POLITECNICO DI MILANO

Lab





Esercitazione 03
Risoluzione numerica di ODE

Corso di Strumentazione e Controllo di Impianti Chimici

Prof. Davide Manca

Tutor: Giuseppe Pesenti

Metodi di Eulero – Esplicito e implicito

$$y'(t) = f(t, y(t))$$

$$y'(t_n) = f(t_n, y(t_n))$$

$$y'(t_n) = \frac{\Delta y(t_n)}{\Delta t_n} = \frac{\Delta y_n}{\Delta t_n}$$

$$y'(t_n) = \frac{y_{n+1} - y_n}{h} = f(t_n, y_n)$$

Metodo di Eulero forward (esplicito)

$$y_{n+1} = y_n + h \cdot f(t_n, y_n)$$

$$y'(t_n) = \frac{y_n - y_{n-1}}{h} = f(t_n, y_n)$$

$$\frac{y_{n+1} - y_n}{h} = f(t_{n+1}, y_{n+1})$$

$$y_{n+1} = y_n + h \cdot f(t_{n+1}, y_{n+1})$$

Metodo di Eulero backward (implicito)

Entrambi i metodi sono **metodi 1-step**: lo step successivo y_{n+1} è calcolato sulla base di **un singolo valore** nel passato y_n.

Espansioni in serie di Taylor – Metodi di Eulero

Espansione in serie di Taylor di $f(x)$ attorno a x_0:

$$P_n(x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k$$

Il polinomio $P_n(x)$ ha ordine di contatto n con la funzione $f(x)$ attorno a x_0.

$$y'(t_n) = f(t_n, y(t_n))$$

Effettuando un'espansione di Taylor di $y(t_{n+1})$ attorno a $y(t_n)$:

$$y(t_{n+1}) = y(t_n) + h \cdot y'(t_n) + \underbrace{\frac{1}{2}h^2 \cdot y''(t_n) + \dots}_{O(h^2)}$$

$$y(t_{n+1}) \approx y(t_n) + h \cdot y'(t_n)$$

1-step error $O(h^2)$: è l'errore che verrebbe introdotto a ogni passo se i valori di partenza y_n fossero esatti.

Metodo di Eulero forward

$$y_{n+1} = y_n + h \cdot f(t_n, y_n)$$

Integrando (nel tempo) su un numero di step proporzionale a $1/h$: si considera un **errore globale** $O(h^1)$

I metodi di Eulero hanno **accuratezza di ordine 1** e sono perciò chiamati **metodi di ordine 1**.

Espansioni in serie di Taylor – Ordine maggiore

Si può aumentare l'ordine del metodo mantenendo un **numero maggiore di termini** dell'espansione di Taylor di $y(t_{n+1})$ attorno a $y(t_n)$:

$$y(t_{n+1}) = y(t_n) + h \cdot y'(t_n) + \frac{1}{2} h^2 \cdot y''(t_n) + \dots$$

$$y'(t) = f(t, y(t))$$

E' necessario **calcolare esplicitamente le derivate di ordine superiore**.

$$y''(t) = \frac{d}{dt} f(t, y(t))$$

$$= f_y(t, y(t)) \cdot y'(t) + f_t(t, y(t))$$

$$y'''(t) = \dots$$

Diventa **complicato**!

Conviene utilizzare metodi che non richiedono il calcolo esplicito delle derivate di ordine superiore.

Metodi di ordine maggiore

- Metodi 1-step (multi-stage): **metodi di Runge-Kutta**
- Metodi multi-step: **metodi lineari multistep**

I termini corrispondenti a derivate di ordine superiore vengono approssimati grazie a un **approccio alle differenze finite**.

Metodi di ordine maggiore – Runge-Kutta (1/3)

- Metodi 1-step (multi-stage): **metodi di Runge-Kutta**
- Metodi multi-step: **metodi lineari multistep**

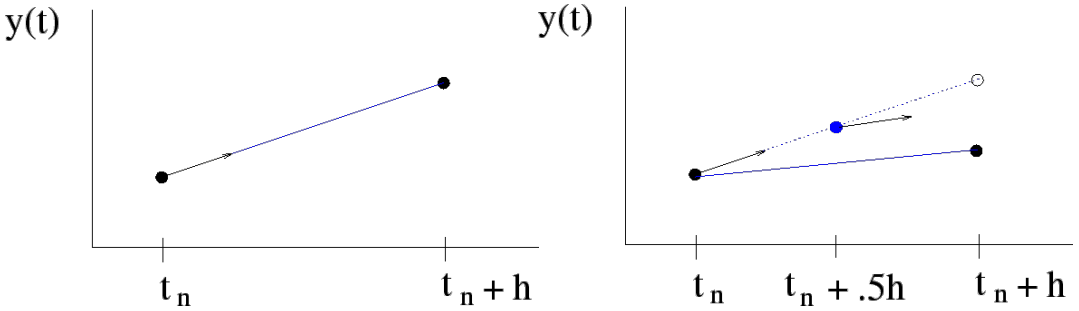
Esempio: metodo Runge-Kutta esplicito **2-stage** (Midpoint method)

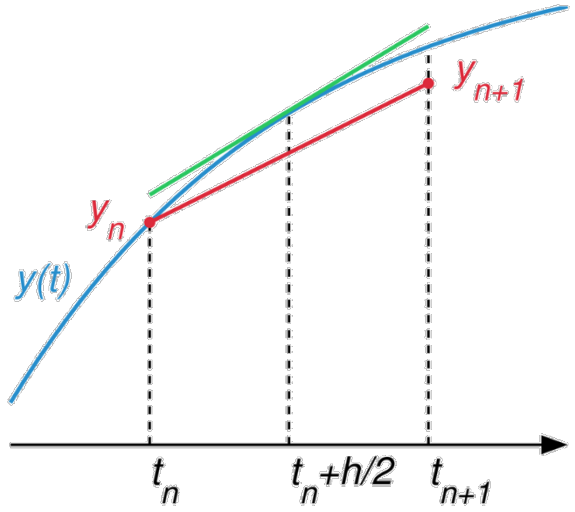
$$y'(t) = f(t, y(t))$$

$$y'(t_{n+1/2}) \approx \frac{y_{n+1} - y_n}{h}$$

$$y_{stage1} = y_{n+1/2} = y_n + \frac{1}{2} h \cdot f(t_n, y_n)$$

$$y_{stage2} = y_{n+1} = y_n + h \cdot f\left(t_{n+1/2}, y_{stage1}\right)$$





$$y_{n+1} = y_{stage2} = y_n + h \cdot f\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}h \cdot f(t_n, y_n)\right)$$

metodo esplicito 1-step con accuratezza di ordine 2

Metodi di ordine maggiore – Runge-Kutta (2/3)

- Metodi 1-step (multi-stage): **metodi di Runge-Kutta**
- Metodi multi-step: **metodi lineari multistep**

Esempio: metodo Runge-Kutta esplicito **4-stage** (RK4)

$$y'(t) = f(t, y(t))$$

y_n

$$F_0 = f(t_n, y_n)$$

$$F_1 = f\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hF_0\right)$$

$$F_2 = f\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hF_1\right)$$

$$F_3 = f(t_n + h, y_n + hF_2)$$

$$y_{n+1} = y_n + \frac{h}{6} \cdot (F_0 + 2F_1 + 2F_2 + F_3)$$

L'errore del singolo step è $O(h^5)$, l'errore globale è $O(h^4)$

RK4 è un **metodo esplicito 1-step con accuratezza di ordine 4**

Utilizzando questo approccio, è possibile sviluppare metodi di Runge-Kutta con ***n*-stage**.

Dato n, si scelgono i coefficienti delle posizioni degli n stage intermedi e, per confronto con la serie di Taylor di y_{n+1} attorno a y_n, si determinano i pesi dei valori di ogni stage in modo che la stima y_{n+1} abbia accuratezza di ordine n.

Metodi di ordine maggiore – Runge-Kutta (3/3)

- Metodi 1-step (multi-stage): **metodi di Runge-Kutta**
- Metodi multi-step: **metodi lineari multistep**

ode45

Metodo esplicito Runge-Kutta di **Dormand-Prince**

Utilizza **6 stage** per calcolare una soluzione con accuratezza di ordine 4 e una di ordine 5.

Usa la differenza tra le due per stimare l'errore della soluzione di **ordine 5**.

ode23

Metodo esplicito Runge-Kutta di **Shampine-Bogacki**

Utilizza **3 stage** per calcolare una soluzione con accuratezza di ordine 2 e una di ordine 3.

Usa la differenza tra le due per stimare l'errore della soluzione di **ordine 3**.

Metodi di ordine maggiore – Multistep (1/2)

- Metodi 1-step (multi-stage): **metodi di Runge-Kutta**
- Metodi multi-step: **metodi lineari multistep**

I metodi multistep cercano di aumentare l'efficienza del calcolo utilizzando le **informazioni degli step precedenti**, invece di scartarle.

I metodi multistep **lineari**, in particolare, utilizzano combinazioni lineari di questi valori.

Esempio: metodo di Adams-Bashforth **2-step**

$$y_{n+2} = y_{n+1} + \frac{3}{6} h \cdot f(t_{n+1}, y_{n+1}) - \frac{1}{2} h \cdot f(t_n, y_n)$$

Dato y_0, se y_1 non è noto, può essere calcolato con un metodo 1-step.

Approccio **Predictor-Corrector**:

- **Predictor**: stima della soluzione del passo successivo con un metodo esplicito
- **Corrector**: rifinitura della soluzione approssimata, solitamente con un metodo implicito

L'approccio **PECE** (Predictor-Evaluate-Corrector-Evaluate) chiama la funzione 2 volte per step.

Metodi di ordine maggiore – Multistep (2/2)

- Metodi 1-step (multi-stage): **metodi di Runge-Kutta**
- Metodi multi-step: **metodi lineari multistep**

ode113

Metodo esplicito e implicito lineare multistep PECE di **Adams-Bashforth-Moulton**

Utilizza un **singolo step** per calcolare una soluzione di **ordine da 1 a 13**.

Usa una formula di ordine 13 per **stimare l'errore** della soluzione.

Metodi di ordine maggiore – Confronto

- I metodi **multi-stage** Runge-Kutta effettuano **numerose chiamate alla funzione**.
  - Chiamano la funzione 6 volte (ode45) o 3 volte (ode23) per step (per ogni tentativo)
  - **Non riutilizzano** i valori della funzione calcolata negli stage interni
- I metodi **multistep** richiedono invece un **numero ridotto di chiamate alla funzione**.
  - Chiamano la funzione solo 2 volte (ode113) per step (per ogni tentativo)
  - **Riutilizzano** i valori degli step precedenti

1. Utilizzare **ode45** come prima scelta.

   Effettua più chiamate alla funzione a ogni step, ma può effettuare **passi più lunghi**.

   I passi adattivi delle soluzioni possono essere anche molto lunghi: è possibile richiedere al solver di **interpolare la soluzione** in un numero maggiore di punti, senza chiamate aggiuntive alla funzione

   In input al solver ode45, fornire un **vettore** (ad es. $t_0 : \Delta t : t_{fin}$) invece di $[t_0, t_{fin}]$

2. In casi con ampia tolleranza, **ode23** può essere più efficiente di ode45.

3. Se effettuare chiamate alla funzione è **costoso** (computazionalmente pesante) usare **ode113**

   In questi casi, il calcolo di stage intermedi utili solo al singolo step diventa inefficiente e lento (ode45, ode23). Conviene quindi utilizzare un metodo multistep (ode113)

Problemi stiff – Errore locale e errore globale

L'integrazione numerica si basa sull'ipotesi che l'errore introdotto a ogni step (**1-step error**)

$$O(h^{n+1})$$

sia ripetuto a ogni step e si trascini **accumulandosi** linearmente.

Sommando l'errore su un numero di step pari a circa $(t_{fin} - t_0)/h$, ci si aspetta che l'**errore globale complessivo** sia di un ordine inferiore.

$$O(h^{n})$$

Questo sarebbe garantito se fosse $y'(t) = f(t)$

Tuttavia, f dipende da y: $y'(t) = f(t, y(t))$

L'errore di ogni step influenza l'input della funzione allo step successivo → **problemi stiff**

L'**ordine atteso** dell'errore globale è garantito solo nel caso in cui il **metodo numerico** sia **stabile**: gli errori degli step precedenti non devono crescere eccessivamente negli step successivi.

Problemi stiff – Stabilità dei metodi numerici

In caso di **problemi stiff**, quindi, è necessario utilizzare metodi numerici con un'**ampia regione di stabilità**.

Tuttavia, per tutti i metodi espliciti:

si ha **garanzia di stabilità** solo se il passo h è minore della scala temporale più rapido.

In generale:

- **tutti i metodi espliciti** (tra cui ode45, ode23)
- e alcuni metodi impliciti come Adams-Moulton (usato da ode113)

hanno **regioni di stabilità limitate**: sono **inefficienti** per problemi stiff.

Questi metodi possono comunque essere utilizzati per risolvere problemi stiff, **ma** sono **estremamente inefficienti**:

trovano la soluzione corretta solo procedendo con passi di integrazione eccessivamente piccoli, impiegando un **tempo molto lungo**.

Problemi stiff – Identificazione

E' a priori **difficile prevedere** se un **problema è stiff**.

- Il rapporto tra gli autovalori $\frac{max|\lambda_P|}{min|\lambda_P|}$ della **matrice Jacobiana** $\frac{\partial f_i}{\partial y_j}$ non è condizione né necessaria né sufficiente perché il problema sia stiff
- Anche **una singola equazione** differenziale può essere stiff
- Un problema può essere stiff solo **in certi intervalli di tempo**

Per **risolvere un sistema di ODE** potenzialmente stiff:

1. tentare la risoluzione con un **solver esplicito: ode45**

2. se ode45 non trova la soluzione corretta e, riducendo le tolleranze, appare **inefficiente**

→ **ipotizzare** che si tratti di un **problema stiff**.
Utilizzare **metodi impliciti** sviluppati appositamente per problemi stiff.

Metodi impliciti

- Richiedono la soluzione numerica di **equazioni implicite** (metodo di Newton).
- Il risolutore deve effettuare **numerose chiamate** alla funzione a ogni step.

ode15s

Metodo implicito NDF di Klopfenstein-Shampine

Utilizza più step per calcolare una soluzione con accuratezza di ordine da 1 a 5.

ode23s

Metodo implicito di Rosenbrock modificato

Utilizza 1 step per calcolare una soluzione con accuratezza di ordine 2.

Usa una soluzione di ordine 3 per stimare l'errore locale.

In generale, **utilizzare ode15s**.

Se la tolleranza è ampia, e per alcuni particolari tipi di problemi stiff, ode23s può essere più efficiente.

Ode suite

Solver per problemi non stiff

ode45

ode23, ode113

Solver per problemi stiff

ode15s, ode23s

Bibliografia:

- **Finite Difference Methods for Differential Equations**, LeVeque, SIAM, 2007
- **Behind and Beyond the Matlab Ode Suite**, Ashino, Computers & Mathematics with Applications, 2000
- **The Matlab Ode Suite**, Shampine, SIAM, 1997