_k + h) &= x\left(t_k + \frac{h}{2}\right) + \frac{h}{2} \Phi\left(t_k + \frac{h}{2}, x\left(t_k + \frac{h}{2}\right), \frac{h}{2}\right) + \\&\quad C\left(t_k + \frac{h}{2}\right)\left(\frac{h}{2}\right)^{p+1} + \mathcal{O}(h^{p+2}) \\&= \tilde{x}_{k+1/2} + C(t_k)\left(\frac{h}{2}\right)^{p+1} + \frac{h}{2} \Phi\left(t_k + \frac{h}{2}, x\left(t_k + \frac{h}{2}\right), \frac{h}{2}\right) + \\&\quad C\left(t_k + \frac{h}{2}\right)\left(\frac{h}{2}\right)^{p+1} + \mathcal{O}(h^{p+2}) \\&\approx \tilde{x}_{k+1/2} + \frac{h}{2} \Phi\left(t_k + \frac{h}{2}, \tilde{x}_{k+1/2}, \frac{h}{2}\right) + \\&\quad 2C(t_k)\left(\frac{h}{2}\right)^{p+1} + \mathcal{O}(h^{p+2}) \\&= x_{k+1}^{[2], h/2} + \frac{1}{2^p} C(t_k) h^{p+1} + \mathcal{O}(h^{p+2}).\end{aligned}$$

En conseqüència,

$$x_{k+1}^{[2],h/2} - x_{k+1}^{[1],h} = C(t_k)h^{p+1} \left(1 - \frac{1}{2^p}\right) + \mathcal{O}(h^{p+2})$$

i, equivalentment,

$$C(t_k)h^{p+1} = \frac{x_{k+1}^{[2],h/2} - x_{k+1}^{[1],h}}{1 - 2^{-p}} + \mathcal{O}(h^{p+2}).$$

Resumint,

$$\begin{aligned} x(t_k + h) &= x_{k+1}^{[2],h/2} + \frac{1}{2^p} C(t_k)h^{p+1} + \mathcal{O}(h^{p+2}) \\ &= x_{k+1}^{[2],h/2} + \frac{x_{k+1}^{[2],h/2} - x_{k+1}^{[1],h}}{2^p - 1} + \mathcal{O}(h^{p+2}). \end{aligned}$$

# Estratègies generals de pas adaptatiu

Com ja hem dit, abusant una mica de la notació,  $x_{k+1}^{[2],h/2}$  és una aproximació d' $x(t_k + h)$  obtinguda a partir d' $x(t_k)$  amb un mètode  $\Phi$  que consisteix a iterar  $\Phi$  dues vegades amb pas  $\frac{h}{2}$ .

L'error absolut d'aquesta aproximació és d'ordre  $p + 1$ :

$$\left| x(t_k + h) - x_{k+1}^{[2],h/2} \right| = \frac{1}{2^p} \varepsilon_{k+1} + \mathcal{O}(h^{p+2}),$$

amb

$$\varepsilon_{k+1} := \frac{\left| x_{k+1}^{[2],h/2} - x_{k+1}^{[1],h} \right|}{1 - 2^{-p}} = |C(t_k)| h^{p+1} + \mathcal{O}(h^{p+2}).$$

**Observació:** aquest mètode d'estimar l'error no és eficient.

En primer lloc, requereix una quantitat important de càlcul: cal usar el mètode  $\Phi$  com a mínim tres vegades a cada pas del temps. El segon punt negatiu és que no tenim cap possibilitat de controlar l'error real de truncament del mètode (un ordre més alt no sempre significa una precisió més alta).

Amb aquesta nova determinació de l'error (local) podem calcular el pas més gran possible  $h_{\max}$  tal que l'error absolut de truncament en aproximar  $x(t_k + h_{\max})$  a partir d' $x(t_k)$  es mantingui per sota d'un *error màxim acceptable*  $\varepsilon^*$ , predefinit.

Dels càlculs anteriors s'obté,

$$\varepsilon^* \geq \left| x(t_k + h_{\max}) - x_{k+1}^{[2], h_{\max}/2} \right| \approx \frac{1}{2^p} |C(t_k)| h_{\max}^{p+1} = \frac{1}{2^p} \frac{\varepsilon_{k+1}}{h_{\max}^{p+1}} h_{\max}^{p+1},$$

que és equivalent a:

$$h_{\max} \approx h \left( \frac{2^p \varepsilon^*}{\varepsilon_{k+1}} \right)^{\frac{1}{p+1}}.$$

- Una primera conseqüència de la fórmula anterior és que si  $\frac{2^p \varepsilon^*}{\varepsilon_{k+1}} \approx 1$ ,  $h_{\max} \approx h$  i el pas  $h$  era correcte. El significat de *correcte* aquí és doble. D'una banda l'error absolut de truncament es manté per sota de l'error màxim acceptable  $\varepsilon^*$ ; i per l'altra, el pas és aproximadament el més gran possible amb aquesta restricció. Això implica que minimitzem el nombre d'iteracions del mètode i, per tant, el temps de càlcul.
- La desigualtat  $\frac{2^p \varepsilon^*}{\varepsilon_{k+1}} \geq 1$ , implica que el pas  $h$  era correcte des del punt de vista d'afitar l'error màxim acceptable però que, per eficiència, cal augmentar el pas al proper iterat.
- Alternativament,  $\frac{2^p \varepsilon^*}{\varepsilon_{k+1}} \leq 1$ , implica que el pas  $h$  **no** era correcte i, per tant, cal disminuir-lo i repetir els càlculs de les aproximacions  $x_{k+1}^{[1],h}$  i  $x_{k+1}^{[2],h/2}$ .

Com que la nostra estimació de l'error no és exacta, en comptes de comparar  $\frac{2^p \varepsilon^*}{\varepsilon_{k+1}}$  amb 1, l'hauríem de comparar amb un factor de seguretat  $\rho < 1$ .

Això dóna el següent algorisme de determinació del pas màxim per a la iteració següent:

$$h_{\max} = \rho h \left( \frac{2^p \varepsilon^*}{\varepsilon_{k+1}} \right)^{\frac{1}{p+1}},$$

amb  $\rho \approx 0.9$ , que dóna un algorisme de pas adaptatiu més empíric i senzill que l'anterior, en forma de tres regles.

# Estratègies generals de pas adaptatiu

Resumint: un algorisme de pas adaptatiu en forma de tres regles

Si  $\varepsilon_{k+1} > 2^p \varepsilon^*$

dividim el pas  $h$  per 2 ( $h \mapsto \frac{h}{2}$ ) i  
recalculem  $x_{k+1}^{[1],h}$ ,  $x_{k+1}^{[2],h/2}$  i  $\varepsilon_{k+1}$ .

Si  $\varepsilon_{k+1} \ll 2^p \varepsilon^*$

acceptem  $h$  i  $x_{k+1}^{[2],h/2}$  com a aproximació d' $x(t_{k+1})$  amb  
 $t_{k+1} := t_k + h$ ; multipliquem el pas  $h$  per 2 ( $h \mapsto 2h$ ) per  
a la iteració següent.

Si  $\varepsilon_{k+1} \leq 2^p \varepsilon^*$  però  $\varepsilon_{k+1} \not\ll 2^p \varepsilon^*$

acceptem  $h$  i  $x_{k+1}^{[2],h/2}$  com a approxi-  
mació d' $x(t_{k+1})$  amb  $t_{k+1} := t_k + h$ .  
No modifiquem  $h$ .

Com hem explicat a la introducció, usarem *dos mètodes de Runge-Kutta entrelaçats, amb ordres de consistència consecutius*. Aquesta estratègia, originalment inventada per Fehlberg, dóna lloc als mètodes anomenats de Runge-Kutta-Fehlberg.

A cada pas, es calculen i comparen dues aproximacions d' $x(t + h)$ , generades amb dos mètodes entrelaçats diferents; un d'ordre  $p$  i l'altre d'ordre  $p + 1$ . Per eficiència computacional cada un dels dos mètodes ha de tenir el mínim nombre d'etapes compatible amb l'entrelaçament i els ordres desitjats.

# Mètodes de Runge-Kutta entrelaçats amb pas adaptatiu

Mètodes de Runge-Kutta entrelaçats amb pas adaptatiu: L'aportació de Fehlberg

La parella de mètodes de Runge-Kutta entrelaçats es representa mitjançant l'anomenada *Taula de Butcher Estesa*:

0					
$\alpha_2$	$\beta_{2,1}$				
$\alpha_3$	$\beta_{3,1}$	$\beta_{3,2}$			
$\vdots$	$\vdots$	$\vdots$	$\ddots$		
$\alpha_m$	$\beta_{m,1}$	$\beta_{m,2}$	$\cdots$	$\beta_{m,m-1}$	
<hr/>					
	$c_1$	$c_2$	$\cdots$	$c_{m-1}$	$c_m$
	$c_1^*$	$c_2^*$	$\cdots$	$c_{m-1}^*$	$c_m^*$

# Mètodes de Runge-Kutta entrelaçats amb pas adaptatiu

## Mètodes de Runge-Kutta entrelaçats amb pas adaptatiu: L'aportació de Fehlberg

Les estimacions d' $x(t_k + h)$  donades pels dos mètodes entrelaçats a partir d' $x_k \approx x(t_k)$  (calculat en una iteració anterior) són:

$$x_{k+1}^{[p]} := x_k + h \Phi_{[p]}(t_k, x_k, h) = x_k + h \sum_{i=1}^m c_i \kappa_i(t_k, x_k), \quad i$$

$$x_{k+1}^{[p+1]} := x_k + h \Phi_{[p+1]}(t_k, x_k, h) = x_k + h \sum_{i=1}^m c_i^* \kappa_i(t_k, x_k),$$

amb

$$\kappa_1(t_k, x_k) = f(t_k, x_k), \quad i$$

$$\kappa_\ell(t_k, x_k) = f\left(t_k + \alpha_\ell h, x_k + h \sum_{j=1}^{\ell-1} \beta_{\ell,j} \kappa_j(t_k, x_k)\right),$$

per  $\ell = 2, 3, \dots, m$ .

Recordem que el mètode  $\Phi_{[p]}$  té ordre  $p$ , mentre que  $\Phi_{[p+1]}$  té ordre  $p + 1$ .

# Mètodes de Runge-Kutta entrelaçats amb pas adaptatiu

## Mètodes de Runge-Kutta entrelaçats amb pas adaptatiu: L'aportació de Fehlberg

La definició d'error local de truncament (acceptant  $x_k$  com el vertader valor d' $x(t_k)$ ) dóna:

$$\begin{aligned}x(t_k + h) &= x_{k+1}^{[p]} + h\tau(t_k, h, \Phi_{[p]}) \\ &= x_{k+1}^{[p]} + C_{[p]}(t_k)h^{p+1} + \mathcal{O}(h^{p+2}), \quad i\end{aligned}$$

$$\begin{aligned}x(t_k + h) &= x_{k+1}^{[p+1]} + h\tau(t_k, h, \Phi_{[p+1]}) \\ &= x_{k+1}^{[p+1]} + C_{[p+1]}(t_k)h^{p+2} + \mathcal{O}(h^{p+3}).\end{aligned}$$

En conseqüència, si denotem

$$\varepsilon_{k+1} := x_{k+1}^{[p+1]} - x_{k+1}^{[p]},$$

tenim

$$\begin{aligned}\varepsilon_{k+1} &= \left(x(t_k + h) - x_{k+1}^{[p]}\right) - \left(x(t_k + h) - x_{k+1}^{[p+1]}\right) = \\ & \quad C_{[p]}(t_k)h^{p+1} + \mathcal{O}(h^{p+2}).\end{aligned}$$

El primer requeriment fonamental que cal fer al mètode és que l'error de l'aproximació obtinguda estigui controlat. Més precisament, donat un *error màxim acceptable*  $\varepsilon^*$ , volem que l'error absolut de l'aproximació  $x_{k+1}^{[p+1]}$  d' $x(t_k + h)$  estigui per sota d' $\varepsilon^*$ .

Tenim,

$$\varepsilon^* \geq \left| x(t_k + h) - x_{k+1}^{[p+1]} \right| \approx |\varepsilon_{k+1}| \approx \left| C_{[p]}(t_k) h^{p+1} \right|.$$

Pel càlcul de l'error i el pas adaptatiu estem seguint



*Introduction to Numerical Analysis*, J. Stoer and R. Bulirsch,  
Springer-Verlag, New York, 1980.

<http://dx.doi.org/10.1007/978-1-4757-5592-3>

# Mètodes de Runge-Kutta entrelaçats amb pas adaptatiu

Mètodes de Runge-Kutta entrelaçats amb pas adaptatiu: L'aportació de Fehlberg

D'altra banda, seguint la mateixa filosofia de control de l'error, estem buscant el pas més gran possible  $h_{\text{nou}}$  tal que l'error de l'aproximació

$$x_{k+2}^{[p+1]}[h_{\text{nou}}] := x_{k+1}^{[p+1]} + h_{\text{nou}} \sum_{i=1}^m c_i^* \kappa_i(t_{k+1}, x_{k+1}^{[p+1]}, h_{\text{nou}})$$

d' $x(t_{k+1} + h_{\text{nou}})$  (on  $t_{k+1} = t_k + h$ ), calculada amb pas  $h_{\text{nou}}$  a partir de  $(t_{k+1}, x_{k+1}^{[p+1]})$ , també estigui controlat per  $\varepsilon^*$ :

$$\varepsilon^* \geq \left| x(t_{k+1} + h_{\text{nou}}) - x_{k+2}^{[p+1]}[h_{\text{nou}}] \right|.$$

De la pàgina anterior i de la darrera identitat de la Pàgina 73, com abans, deduïm:

$$\varepsilon^* \gtrsim \left| C_{[p]}(t_{k+1}) h_{\text{nou}}^{p+1} \right| \approx \left| C_{[p]}(t_k) \right| \left| h_{\text{nou}}^{p+1} \right| \approx \frac{|\varepsilon_{k+1}|}{|h^{p+1}|} \left| h_{\text{nou}}^{p+1} \right|.$$

Per tant,

$$h_{\text{nou}} \approx h \left( \frac{\varepsilon^*}{|\varepsilon_{k+1}|} \right)^{\frac{1}{p+1}}.$$

Com hem fet abans, donat que l'estimació de l'error no és exacta, es recomana implementar un factor de seguretat  $\rho < 1$  que dóna la següent fórmula per  $h_{\text{nou}}$ :

$$h_{\text{nou}} \approx \rho h \left( \frac{\varepsilon^*}{|\varepsilon_{k+1}|} \right)^{\frac{1}{p+1}},$$

amb  $\rho \approx 0.9$

### Observació

Alguns autors, basats en experimentació numèrica intensiva, recomanen la següent versió modificada de la fórmula anterior:

$$h_{\text{nou}} \approx \rho h \left( \frac{\varepsilon^* |h|}{|\varepsilon_{k+1}|} \right)^{\frac{1}{p}}.$$



# Mètodes de Runge-Kutta entrelaçats amb pas adaptatiu

El mètode Runge-Kutta-Fehlberg-7(8) en pseudocodi

Descripció  
dels paràmetres

paràmetre	tipus	valor
$t$	E/S	Node de discretització actual
$x$	E/S	Aproximació d' $x(t)$ actual (obtinguda amb l'RK-8)
$h$	E/S	A l'entrada: pas d'integració proposat. A la sortida: $h_{\text{NOU}}$ calculat internament
$h_{\text{min}}$	E	Valor absolut del pas mínim permès
$h_{\text{max}}$	E	Valor absolut del pas màxim permès
$\varepsilon$	S	Error estimat de l'aproximació obtinguda
$\varepsilon^*$	E	Error màxim acceptable
$f$	E	Funció que calcula el camp de vectors al punt $(t, x)$

**Llegenda de la segona columna:** **E** denota paràmetre d'entrada;  
**S** denota paràmetre de sortida.

## RKF7(8) en pseudocodi

procedure RKF7(8)( $t, x, h, h_{\text{min}}, h_{\text{max}}, \varepsilon, \varepsilon^*, f$ )

$\alpha[2 : 13]$	$\leftarrow \frac{2}{27}, \frac{1}{9}, \dots, 1$	} <i>Inicialització de la Taula de Butcher Ampliada i de <math>\rho</math></i>
$\beta[\ell = 2 : 13][1 : \ell - 1]$	$\leftarrow \frac{2}{27}, \frac{1}{36}, \frac{1}{12}, \dots, 1$	
$c[1 : 11]$	$\leftarrow \frac{41}{840}, 0, 0, 0, 0, \frac{34}{105}, \dots, 0$	
$c^*[1 : 13]$	$\leftarrow 0, 0, 0, 0, 0, \frac{34}{105}, \dots, \frac{41}{840}$	
$\rho$	$\leftarrow 0.9$	
$h \leftarrow \text{sign}(h) \cdot \min\{h_{\text{max}}, \max\{h_{\text{min}},  h \}\}$		<i>▷ Per seguretat forcem <math>h</math> dins del seu rang</i>
$\kappa[1] \leftarrow f(t, x)$		<i>▷ <math>\kappa_1</math> no depèn d'<math>h</math>; no cal que estigui al llaç principal</i>

# Mètodes de Runge-Kutta entrelaçats amb pas adaptatiu

## El mètode Runge-Kutta-Fehlberg-7(8) en pseudocodi (II)

**while true do**

**for**  $\ell$  from 2 to 13 **do**

$\kappa[\ell] \leftarrow f(t + \alpha[\ell] \cdot h, x + h \sum_{j=1}^{\ell-1} \beta[\ell][j] \cdot \kappa[j])$

**end for**

$x7 \leftarrow h \sum_{i=1}^{11} c[i] \cdot \kappa[i]$

$x8 \leftarrow h \sum_{i=6}^{13} c^*[i] \cdot \kappa[i]$

**if** isnan( $x8$ ) **or** isnan( $x7$ ) **then return error; end if**

$\varepsilon \leftarrow |x7 - x8|$

**if**  $\varepsilon < \varepsilon^*$  **then**

$t \leftarrow t + h$

$x \leftarrow x + x8$

**if**  $\varepsilon < \frac{\varepsilon^*}{2^8}$  **then**  $h \leftarrow \text{sign}(h) \cdot \min\{h_{\max}, 2\rho |h|\}$ ; **return; end if**

$h \leftarrow \text{sign}(h) \cdot \min\left\{h_{\max}, \max\left\{h_{\min}, |h| \cdot \rho \cdot \sqrt[8]{\frac{\varepsilon^*}{\varepsilon}}\right\}\right\}$

**return**

**end if**

▷ A partir d'aquí l'error és gran:  $\varepsilon \geq \varepsilon^*$ . En particular  $\sqrt[8]{\frac{\varepsilon^*}{\varepsilon}} \leq 1$ .

**if**  $|h| \leq h_{\min}$  **then**  $t \leftarrow t + h$ ;  $x \leftarrow x + x8$ ;  $h \leftarrow h_{\min}$ ; **return; end if**

$h \leftarrow \text{sign}(h) \cdot \max\left\{h_{\min}, |h| \cdot \rho \cdot \sqrt[8]{\frac{\varepsilon^*}{\varepsilon}}\right\}$

▷ Actualització d' $h$  quan l'error és gran, per a recalcul  $x7$  i la predicció  $x + x8$ .

Donat que  $|h| \leq h_{\max}$ , ja es compleix

$$|h| \cdot \rho \cdot \sqrt[8]{\frac{\varepsilon^*}{\varepsilon}} \leq \rho h_{\max} < h_{\max}.$$

**end while**

**end procedure**

▷ Un llaç infinit (a priori)

▷ Calculem la resta de  $\kappa_\ell$ 's (que depenen d' $h$ )

▷ El nivell d'error de la nova aproximació és acceptable. Sortim actualitzant  $t$ ,  $x$  i calculant la nova  $h$ .

▷ Actualitzem  $t$

▷ Actualitzem  $x$

Encara que l'error de la nova aproximació sigui molt petit, no volem que el nou pas sigui massa gran.

El controlem i sortim.

# Mètodes de Runge-Kutta entrelaçats amb pas adaptatiu

## El mètode Runge-Kutta-Fehlberg-7(8) en pseudocodi (II)

En aquesta situació l'error és gran i *no podem reduir el pas*. És a dir, *no es pot aconseguir una aproximació amb el nivell d'error desitjat*.

El pas no es pot reduir perquè

$$\min \left\{ h_{\max}, \max \left\{ h_{\min}, |h| \cdot \rho \cdot \sqrt[8]{\frac{\varepsilon^*}{\varepsilon}} \right\} \right\} = h_{\min},$$

que se segueix trivialment de la desigualtat

$$|h| \cdot \rho \cdot \sqrt[8]{\frac{\varepsilon^*}{\varepsilon}} \leq \rho h_{\min} < h_{\min}.$$

*El valor de l'error (de sortida) mesura el nivell de qualitat de l'aproximació, i informa de que aquest és baix.*

# Mètodes de Runge-Kutta entrelaçats amb pas adaptatiu

Una implementació del mètode **RKF7(8)** en dimensió 1

Implementació en **C** del mètode Runge-Kutta-Fehlberg-7(8): declaracions i inicialitzacions

ATENCIÓ: La declaració de les  $\beta$ 's és infinita i s'acaba dues pàgines més endavant

```
#define MIN(x,y) ((x) < (y) ? (x) : (y))
#define MAX(a,b) ((a) > (b) ? (a) : (b))

double eighthroot (double);

/* alpha[i]'s DICTIONARY:  $\alpha_{-1} = 0$ ;  $\alpha_i = \alpha_{i-2}$  for  $i=2,3,4,\dots,10$ ;
    $\alpha_{11} = 1$ ;  $\alpha_{12} = 0$ ;  $\alpha_{13} = 1$  */
static double alpha[9] = { 2.e0/27.e0, 1.e0/9.e0, 1.e0/6.e0, 5.e0/12.e0,
                          0.5e0, 5.e0/6.e0, 1.e0/6.e0, 2.e0/3.e0, 1.e0/3.e0 };

/* beta's DICTIONARY is intertwined with the declaration and initialization:
   *  $\beta_{2,1} = \beta_{21}$  */
static double beta21 = 2.e0/27.e0,
/*  $\beta_{3,j} = \beta_{3[j-1]}$  for  $j = 1,2$  */
   beta3[2] = { 1.e0/36.e0, 1.e0/12.e0 },
/*  $\beta_{4,1} = \beta_{41}$ ;  $\beta_{4,2} = 0$ ;  $\beta_{4,3} = \beta_{43}$  */
   beta41 = 1.e0/24.e0, beta43 = 1.e0/8.e0,
/*  $\beta_{5,1} = \beta_{51}$ ;  $\beta_{5,2} = 0$ ;
   *  $\beta_{5,3} = -\beta_{534}$ ;  $\beta_{5,4} = \beta_{534}$  */
   beta51 = 5.e0/12.e0, beta534 = 25.e0/16.e0,
/*  $\beta_{6,1} = \beta_{61}$ ;  $\beta_{6,2} = \beta_{6,3} = 0$ ;
   *  $\beta_{6,j} = \beta_{6[j-4]}$  for  $j = 4,5$  */
   beta61 = 5.e-2, beta6[2] = { 0.25e0, 0.2e0 },
```

# Mètodes de Runge-Kutta entrelaçats amb pas adaptatiu

Una implementació del mètode **RKF7(8)** en dimensió 1

Implementació en **C** del mètode Runge-Kutta-Fehlberg-7(8): declaracions i inicialitzacions (II)

```
/*       $\beta_{\{7,1\}} = \text{beta71}; \beta_{\{7,2\}} = \beta_{\{7,3\}} = 0;$ 
 *       $\beta_{\{7,j\}} = \text{beta7}[j-1]$  for j = 4,5,6 */
beta71 = -25.e0/108.e0,
beta7[3] = { 125.e0/108.e0, -65.e0/27.e0, 125.e0/54.e0 },
/*       $\beta_{\{8,1\}} = \text{beta81}; \beta_{\{8,2\}} = \beta_{\{8,3\}} = \beta_{\{8,4\}} = 0;$ 
 *       $\beta_{\{8,j\}} = \text{beta8}[j-5]$  for j = 5,6,7 */
beta81 = 31.e0/300.e0,
beta8[3] = { 61.e0/225.e0, -2.e0/9.e0, 13.e0/900.e0 },
/*       $\beta_{\{9,1\}} = \text{beta91}; \beta_{\{9,2\}} = \beta_{\{9,3\}} = 0;$ 
 *       $\beta_{\{9,j\}} = \text{beta9}[j-4]$  for j = 4,5,6,7,8 */
beta91 = 2.e0,
beta9[5] = { -53.e0/6.e0, 704.e0/45.e0, -107.e0/9.e0,
             67.e0/90.e0, 3.e0 },
/*       $\beta_{\{10,1\}} = \text{betax1}; \beta_{\{10,2\}} = \beta_{\{10,3\}} = 0;$ 
 *       $\beta_{\{10,j\}} = \text{betax}[j-4]$  for j = 4,5,6,7,8,9 */
betax1 = -91.e0/108.e0,
betax[6] = { 23.e0/108.e0, -976.e0/135.e0, 311.e0/54.e0,
            -19.e0/60.e0, 17.e0/6.e0, -1.e0/12.e0 },
/*       $\beta_{\{11,1\}} = \text{betaxi1}; \beta_{\{11,2\}} = \beta_{\{11,3\}} = 0;$ 
 *       $\beta_{\{11,j\}} = \text{betaxi}[j-4]$  for j = 4,5,6,7,8,9,10 */
betaxi1 = 2383.e0/4100.e0,
betaxi[7] = { -341.e0/164.e0, 4496.e0/1025.e0, -301.e0/82.e0,
              2133.e0/4100.e0, 45.e0/82.e0, 45.e0/164.e0,
              18.e0/41.e0 },
```

# Mètodes de Runge-Kutta entrelaçats amb pas adaptatiu

Una implementació del mètode **RKF7(8)** en dimensió 1

Implementació en **C** del mètode Runge-Kutta-Fehlberg-7(8): declaracions i inicialitzacions (III)

```
/*       $\beta_{\{12,1\}} = \text{betaxii1}; \beta_{\{12,2\}} = \beta_{\{12,3\}} = \beta_{\{12,4\}} = \beta_{\{12,5\}} = 0;$ 
 *       $\beta_{\{12,j\}} = \text{betaxii}[j-6]$  for j = 6,7,8,9,10;  $\beta_{\{12,11\}} = 0$  */
betaxii1 = 3.e0/205.e0,
betaxii[5] = { -6.e0/41.e0, -3.e0/205.e0, -3.e0/41.e0,
               3.e0/41.e0, 6.e0/41.e0 },

/*       $\beta_{\{13,1\}} = \text{betaxiii1}; \beta_{\{13,2\}} = \beta_{\{13,3\}} = 0;$ 
 *       $\beta_{\{13,j\}} = \text{betaxiii}[j-4]$  for j=4,5,6,7,8,9,10;
 *       $\beta_{\{13,11\}} = 0; \beta_{\{13,12\}} = 1$  */
betaxiii1 = -1777.e0/4100.e0,
betaxiii[7] = { -341.e0/164.e0, 4496.e0/1025.e0, -289.e0/82.e0,
                2193.e0/4100.e0, 51.e0/82.e0, 33.e0/164.e0,
                12.e0/41.e0 };

/*  $\beta$ 's end of declaration and initialization */

/* c78[i]'s DICTIONARY: c_1 = c78_0xi; c_2 = c_3 = c_4 = c_5 = 0;
 * c_i = c78[i-6] for i = 6,7,8,9,10;
 * c_{11} = c78_0xi; c_{12} = c_{13} = 0.
 *
 * c*_1 = c*_2 = c*_3 = c*_4 = c*_5 = 0;
 * c*_i = c78[i-6] for i = 6,7,8,9,10;
 * c*_{11} = 0; c*_{12} = c*_{13} = c78_0xi. */
static double c78[5] = { 34.e0/105.e0, 9.e0/35.e0, 9.e0/35.e0,
                       9.e0/280.e0, 9.e0/280.e0 },
c78_0xi = 41.e0/840.e0;

static double rho = 0.9;
```

# Mètodes de Runge-Kutta entrelaçats amb pas adaptatiu

Una implementació del mètode **RKF7(8)** en dimensió 1

Implementació en **C** del mètode Runge-Kutta-Fehlberg-7(8)

```
/* The procedure RKF78 integrates forward the vector field: it computes an
 * approximation of x(t + h) starting at x \approx x(t) WITH h POSITIVE.
 * This function assumes WITHOUT CHECKING IT that 0 < hmin <= h <= hmax. */
```

```
int RKF78_forward (
    double *t, double *x,
    double *h, double *err,
    double hmin, double hmax, double tol,
    void *ParmsStruct,
    void (*ODE)(double, double, double *, void *))
{
    double kappa[13]; // kappa's DICTIONARY: k_i = kappa[i-1]

    /* Computed outside of the loop coz it does depend on h
     * k_1 */      ODE(*t, *x, kappa, ParmsStruct);

    while (1) {
/* k_2 */      ODE(*t + alpha[0] * *h, *x + *h * beta21 * kappa[0],
                  kappa+1, ParmsStruct);
/* k_3 */      ODE(*t + alpha[1] * *h,
                  *x + *h * (beta3[0] * kappa[0] + beta3[1] * kappa[1]),
                  kappa+2, ParmsStruct);
/* k_4 */      ODE(*t + alpha[2] * *h,
                  *x + *h * (beta41 * kappa[0] + beta43 * kappa[2]),
                  kappa+3, ParmsStruct);
```

Que és més ràpid?:

- `betax[5] * kappa[8];`
- `kappa[8] / 12; o`
- `kappa[8] * 8.333333333333333333333333333333e-2;`

# Mètodes de Runge-Kutta entrelaçats amb pas adaptatiu

Una implementació del mètode **RKF7(8)** en dimensió 1

Implementació en **C** del mètode Runge-Kutta-Fehlberg-7(8): continuació del càlcul de les  $\kappa$ 's

```
/* k_5 */      ODE(*t + alpha[3] * *h,
                *x + *h * (beta51 * kappa[0] + beta534 * (kappa[3]-kappa[2])),
                kappa+4, ParamsStruct);
/* k_6 */      ODE(*t + alpha[4] * *h,
                *x + *h * (beta61 * kappa[0] + beta6[0] * kappa[3] +
                beta6[1] * kappa[4]),
                kappa+5, ParamsStruct);
/* k_7 */      ODE(*t + alpha[5] * *h,
                *x + *h * (beta71 * kappa[0] + beta7[0] * kappa[3] +
                beta7[1] * kappa[4] +beta7[2] * kappa[5]),
                kappa+6, ParamsStruct);
/* k_8 */      ODE(*t + alpha[6] * *h,
                *x + *h * (beta81 * kappa[0] + beta8[0] * kappa[4] +
                beta8[1] * kappa[5] + beta8[2] * kappa[6]),
                kappa+7, ParamsStruct);
/* k_9 */      ODE(*t + alpha[7] * *h,
                *x + *h * (beta91 * kappa[0] + beta9[0] * kappa[3] +
                beta9[1] * kappa[4] + beta9[2] * kappa[5] +
                beta9[3] * kappa[6] + beta9[4] * kappa[7]),
                kappa+8, ParamsStruct);
```

# Mètodes de Runge-Kutta entrelaçats amb pas adaptatiu

Una implementació del mètode **RKF7(8)** en dimensió 1

Implementació en **C** del mètode Runge-Kutta-Fehlberg-7(8): continuació del càlcul de les  $\kappa$ 's (II)

```
/* k_10 */    ODE(*t + alpha[8] * *h,
               *x + *h * (betax1 * kappa[0] + betax[0] * kappa[3] +
                           betax[1] * kappa[4] + betax[2] * kappa[5] +
                           betax[3] * kappa[6] + betax[4] * kappa[7] +
                           betax[5] * kappa[8]),
               kappa+9, ParamsStruct);
/* k_11 */    ODE(*t + *h,
               *x + *h * (betaxi1 * kappa[0] + betaxi[0] * kappa[3] +
                           betaxi[1] * kappa[4] + betaxi[2] * kappa[5] +
                           betaxi[3] * kappa[6] + betaxi[4] * kappa[7] +
                           betaxi[5] * kappa[8] + betaxi[6] * kappa[9]),
               kappa+10, ParamsStruct);
/* k_12 */    ODE(*t,
               *x + *h * (betaxii1 * kappa[0] + betaxii[0] * kappa[5] +
                           betaxii[1] * kappa[6] + betaxii[2] * kappa[7] +
                           betaxii[3] * kappa[8] + betaxii[4] * kappa[9]),
               kappa+11, ParamsStruct);
/* k_13 */    ODE(*t + *h,
               *x + *h * (betaxiii1 * kappa[0] + betaxiii[0] * kappa[3] +
                           betaxiii[1] * kappa[4] + betaxiii[2] * kappa[5] +
                           betaxiii[3] * kappa[6] + betaxiii[4] * kappa[7] +
                           betaxiii[5] * kappa[8] + betaxiii[6] * kappa[9] +
                           kappa[11]),
               kappa+12, ParamsStruct);
```

# Mètodes de Runge-Kutta entrelaçats amb pas adaptatiu

Una implementació del mètode **RKF7(8)** en dimensió 1

Implementació en **C** del mètode Runge-Kutta-Fehlberg-7(8): El nucli del pas adaptatiu

```
double x7 = c78[0] * kappa[5] + c78[1] * kappa[6] + c78[2] * kappa[7] +
           c78[3] * kappa[8] + c78[4] * kappa[9],
       x8 = x7 + c78_0xi * (kappa[11] + kappa[12]);
       x7 += c78_0xi * (kappa[0] + kappa[10]);

if (isnan(x7) || isnan(x8)) return 66;

*err = *h * fabs(x8 - x7);
x8 = *x + *h * x8;

double tolrel = tol * (1 + fabs(x8) / 100.0);
if (*err < tolrel) { *t += *h; *x = x8;
    if (*err <= ldexp(tolrel, -8)) {
        *h *= (rho + rho); *h = MIN(hmax, *h);
    }
    *h *= rho * eighthroot(tolrel / *err);
    *h = MIN(hmax, MAX(hmin, *h))
    return 0;
}

if (*h <= hmin) { *t += *h; *x = x8; *h = hmin; return 0; }
*h *= rho * eighthroot(tolrel / *err); *h = MAX(hmin, *h);
} // End of main while loop; going to recompute the RK78 estimates
return 0;
} /* and this is the end */
```

L'error màxim acceptable **tol** es "tuneja" en funció de **fabs(x8)** penalitzat dos dígits.

• Quan **fabs(x8)** és petit, **tolrel**  $\approx$  **tol** i, com diu la teoria, s'usa com **error absolut**.

• Quan **fabs(x8)** és gran,

**tolrel** = **tol** · (1 +  $\delta$ ) amb  $\delta = \frac{\text{fabs}(x8)}{100}$ .

Com que **tol** > 0,  $\delta = \frac{\text{tolrel} - \text{tol}}{\text{tol}}$  i **tolrel** és una aproximació de **tol** amb **error relatiu**  $\delta = \frac{\text{fabs}(x8)}{100}$ .

# Mètodes de Runge-Kutta entrelaçats amb pas adaptatiu

Una implementació del mètode **RKF7(8)** en dimensió 1

Exercici fàcil i mecànic de programació (temps indicatiu 5 minuts)

Programar una variant de l'**RKF7(8)**, que es podria dir

```
int RKF78backward (double *t, double *x, double *h, double *err,  
                  double hmin, double hmax, double tol,  
                  void *ParmsStruct,  
                  void (*ODE)(double, double, double *, void *))
```

que, mantenint la  $h$  positiva, integri a temps negatiu.

Exercici mecànic però no tan fàcil

Programar l'**RKF7(8)** per camps vectorials (sistemes).

Considerem el següent problema de valor inicial

$$\begin{pmatrix} \dot{w}(t) \\ \dot{s}(t) \end{pmatrix} = \begin{pmatrix} s(t) \\ \alpha w(t)^r \end{pmatrix} \quad \text{amb} \quad w(t_0) = w_0 \text{ i } s(t_0) = s_0,$$

on  $\alpha$  i  $r$  són paràmetres.

Volem integrar aquest PVI prenent  $w_0 = 4$ ,  $\alpha = \frac{3}{2}$  i  $r = 2$ , per diferents valors d' $s_0$ , des de  $t_0 = 0$  fins  $t_{\text{final}} = 1$ .

Els resultats es mostren a la taula següent

$s_0$	$w(t_{\text{final}})$	Error Màxim
10	2.00066e+24	1.79769e+308 ( $t_{\text{final}} = 0.966803$ )
2	199.191416	1.41689e-14
0	87.080122	4.98599e-15
-2	40.780432	1.95395e-15
-5	12.057576	7.51366e-16
-10	-2.400837	9.10855e-16

**Atenció:** Què ha passat a la primera línia de la taula (per  $s_0 = 10$ ).

Mirar el significat d'**Error Màxim** directament al codi.

## L'inici del programa

**Integra2tfin**: un procediment prescindible que es podria posar al **main**

```
#include <stdio.h>
#include "RKF78.h"

double
Integra2tfin (double *t, double w[], double tfin, void *Parms,
              void (*CV) (double, double *, unsigned, double *, void *))
{
    double hmin = 1.e-8, hmax = 1.0, h = 1.e-6, err, maxerr = -1.0;

    while (*t + h < tfin)
    {
        if (RKF78_VF_forward
            (t, w, 2, &h, &err, hmin, hmax, 1.e-15, Parms, CV))
            return MAXDOUBLE;
        maxerr = MAX (maxerr, err);
    }

    double tt = *t;
    h = tfin - *t;
    *t = tfin;
    maxerr = MAX (maxerr, RKF78_VF_1step(tt, w, 2, h, Parms, CV));
    return maxerr;
}
```

"Header" condicionat per les crides als procediments d'integració **RKF7(8)**.

Aquest **while** és la solució estàndard d'un dels problemes típics del pas variable: "clavar" el temps final.

Aquí és realment on s'integra **exactament** fins el temps final.

Volem que **\*t** retorni el temps final d'integració (especialment en cas d'error).

# Un exemple complet i comentat d'integració d'un PVI

És molt mala idea que una potència sigui un paràmetre. La funció `pow` és (moooolt) lenta i fa augmentar l'error. Per potències enteres que obligatòriament hagin de ser paràmetres això s'hauria de programar millor.

`struct` és un (l'únic?) contenidor universal. Admet variables de tots els tipus, vectorialitzades o no. És la manera més senzilla d'agrupar un `double` i un `int`.

## El programa (ara sí)

```
typedef struct { double alpha; int r; } ODE_Parameters;
```

`void`

```
CampVectorial (double t, double w[], unsigned n,  
               double f[], void *pars)
```

```
{  
    f[0] = w[1];  
    f[1] = ((ODE_Parameters *) pars)->alpha *  
           pow (w[0], ((ODE_Parameters *) pars)->r);
```

Forçament de tipus (`cast`). Per a poder usar `alpha` i `r` hem de passar `pars` de `void *` a `ODE_Parameters *`.

```
}  
  
int
```

```
main ()  
{  
    double max_error;  
    ODE_Parameters CVPars = {1.5e0, 2};  
    double t = 0.0,  
           s = -10.0,  
           w[2] = { 4.0, s };  
  
    max_error = Integra2tfin (&t, w, 1.0, &CVPars, CampVectorial);  
  
    printf ("s_0 =-%lf; - tfinal =-%lf; -w(tfin) =-%lf; - max_error =-%g\n",  
           s, t, w[0], max_error);  
  
    return 0;  
}
```

Imposada per la implementació de la **RKF7(8)**: `void*(double, double *, unsigned, double *, void *)`  
El fet de declarar `pars` com `void *` és un truc estàndard per a no condicionar el tipus de paràmetre que es passa (o dit d'altra manera, per passar qualsevol cosa). El preu que es paga és que després `pars` s'ha de convertir a alguna cosa útil (i coneguda) per la funció.

En aquest cas, la `t` i la `n` no s'usen (camp autònom).

Cal passar un **apuntador** a l'estructure `CVPars`.

Molt malament que `s0` estigui "hardcoded" en comptes que sigui un paràmetre del `main`. Això obliga a compilar el programa cada vegada que cal canviar `s0`.

## Observació terminològica

Tenim un problema amb la paraula *paràmetres*. A la discussió següent, la paraula surt fent referència al conjunt de paràmetres d'una EDO barrejada amb referències als paràmetres de la crida a un procediment informàtic (o subrutina). Per evitar confusions, encara que sigui una mica pesat, usarem la següent terminologia que, de fet, és bastant òbvia:

- Conjunt de paràmetres d'una EDO.
- Paràmetre o paràmetres de capçalera d'una subrutina.

## Exercici fàcil i mecànic de programació

Modificar el programa anterior de manera que en comptes de donar, com ara, l'aproximació  $w(t_{fin})$  de la solució a  $t = t_{fin}$  doni les aproximacions (i temps) obtinguts a tots el passos d'integració, de manera que es pugui dibuixar la solució aproximada entre  $t_0 = 0$  i  $t = t_{fin}$ . A més, ja que hi estem posats, dibuixar les solucions.

# Un exemple complet i comentat d'integració d'un PVI

Comentaris detallats sobre les subrutines **RKF7(8)**: Els paràmetres de capçalera

paràmetre de capçalera	tipus	valor
<code>*t</code>	E/S	Node de discretització actual
<code>x</code>	E/S	Aproximació actual d' $x(t)$ (obtinguda amb l' <b>RKF8</b> )
<code>dim</code>	E	Dimensió del vector $x$ i de la funció $f$
<code>h</code>	E/S	A l'entrada: pas d'integració proposat. A la sortida: $h_{nou}$ calculat internament
<code><math>\epsilon = *err</math></code>	S	<b>Norma</b> de l'error estimat de l'aproximació obtinguda
<code><math>h_{min}</math></code>	E	Valor absolut del pas mínim permès
<code><math>h_{max}</math></code>	E	Valor absolut del pas màxim permès
<code><math>\epsilon^* = tol</math></code>	E	Error màxim acceptable <b>en norma</b>
<code>*Parms</code>	E	Apuntador (pot ser <b>NULL</b> ) al contenidor del conjunt de paràmetres de l'EDO $f$ . Aquesta funció, òbviament, està programada depenent del contingut de <code>*Parms</code>
<code>*VF</code>	E	(Apuntador a la) Funció $f$ , que avalua el camp de vectors al punt $(t, x(t))$ . Aquest procediment <b>ha de tenir</b> el següent prototipus: <code>void (*)(double, double *, unsigned, double *, void *)</code> o, per més claredat (encara que els noms de les variables no estan fixats), la següent capçalera: <code>void (*)(double t, double x[], unsigned dim, double f[], void *Parms)</code>

**Tipus de paràmetres de capçalera de les RKF7(8):** **E** = Entrada; **S** = Sortida

# Un exemple complet i comentat d'integració d'un PVI

Comentaris detallats sobre els procediments d'integració **RKF7(8)**

## El procediment **RKF78\_VF\_forward**

```
int RKF78_VF_forward (double *t, double x[], unsigned dim,
    double *h, double *err,
    double hmin, double hmax, double tol,
    void *Parms,
    void (*VF)(double, double *, unsigned, double *, void *))
```

És el procediment **Runge-Kutta-Fehlberg-7(8)** vist anteriorment (veure les pàgines 79, 81 i següents), adaptat per camps de vectors (és a dir per dimensió més gran que 1) i amb integració endavant (és a dir, amb  $t \mapsto t + h$ ).

Observem com s'utilitza dins d'**Integra2tfin** per aproximar el temps final d'integració final sense superar-lo.

# Un exemple complet i comentat d'integració d'un PVI

Comentaris detallats sobre els procediments d'integració **RKF7(8)**

El procediment **RKF78\_VF\_1step**: És realment necessari?

```
double RKF78_VF_1step (double t, double x[], unsigned dim,
                      double h, void *Parms,
                      void (*VF)(double, double *, unsigned, double *, void *))
```

És una simplificació del procediment **RKF78\_VF\_forward** sense el control automàtic de pas. Solament calcula la predicció **RKF8** a  $t + h$  a partir d' $x(t)$ , juntament amb l'error de predicció  $\epsilon = *err$ . Observem que,  $t$  i  $h$  són paràmetres (només) d'entrada de la subrutina **RKF78\_VF\_1step**,  $h_{min}$ ,  $h_{max}$  i  $\epsilon^* = tol$  no són necessaris i  $\epsilon = *err$  s'obté com a valor de retorn.

**NOTA:** Això també es podria fer amb el codi

```
h = tfin - *t;
RKF78_VF_forward
(t, w, 2, &h, &err, hmin, hmax, 1.e-15, Parms, CV);
```

però hi ha un problema: el procediment **RKF78\_VF\_forward** té permís per canviar el pas  $h$ . El pas  $h_{nou}$ , s'ha calculat a la iteració anterior (la darrera del **while**) al node  $*t - h_{vell}$ , amb el  $w(t)$  anterior. Llavors,  $h = tfin - *t$  és més petit que el pas  $h_{nou}$  però l'**RKF78\_VF\_forward**, internament, pot decidir recalculer  $h$  i donar una predicció de la solució a un punt  $*t < tfin$ .

# Perquè i coms del paràmetre de capçalera `void *Parms`

## Quin és el problema?

Els procediments d'integració **RKF7(8)** necessiten cridar les funcions que calculen les EDO's i Camps de Vectors (veure les pàgines 79 i 84–86).

Si aquestes funcions usen un conjunt de paràmetres per avaluar l'EDO, cal passar els hi des del mòdul que crida els procediments **RKF7(8)**.

Una pràctica habitual per a fer això és usar variables globals; **un estil dolent i perillós de programació**. La manera correcta de fer-ho és passar el conjunt de paràmetres de l'EDO a l'**RKF7(8)** i programar-la de manera que l'**RKF7(8)**, al seu torn, el passi a les funcions que avaluen les EDOS.

El problema d'això és que les EDO's tenen moltes classes de conjunts paràmetres: des de *no hi ha paràmetres* fins a un barrejat de tipus diferents de dades; vectors o no (com a l'exemple que estem discutint, on hem forçat artificialment que els paràmetres siguin un **double** i un **int**).

### El paràmetre de capçalera `void *Parms` de les subrutines **RKF7(8)**

Dóna una manera general, independent de les diferents combinacions possibles de tipus de dades, de passar un conjunt de paràmetres a les EDO's a través de l'**RKF7(8)**.

Òbviament, en aquest context, cal unificar part de les capçaleres dels procediments **RKF7(8)** i de les funcions que calculen les EDO's i els Camps de Vectors. Així mateix el codi d'aquestes darreres funcions també està condicionat pel contingut de **\*Parms**.

Ho veurem amb uns exemples concrets (apart del programa anterior).

# Perquè i coms del paràmetre de capçalera `void *Parms`

Exemples concrets (a més del programa anterior) d'us de `void *Parms`

L'exemple més simple: no hi ha paràmetres

```
void CV (double t, double w[], unsigned n, double f[], void *Parms){
    f[0] = w[1]; f[1] = 1.5e0 * w[0] * w[0];
}

int main () {
    RKF78_VF_forward (., ., ., ., ., ., ., ., NULL, CV);
}
```

Exemple d'un sol paràmetre (també és pot fer amb **typedef struct**)

```
void CV (double t, double w[], unsigned n, double f[], void *Parms){
    f[0] = w[1]; f[1] = *((double *) Parms) * w[0] * w[0];
}

int main () { double alpha = 1.5e0;
    RKF78_VF_forward (., ., ., ., ., ., ., ., &alpha, CV);
}
```

Exemple de paràmetres vectoritzats del mateix tipus i amb nombre fixat

```
void CV (double t, double w[], unsigned n, double f[], void *Parms){
    f[0] = w[1];
    f[1] = ((double *) Parms)[0] * pow (w[0], ((double *) Parms)[1]);
}

int main () { double CVPars[2] = { 1.5e0, 2.0 };
    RKF78_VF_forward (., ., ., ., ., ., ., ., CVPars, CV);
}
```

Atenció: un vector ja és un apuntador. No cal `&`.

# Perquè i coms del paràmetre de capçalera `void *Parms`

Recapitulant: com funciona

El procés de passar un conjunt de paràmetres a les EDO's a través dels procediments **RKF7(8)** es basa en el següent:

- 1 Cal “empaquetar” el conjunt de paràmetres de l'EDO en *un únic contenidor de paràmetres*. Quan el conjunt de paràmetres té estructura complicada (diversos tipus de dades; vectors i dimensions dels vectors, ...) cal crear un nou tipus de dades dedicat amb **typedef struct** (veure el programa d'exemple a la pàgina 91).
- 2 El prototip de les subrutines **RKF7(8)** obliga a passar el contenidor del conjunt de paràmetres de l'EDO com a apuntador, sense especificar-ne el tipus. És a dir, com a `void *`. Si el conjunt de paràmetres de l'EDO és buit, es passa **NULL** com a paràmetre de capçalera de la subrutina.

# Perquè i coms del paràmetre de capçalera `void *Parms`

Recapitulant: com funciona (cont.)

- 3 Les funcions que calculen les EDO's i els Camps de Vectors reben el contenidor del conjunt de paràmetres igual com es passa: com a `void *`. Com a conseqüència l'EDO no coneix el tipus (ni el contingut) del contenidor del conjunt de paràmetres i, per a poder-lo usar, cal fer un forçament de tipus (o `cast`) que el passi de `void *` al tipus correcte predefinit (veure les funcions `CampVectorial` i `CV` als quatre exemples anteriors).  
Òbviament, aquests `cast`'s estan determinats per la declaració del tipus de dades del contenidor del conjunt de paràmetres de l'EDO.

Perquè cal passar el contenidor del conjunt de paràmetres com `apuntador`

Un primer motiu senzill és que els apuntadors admeten el valor `NULL = no definit`. Amb variables caldria definir un codi específic de `no hi ha conjunt de paràmetres`.

El motiu important és que *quasi mai* s'ha de passar una estructura a una subrutina. Cal passar un *apuntador a l'estructura*.

Recordem que, en **C**, els paràmetres de capçalera de les subrutines es passen *per valor*. Això vol dir que, quan un programa crida una subrutina i l'hi passa una variable com a paràmetre de capçalera, el contingut de la variable es còpia a la variable local corresponent de la subrutina. En particular si estem passant, per exemple, una estructura que ocupa 2MB en memòria, estem copiant *per valor* 2MB cada vegada que cridem la subrutina. En canvi, si passem un apuntador a l'estructura estem copiant solament els 8 bytes de l'adreça inicial de l'estructura.

**GOOD LUCK**

