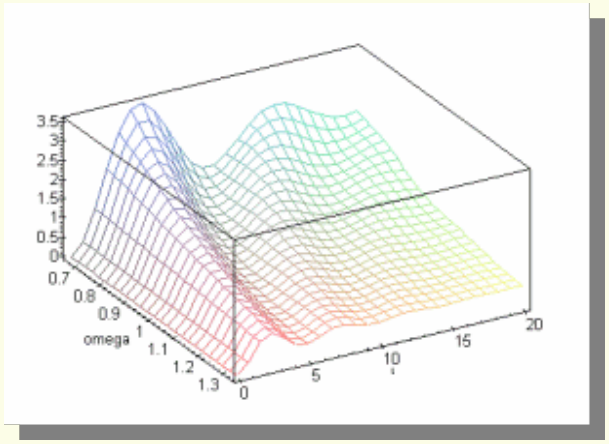
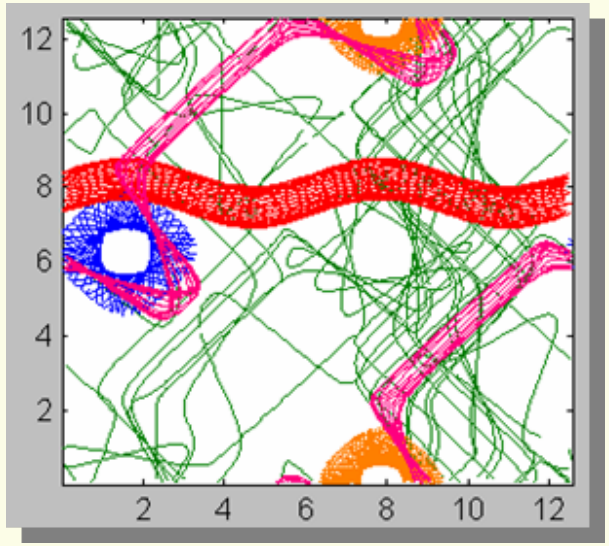
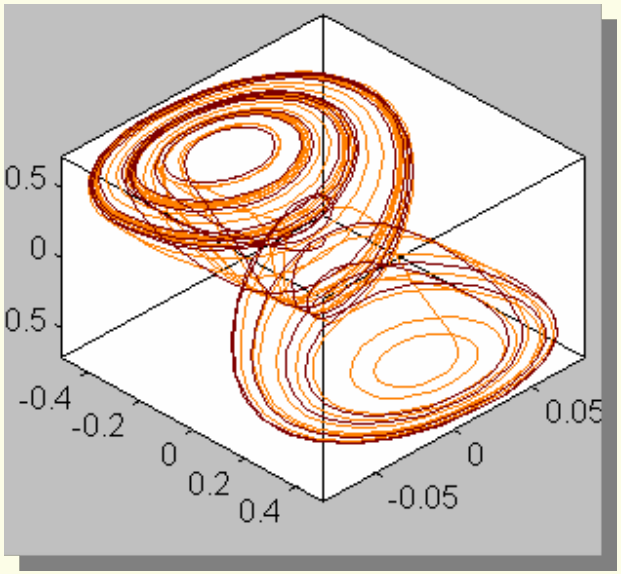
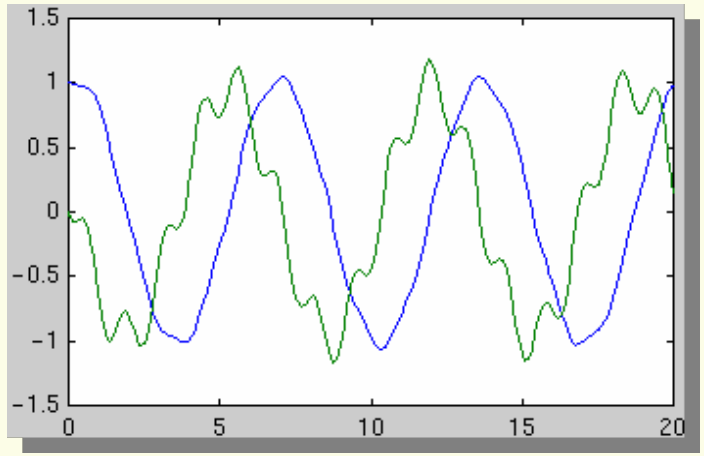
L8 Equazioni e sistemi differenziali ordinari con condizioni iniziali









Introduzione

Nella presente trattazione si considera il problema dell'integrazione di un sistema di **equazioni differenziali ordinarie**, **ODE**, del **primo ordine** in **forma esplicita**: $\mathbf{y}' = \mathbf{f}(\mathbf{y}, t)$ con le **condizioni iniziali**: $\mathbf{y}(t_0) = \mathbf{y}_0$

Il problema ai valori iniziali per una equazione o un sistema differenziale di ordine uno consiste nel trovare la funzione, $y(t)$, che soddisfa l'equazione:

$$\frac{dy(t)}{dt} = f\left(t, y(t)\right)$$

insieme alla condizione iniziale: $y(t_0) = y_0$

La soluzione numerica a questo problema genera una serie di valori per la variabile indipendente t_0, t_1, …, cui corrisponde una serie di valori per le variabili dipendenti y_0, y_1, …, tale che ogni y_n approssimi la soluzione analitica $y(t)$ nei punti t_n: $y_n \approx y(t_n) \qquad n = 0,1,\ldots$

Introduzione

I metodi numerici moderni determinano automaticamente l'ampiezza del passo di integrazione: $h_n = t_{n+1} - t_n$ in modo tale che la stima dell'errore della soluzione numerica sia controllata da una opportuna tolleranza.

Il **teorema fondamentale del calcolo integrale** fornisce un importante legame analitico tra equazioni differenziali ed integrali:

$$y(t+h) = y(t) + \int_t^{t+h} f\left(s, y(s)\right) ds$$

Si noti però che **non** è possibile utilizzare un qualsiasi metodo numerico di integrazione definita per approssimare l'integrale in quanto la funzione $y(s)$ non è nota e quindi non è definita la funzione integranda.

Nel caso speciale in cui $f(t,y(t))$ sia unicamente funzione di t, la soluzione numerica di tale equazione differenziale diviene una semplice sequenza di quadrature:

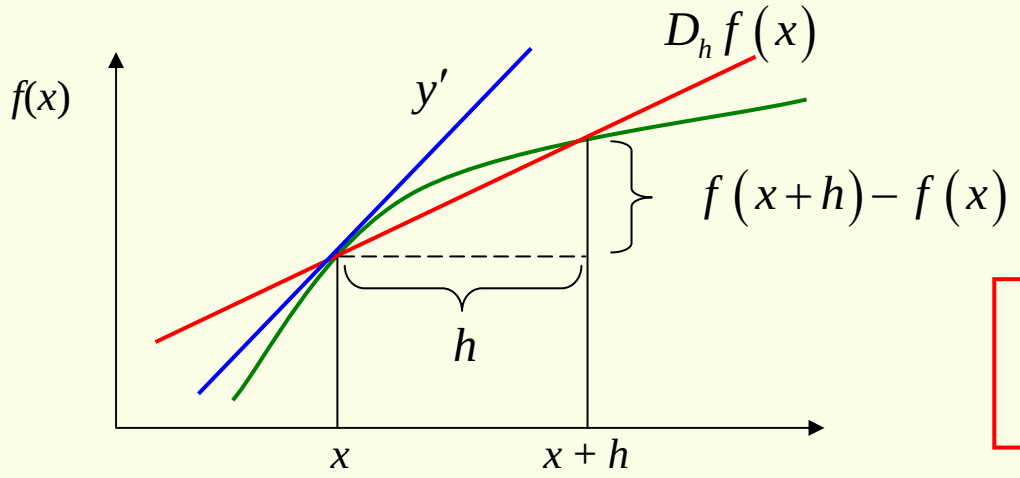
$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} f(s)\, ds$$

Calcolo numerico della derivata

Come appena visto, la valutazione numerica di un integrale non è in genere di aiuto alla risoluzione di sistemi ODE. È opportuno, viceversa, focalizzare l'attenzione sulla discretizzazione della derivata di una funzione e sul suo calcolo numerico. Questo argomento oltre ad essere utile in sé, è anche propedeutico alla integrazione di sistemi ODE.

Qualora sia necessario calcolare numericamente la **derivata di una funzione** $y = f(x)$, è possibile ricondursi alla sua definizione analitica intesa come limite del rapporto incrementale (se la funzione f è continua e se il limite esiste):

$$y' = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \quad \Rightarrow \quad y' \approx \frac{f(x+h) - f(x)}{h} = D_h f(x)$$



$D_h f(x)$
derivata numerica di $f(x)$

Calcolo numerico della derivata

Sviluppando in serie di Taylor è possibile scrivere:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2} f''(c) \qquad \text{con} \quad x \le c \le x+h$$

Quindi:

$$D_h f(x) = \frac{\left[f(x) + hf'(x) + 1/2\, h^2 f''(c) \right] - f(x)}{h} = f'(x) + \frac{h}{2} f''(c)$$

Perciò l'errore commesso approssimando la derivata prima diviene:

$$e_h = f'(x) - D_h f(x) = \frac{1}{2} h\, f''(c)$$

Si noti che analiticamente l'errore e_h dipende dall'ampiezza dell'intervallo con cui si calcola la derivata numerica.

Geometricamente parlando, la secante tende alla tangente riducendo la distanza dei punti, h, per i quali passa la secante.

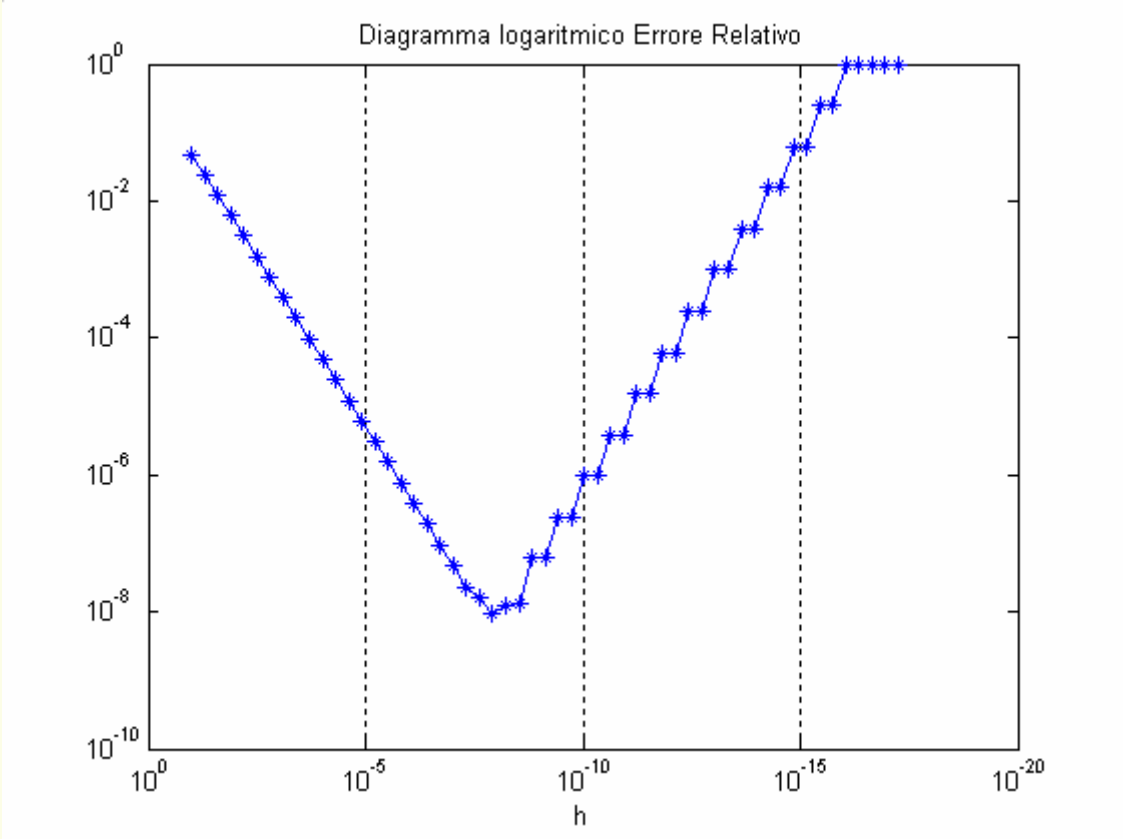
Apparentemente si potrebbe affermare che dimezzando l'ampiezza, h, dell'intervallo, l'errore, e_h, si dimezza. In realtà, calcolando numericamente la derivata di una funzione, subentra un ulteriore problema legato alla precisione del calcolatore.

Calcolo numerico della derivata

Il calcolo numerico della derivata di una funzione (nel caso specifico $y = log(x)$) evidenzia un andamento peculiare dell'errore commesso. L'errore non decrescere proporzionalmente con la riduzione dell'ampiezza dell'intervallo, h. Al contrario, l'errore perviene ad un minimo per poi crescere di nuovo raggiungendo il 100%.

N.B.: per quanto riguarda il grafico, l'asse delle ascisse è **rovesciato** ad indicare la dinamica del calcolo. Partendo da un valore di h pari a **`0.1`** questi viene ridotto fin oltre la precisione della macchina in doppia precisione: $\varepsilon \approx$ **`2.e-16`**.

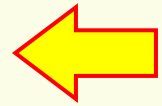
Dapprima l'errore relativo si riduce linearmente, su scala logaritmica, come previsto dalla formula. Raggiunge però un valore minimo e quindi prosegue aumentando a balzi fino ad una perdita completa di precisione della derivata numerica.



Calcolo numerico della derivata

Si seguito è riportato l'andamento della derivata numerica di $\log(x)$ in $x = 1$:

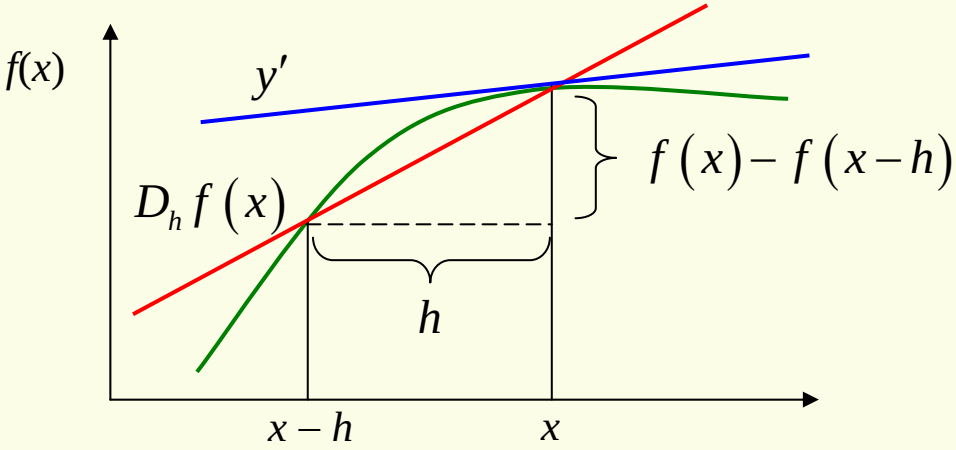
h	derivata numerica	errore relativo
1.000000000000000e-001	9.531017980432494e-001	4.689820195675065e-002
1.000000000000000e-002	9.950330853168092e-001	4.966914683190771e-003
1.000000000000000e-003	9.995003330834232e-001	4.996669165767687e-004
1.000000000000000e-004	9.999500033329731e-001	4.999666702687478e-005
1.000000000000000e-005	9.999950000398841e-001	4.999960115936730e-006
1.000000000000000e-006	9.999994999180666e-001	5.000819334188833e-007
1.000000000000000e-007	9.999999505838703e-001	4.941612974374010e-008
1.000000000000000e-008	9.999999889225290e-001	1.107747105155710e-008
1.000000000000000e-009	1.000000082240371e+000	8.224037073567558e-008
1.000000000000000e-010	1.000000082690371e+000	8.269037077290875e-008
1.000000000000000e-011	1.000000082735371e+000	8.273537055458746e-008
1.000000000000000e-012	1.000088900581841e+000	8.890058184074512e-005
1.000000000000000e-013	9.992007221625907e-001	7.992778374092957e-004
1.000000000000000e-014	9.992007221626357e-001	7.992778373643317e-004
1.000000000000000e-015	1.110223024625156e+000	1.102230246251559e-001
1.000000000000000e-016	0	1.000000000000000e+000



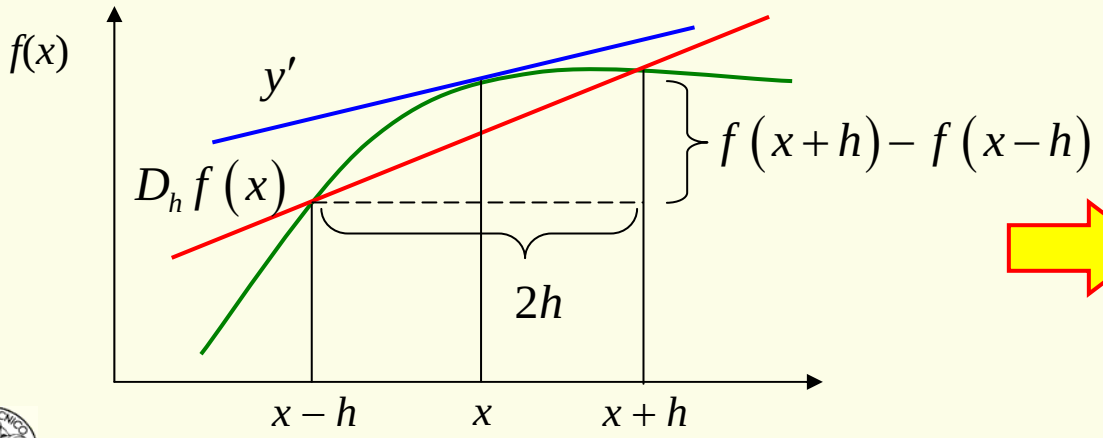
È fondamentale osservare che il minimo errore commesso nel calcolo della derivata numerica è ottenuto per un valore della perturbazione h dell'ordine di grandezza della radice quadrata del *macheps*. Dato che i calcoli sono stati condotti in doppia precisione allora: $\sqrt{macheps} = 1.490116119384765625\text{E-}8$

Calcolo numerico della derivata

In analogia con quanto visto in precedenza e riconducendoci al significato geometrico della derivata prima di una funzione è possibile introdurre le seguenti approssimazioni numeriche:



$$D_h f(x) = \frac{f(x)-f(x-h)}{h}$$



$$D_h f(x) = \frac{f(x+h)-f(x-h)}{2h}$$

Calcolo numerico della derivata

Riassumendo sono date le seguenti alternative per il calcolo numerico della derivata prima di una funzione:

Nome	Formula	Errore
FORWARD	$D_h f(x) = \frac{f(x+h) - f(x)}{h}$	$\varepsilon_h = \frac{1}{2} h f''(c_1)$
BACKWARD	$D_h f(x) = \frac{f(x) - f(x-h)}{h}$	$\varepsilon_h = \frac{1}{2} h f''(c_2)$
CENTRATA	$D_h f(x) = \frac{f(x+h) - f(x-h)}{2h}$	$\varepsilon_h = \frac{1}{6} h^2 f''(c_3)$

La formula centrata commette un errore che è proporzionale al quadrato della perturbazione: h. Al contrario le formule “forward” e backward” producono un errore proporzionale all’ampiezza dell’intervallo, h.

N.B.: qualora sia già noto il valore della funzione nel punto x, le formule “forward” e “backward” richiedono un solo ulteriore calcolo di funzione. Al contrario la formula “centrata” richiede due calcoli di funzione.

Calcolo numerico della derivata

Si seguito è riportato l'andamento della derivata numerica di $\log(x)$ in $x = 1$:

h	y'forward	y'backward	y'centrata	err forward	err backw.	err centrata
1.0000e-001	9.5310e-001	1.0536e+000	1.0034e+000	4.6898e-002	5.3605e-002	3.3535e-003
1.0000e-002	9.9503e-001	1.0050e+000	1.0000e+000	4.9669e-003	5.0336e-003	3.3335e-005
1.0000e-003	9.9950e-001	1.0005e+000	1.0000e+000	4.9967e-004	5.0033e-004	3.3333e-007
1.0000e-004	9.9995e-001	1.0001e+000	1.0000e+000	4.9997e-005	5.0003e-005	3.3332e-009
1.0000e-005	1.0000e+000	1.0000e+000	1.0000e+000	5.0000e-006	5.0000e-006	3.4333e-011
1.0000e-006	1.0000e+000	1.0000e+000	1.0000e+000	5.0008e-007	5.0003e-007	2.6422e-011
1.0000e-007	1.0000e+000	1.0000e+000	1.0000e+000	4.9416e-008	4.9474e-008	2.8759e-011
1.0000e-008	1.0000e+000	1.0000e+000	1.0000e+000	1.1077e-008	1.0025e-008	5.2636e-010
1.0000e-009	1.0000e+000	1.0000e+000	1.0000e+000	8.2240e-008	2.7782e-008	2.7229e-008
1.0000e-010	1.0000e+000	1.0000e+000	1.0000e+000	8.2690e-008	8.2790e-008	8.2740e-008
1.0000e-011	1.0000e+000	1.0000e+000	1.0000e+000	8.2735e-008	8.2745e-008	8.2740e-008
1.0000e-012	1.0001e+000	9.9998e-001	1.0000e+000	8.8901e-005	2.2122e-005	3.3389e-005
1.0000e-013	9.9920e-001	1.0003e+000	9.9976e-001	7.9928e-004	3.1095e-004	2.4417e-004
1.0000e-014	9.9920e-001	9.9920e-001	9.9920e-001	7.9928e-004	7.9928e-004	7.9928e-004
1.0000e-015	1.1102e+000	9.9920e-001	1.0547e+000	1.1022e-001	7.9928e-004	5.4712e-002
1.0000e-016	0	1.1102e+000	5.5511e-001	1.0000e+000	1.1022e-001	4.4489e-001
1.0000e-017	0	0	0	1.0000e+000	1.0000e+000	1.0000e+000

È fondamentale osservare che il minimo errore commesso nel calcolo della derivata numerica è ottenuto per un valore della perturbazione h dell'ordine di grandezza della radice quadrata del *macheps* nel caso delle formule "forward" e "backward". Al contrario la formula "centrata" produce la migliore stima con una perturbazione h maggiore di due ordini di grandezza.

Calcolo numerico della derivata

In analogia a quanto visto per altre tematiche dell'analisi numerica, la quantità h con cui perturbare la variabile x, nel calcolo della derivata numerica, assume la forma: $h = x \cdot xTolRel + xTolAbs$ per evitare che h sia nullo o comunque trascurabile qualora $x \to 0$.

$xTolRel$ dovrà essere scelto dell'ordine di grandezza della radice quadrata del macheps qualora si lavori con le formule "forward" o "backward".

$xTolAbs$ dovrà essere scelto dello stesso ordine di grandezza del prodotto $\aleph \cdot xTolRel$ con $\aleph$ valore non nullo rappresentativo del range di variabilità di x.

N.B.: le formule "**forward**" e "**backward**" sono di estrema importanza in quanto utilizzate nella integrazione di sistemi ODE (si pensi ad esempio ai metodi di Eulero esplicito (forward) ed implicito (backward). La formula "**centrata**" è invece sconsigliata come approssimazione della derivata prima in un sistema ODE in quanto la componente in avanti (in essa presente) richiede una previsione delle funzioni non ancora disponibile. Il metodo numerico, a cui tale formula venisse applicata, perderebbe quindi di affidabilità e robustezza (vedi anche PDE).

Calcolo numerico della derivata

N.B.: i metodi più frequentemente usati per risolvere sistemi ODE applicati a problemi dell'ingegneria chimica (quindi frequentemente *stiff*) richiedono il calcolo della **matrice Jacobiana** ovvero delle derivate parziali prime delle funzioni costituenti il sistema differenziale. In genere non è possibile effettuare tale calcolo tramite differenziazione analitica in quanto la struttura di tali funzioni è estremamente articolata. L'utente lascia quindi alla procedura di risoluzione l'onere del calcolo delle derivate. Tale calcolo viene condotto dalla routine utilizzando gli accorgimenti appena descritti.

N.B.: qualora occorra effettuare il calcolo della **derivata seconda** si applica alla derivata prima approssimata dal rapporto incrementale una nuova derivazione. Partendo ad esempio dalla formula "centrale" Si ottiene la seguente approssimazione numerica:

$$y'' \approx \frac{f(x+2h)-2f(x)+f(x-2h)}{4h^2} = \frac{f(x+k)-2f(x)+f(x-k)}{k^2}$$

Calcolo numerico della derivata

È possibile individuare alcuni motivi distinti per il calcolo numerico della derivata:

1. **Approssimazione delle derivate** nelle equazioni differenziali alle derivate ordinarie, ODE, o parziali, PDE. L'obiettivo è quello di semplificare la formulazione dell'equazione differenziale per poterla risolvere più agevolmente.
2. Calcolo della derivata di una funzione $f(x)$ di cui si conoscano soltanto i **valori numerici** in alcuni punti: $(x_i, y_i),\ i = 1,\ldots n$.
3. Calcolo dello **Jacobiano** di un sistema differenziale (metodi multivalue, impliciti per sistemi ODE stiff o sistemi DAE (algebrico-differenziali)).
4. Derivata di una funzione avente una struttura così articolata o complessa per cui la **determinazione analitica** della sua derivata sia praticamente **impossibile** o non sia proponibile, a livello di tempi di calcolo, il calcolo numerico della derivata determinata per via analitica. Il valore della derivata della funzione può essere richiesto nell'ambito della implementazione di specifici metodi numerici.

Introduzione ai sistemi ODE

Un'equazione differenziale di ordine m che sia della forma: $\mathbf{y}^{(m)} = \mathbf{f}\left(\mathbf{y}, \mathbf{y}', \ldots, \mathbf{y}^{(m-1)}, t\right)$

con le condizioni iniziali: $\mathbf{y}(t_0) = \mathbf{y}_0 \quad \mathbf{y}'(t_0) = \mathbf{y}'_0 \quad \ldots \quad \mathbf{y}^{(m-1)}(t_0) = \mathbf{y}_0^{(m-1)}$

può essere trasformata in un sistema di equazioni differenziali del primo ordine tramite l'introduzione di **nuove variabili dipendenti ausiliarie**.

Esempio: $y''' = t\,y'' + y' + (1 + t + y)^2 + \sin(t)$

$y''(0) = 1 \quad y'(0) = 2 \quad y(0) = 3$

$$\begin{cases} y_1' = y_2 \\ y_2' = y_3 \\ y_3' = t\,y_3 + y_2 + (1 + t + y)^2 + \sin(t) \end{cases}$$

con le condizioni iniziali: $y_3(0) = 1 \quad y_2(0) = 2 \quad y_1(0) = 3$

e l'introduzione delle variabili: $y_1 = y \quad y_2 = y' = y_1' \quad y_3 = y'' = y_2'$

Introduzione ai sistemi ODE

Definizione di Sistema Autonomo

Se le equazioni di un sistema differenziale **non** dipendono esplicitamente dalla variabile indipendente, t, il sistema viene detto **autonomo**:

$$\mathbf{y}' = \mathbf{f}\left(\mathbf{y}\right)$$

N.B.: è sempre possibile trasformare un sistema non autonomo in uno autonomo mediante aggiunta di una variabile dipendente e di una equazione differenziale:

$$y'_{m+1} = 1 \qquad y_{m+1}\left(t_0\right) = t_0$$

e sostituendo alla variabile indipendente, t, la nuova variabile dipendente: y_{m+1}

N.B.: alcuni algoritmi numerici risolutivi sono applicabili **solo** a sistemi autonomi.

Introduzione ai sistemi ODE

N.B.: i metodi numerici qui considerati permettono di ottenere la **soluzione approssimata** del sistema ODE in corrispondenza di alcuni valori della variabile indipendente, t_0, t_1, t_2, ...

Convenzione

Il valore **esatto** delle variabili $\mathbf{y}$ in corrispondenza di t_n viene indicato: $\mathbf{y}(t_n)$

Il valore **approssimato** delle variabili $\mathbf{y}$ in t_n viene indicato: $\mathbf{y}_n$

Analogamente: $\mathbf{y}'(t_n)$ e $\mathbf{y}'_n$ indicano rispettivamente il valore esatto ed approssimato della derivata prima.

Il simbolo, h_n, indica infine il **passo di integrazione** $t_n - t_{n-1}$.

Introduzione ai sistemi ODE

Definizione di Algoritmi One Step

Un algoritmo che per calcolare il valore di $\mathbf{y}_{n+1}$ **non utilizza** i valori di $\mathbf{y}$ o $\mathbf{y}$' valutati in corrispondenza di punti precedenti a t_n viene chiamato **One Step**.

Esempio: il metodo di **Eulero forward** è un algoritmo one step

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{f}\left(\mathbf{y}_n, t_n\right)$$

Definizione di Algoritmi Multi Step

Un algoritmo che per calcolare il valore di $\mathbf{y}_{n+1}$ **utilizza** i valori di $\mathbf{y}$ o $\mathbf{y}$' valutati in corrispondenza di punti precedenti a t_n viene chiamato **Multi Step**.

Esempio: il seguente è un esempio di algoritmo multi step

$$\mathbf{y}_{n+1} = \mathbf{y}_{n-1} + 2h\mathbf{f}\left(\mathbf{y}_n, t_n\right)$$

Introduzione ai sistemi ODE

Definizione di Metodi Espliciti

Un metodo che **non richiede** il calcolo delle funzioni $\mathbf{f}$ in corrispondenza di $\mathbf{y}_{n+1}$, t_{n+1} viene detto **esplicito**.

Esempio: il seguente metodo di **Runge-Kutta** del **secondo ordine** è esplicito

$$\mathbf{k}_1 = h\mathbf{f}\left(\mathbf{y}_n, t_n\right)$$
$$\mathbf{k}_2 = h\mathbf{f}\left(\mathbf{y}_n + \mathbf{k}_1, t_n + h\right)$$
$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{\mathbf{k}_1 + \mathbf{k}_2}{2}$$

Definizione di Metodi Impliciti

Un metodo che **richiede** il calcolo delle funzioni $\mathbf{f}$ in corrispondenza di $\mathbf{y}_{n+1}$, t_{n+1} viene detto **implicito**.

Esempio: il metodo di **Eulero backward** è implicito

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{f}\left(\mathbf{y}_{n+1}, t_{n+1}\right)$$

Introduzione ai sistemi ODE

N.B.: con riferimento ai **metodi impliciti**, se le funzioni $\mathbf{f}$ sono **non lineari** occorre risolvere un sistema di equazioni non lineari.

Criteri per la **classificazione** dei metodi risolutivi per sistemi ODE:

1. One step oppure multi step;
2. In base al criterio utilizzato per ricavare la formula: espansione in serie di Taylor o approssimazione polinomiale;
3. Esplicito oppure implicito.

Accuratezza degli algoritmi numerici

Un algoritmo numerico per la risoluzione di sistemi ODE è caratterizzato da: un **errore locale** ed un **errore di propagazione**.

L'**errore locale** viene calcolato nell'ipotesi di **assenza di errori numerici nei calcoli e nei dati**. Esso è legato soltanto al modello approssimato usato.

N.B.: l'errore locale dipende da una potenza nota del passo di integrazione.

Esempio: l'errore locale del metodo di Eulero forward è $h_n^2 \frac{y''(\xi_n)}{2}$ e corrisponde a $O(h^2)$

Definizione di Ordine di un Algoritmo

L'ordine di un algoritmo è uguale all'ordine del suo errore locale meno uno.

Un algoritmo di ordine p ha un errore locale di ordine $O(h^{p+1})$.

Accuratezza degli algoritmi numerici

Da un punto di vista teorico un algoritmo di ordine più elevato ha un errore locale minore e permette un passo di integrazione più grande.

Da un punto di vista pratico occorre scegliere un ordine di compromesso.

L'ordine non deve essere **troppo basso** per evitare errori locali troppo grandi e quindi un passo di integrazione troppo piccolo.

L'ordine non deve essere **troppo elevato** per evitare che l'algoritmo presenti problemi numerici.

In molti casi l'ordine ottimale è $4-5$.

Definizione di Convergenza di un Algoritmo

Un algoritmo è convergente se al tendere del passo di integrazione, h, a zero la soluzione diventa sempre più accurata nell'ipotesi di errori di arrotondamento nulli.

N.B.: tutti i metodi di interesse pratico sono convergenti.

Condizionamento del problema

Occorre distinguere tra **condizionamento di un problema** e **stabilità di un algoritmo numerico**.

Un'equazione differenziale è **ben condizionata** se una piccola perturbazione della funzione f e/o della condizione iniziale y_0

$$y' = f(y,t) + \delta(t) \qquad y(t_0) = y_0 + \varepsilon$$

generano una soluzione (ottenuta senza errori numerici) che si discosta di poco da quella teorica del problema iniziale.

Condizionamento del problema

Esempio: $y' = 9y - 10e^{-t} \qquad y(0) = 1$ ⇒ $y = e^{-t} + c\,e^{9t}$

soluzione generale

⇓

$y = e^{-t}$

soluzione particolare

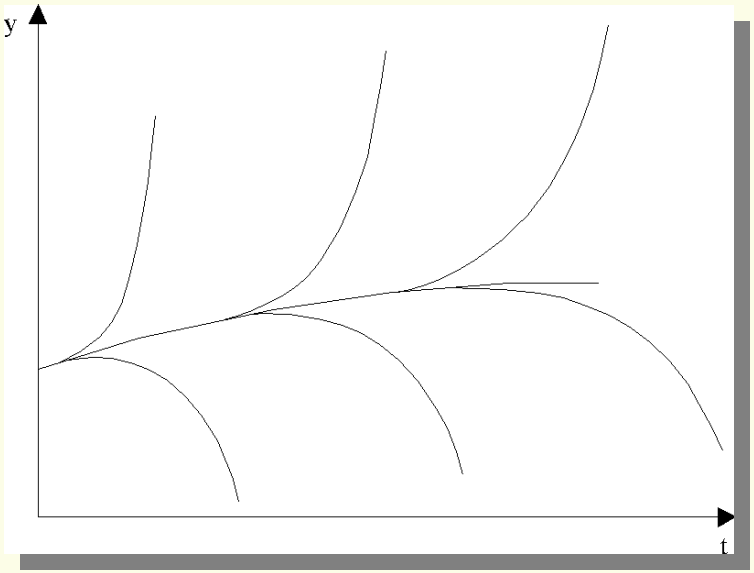
---

se però: $y(0) = 1.0001$ ⇒ $y = e^{-t} + 0.0001e^{9t}$

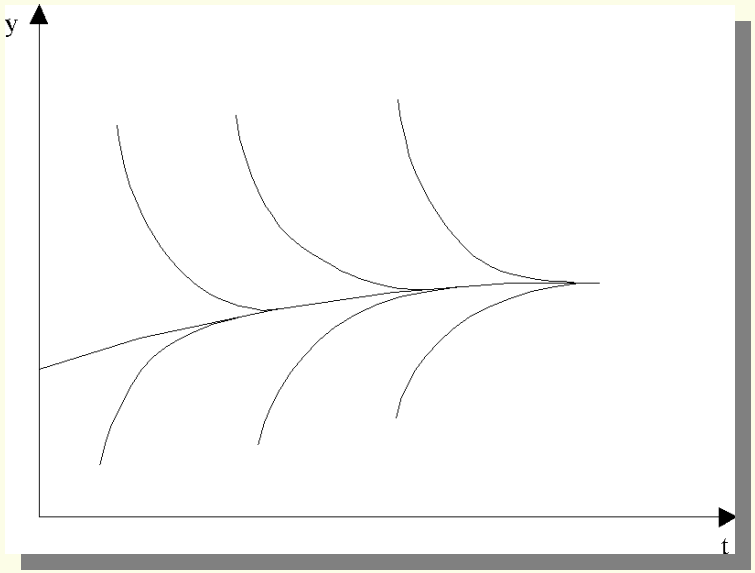
soluzione particolare

N.B.: per $t = 2$ le due soluzioni particolari valgono: 0.18861 e 331.23 quindi il **problema è malcondizionato**.

Condizionamento del problema



Problema mal condizionato



Problema ben condizionato

La soluzione di un'equazione differenziale consiste in una particolare curva univocamente determinata dall'equazione differenziale e dalle condizioni iniziali. Al variare delle condizioni iniziali si ottengono differenti curve.

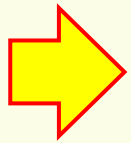
Condizionamento del problema

N. B.: il condizionamento di un problema dipende dalla direzione di integrazione. Una equazione ben condizionata in una direzione è mal condizionata nella direzione opposta.

N.B.: nei casi pratici può succedere che un'equazione sia malcondizionata in una porzione dell'intervallo di integrazione e ben condizionata in un'altra.

Esempio:

$$y' = 9y - 10e^{-t} \qquad y(0) = 1$$
$$y' = e^{-t} \qquad y(0) = 1$$
$$y' = -y \qquad y(0) = 1$$



$y(t) = e^{-t}$ unica soluzione

però se si perturba la condizione iniziale di poco: $y(0) = 1.0001$

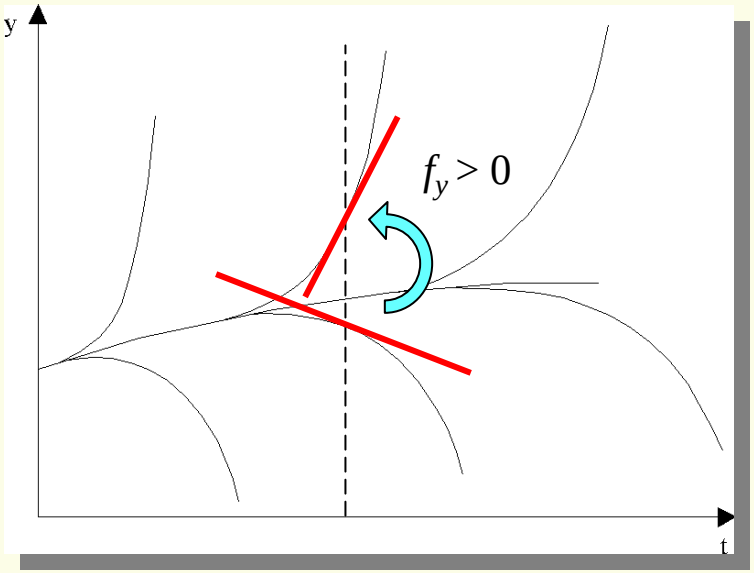
si ottiene:

$y(t) = e^{-t} + 0.0001e^{9t}$ la soluzione diverge da e^{-t}

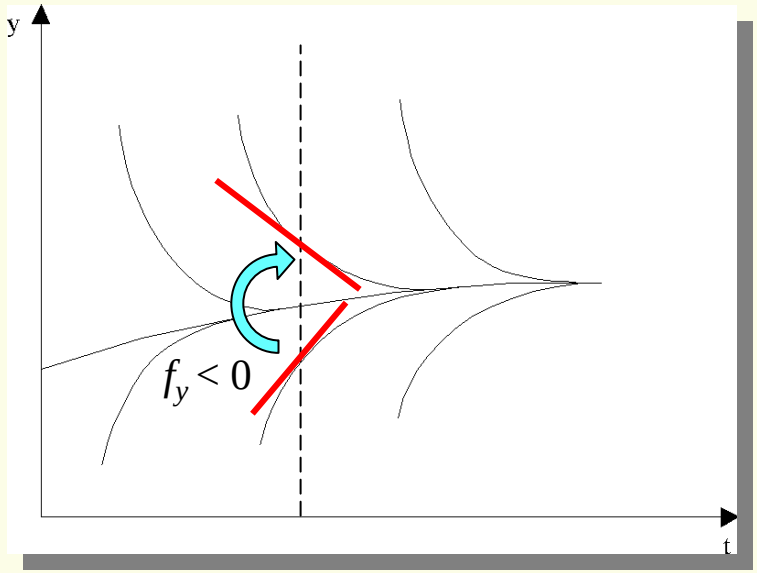
$y(t) = e^{-t} + 0.0001$ la soluzione resta a distanza costante da e^{-t}

$y(t) = e^{-t} + 0.0001e^{-t}$ la soluzione converge su e^{-t}

Condizionamento del problema



Problema mal condizionato



Problema ben condizionato

Affinché l'equazione differenziale sia **ben condizionata**, la derivata f_y deve essere **negativa**: solo in questo caso le curve della famiglia tendono a ricadere sulla soluzione cercata.

Se f_y è **positiva** l'equazione differenziale è **mal condizionata**.

Condizionamento del problema

Il condizionamento di un **sistema** di equazioni differenziali viene studiato analizzando il valore degli autovalori $\lambda_1, \lambda_2, \ldots, \lambda_N$ della matrice Jacobiana.

$$\mathbf{J} = \left\{ \frac{\partial f_i}{\partial y_j} \right\}$$

Se la **parte reale** di un **autovalore** è **grande** e **positiva** il sistema è **malcondizionato**.

Se tutti gli autovalori hanno parte reale negativa il sistema è **ben condizionato**.

Stabilità degli algoritmi numerici

Come detto in precedenza, il concetto di **stabilità** viene applicato agli **algoritmi** numerici per la risoluzione di sistemi ODE.

Per semplificare il ragionamento, si considera l'algoritmo di **Eulero esplicito** applicato ad una sola equazione differenziale:

$$y_{n+1} = y_n + h_n \, f\left(y_n, t_n\right)$$

se questa relazione viene sottratta all'espansione in serie di

Taylor di $\mathbf{y}'\left(t_{n+1}\right)$ si ottiene:

$$\underbrace{y\left(t_{n+1}\right) - y_{n+1}}_{\text{Errore globale}} = \underbrace{y\left(t_n\right) - y_n + h_n\left[\, f\left(y\left(t_n\right), t_n\right) - f\left(y_n, t_n\right)\right]}_{\text{Errore di propagazione}} + \underbrace{h_n^2 y''\left(\xi_n\right)/2}_{\text{Errore locale}}$$

Leonhard Euler
(1707-1783)

Stabilità degli algoritmi numerici

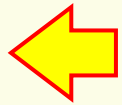
L'errore globale, in assenza di errori numerici di arrotondamento, è la somma di due termini:

Errore locale: è dovuto al modello approssimato usato. Lo si stima supponendo di eseguire i calcoli senza errori numerici e di conoscere esattamente i dati necessari (si suppone cioè di conoscere il valore corretto di y nel punto t_n, $y(t_n)$).

Errore di propagazione: in realtà nel punto t_n non si conosce il valore esatto $y(t_n)$ bensì una sua approssimazione: y_n e ciò porta ad una seconda fonte di errore che è indipendente dall'errore locale.

Per il teorema della media: $f\left(y\left(t_n\right),t_n\right)-f\left(y_n,t_n\right)=f_y\left(z\right)\left(y\left(t_n\right)-y_n\right) \qquad t_n<z<t_{n+1}$

Quindi: $E.G.\left(t_{n+1}\right)=\left(1+h_n f_y\right)E.G.\left(t_n\right)+E.L.$



Avendo indicato con: $E.G.$ (errore globale); $E.L.$ (errore locale).

Stabilità degli algoritmi numerici

Per quanto riguarda l'algoritmo di Eulero esplicito è quindi possibile osservare che esso è **instabile** se:

$$\left|1+h_n f_y\right| > 1$$

fattore di amplificazione κ

dato che l'errore globale tende ad aumentare da una iterazione all'altra.

Affinché l'algoritmo sia **stabile** occorre scegliere un passo di integrazione h_n tale che il fattore di amplificazione, κ, sia inferiore all'unità lungo tutto il corso dell'integrazione.

N.B.: se un problema è malcondizionato non ha senso interrogarsi sulla stabilità dello specifico algoritmo adottato per risolverlo. Al contrario quando un problema è ben condizionato allora è possibile chiedersi quali siano le condizioni per le quali l'algoritmo risulta stabile.

Stabilità degli algoritmi numerici

Esempio: risolvere con il metodo di Eulero forward l'equazione differenziale:

$$y' = -1000y + 1000 \qquad y(0) = 10$$

l'equazione è molto ben condizionata in quanto risulta: $f_y = -1000$ ed ha come soluzione analitica: $y = 1 + 9e^{-1000t}$

Se però viene utilizzato un passo di integrazione superiore a: $h = 0.002$ il metodo di Eulero esplicito diventa instabile.

N.B.: già per $t > 1$ l'errore locale diventa nullo (la soluzione diviene uguale a 1) e quindi si potrebbe pensare che un passo di integrazione comunque grande potrebbe essere adatto. Così non succede, a causa del fattore di amplificazione che richiede un valore del passo $h < 0.002$.

Stabilità degli algoritmi numerici

Definizione di Stabilità di un Algoritmo

Se il fattore di amplificazione di un algoritmo è minore di uno, l'algoritmo è **stabile**. In caso contrario è **instabile**.

La stabilità di un algoritmo applicato ad un **sistema ODE** richiede la valutazione della matrice Jacobiana. Nel caso del metodo di **Eulero esplicito** il fattore di amplificazione diviene:

$$\left\| \mathbf{I} + h_n \mathbf{J}_n \right\|$$

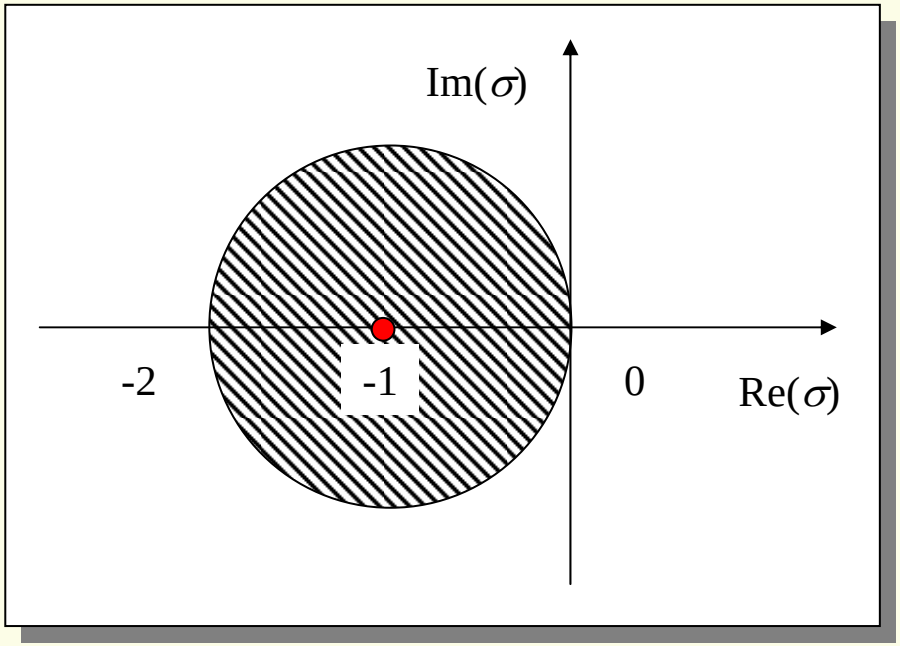
indicando con λ_n il **massimo autovalore** della matrice Jacobiana, l'algoritmo di Eulero esplicito è stabile se:

$$\left| 1 + h_n \lambda_n \right| < 1$$

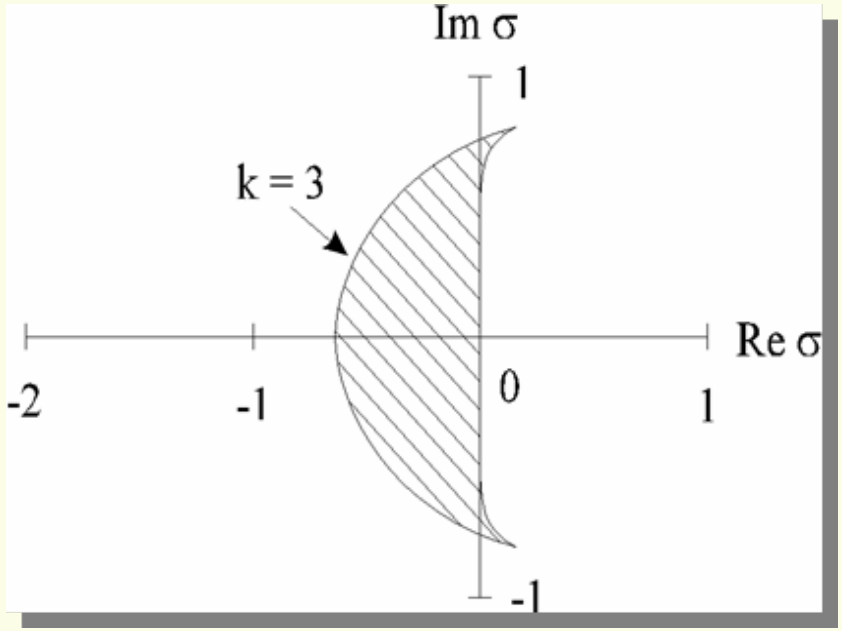
N.B.: λ_n in genere è un numero complesso e può variare nel corso dell'integrazione.

Stabilità degli algoritmi numerici

Definendo $\sigma = h\lambda$, è possibile determinare per ogni algoritmo il dominio nel campo complesso che separa i punti in cui il fattore di amplificazione è minore di 1 (stabilità) da quelli in cui è maggiore di 1 (instabilità).



Regione di stabilità del metodo di Eulero forward



Regione di stabilità del metodo di Adams-Moulton implicito del III ordine

Stabilità degli algoritmi numerici

Osservazioni

- A parità di ordine risulta più stabile un metodo **implicito** rispetto ad uno **esplicito**.
- **La stabilità diminuisce al crescere dell'ordine di un algoritmo**.

Definizione di Algoritmi A-stabili

Dato un problema ben condizionato, un algoritmo viene detto **A-stabile** se il fattore di amplificazione risulta minore di uno per ogni valore positivo di h.

Un algoritmo A-stabile ha come dominio di stabilità tutto il semipiano di sinistra.

Esempio: l'algoritmo del trapezio e quello di Eulero backward sono A-stabili.

Stabilità degli algoritmi numerici

Definizione di Algoritmi fortemente A-stabili

Un algoritmo è **fortemente A-stabile** se il fattore di amplificazione tende a zero quando $h\lambda$ tende a meno infinito.

Esempio: l'algoritmo di Eulero backward è fortemente A-stabile mentre quello del trapezio non lo è (il suo fattore di amplificazione tende a -1 per $h\lambda \to -\infty$)

Gli algoritmi **One Step** sono **sempre stabili** per h tendente a zero.

Gli algoritmi **Multi Step** possono essere **instabili** per h tendente a zero a causa di soluzioni parassite che possono prevalere sulla soluzione vera del problema.

Contrariamente a quanto si possa pensare esistono dei casi in cui alcuni algoritmi risultano più instabili con la diminuzione del passo di integrazione.

Stabilità degli algoritmi numerici

Teorema di Dahlquist (1963)

Dahlquist ha dimostrato che **non** esistono algoritmi Multi Step di ordine maggiore di due che siano A-stabili e quindi tantomeno fortemente A-stabili.

N.B.: finora non si è tenuto conto dell'effetto dell'**errore di arrotondamento** sull'errore globale. Nella maggior parte dei casi l'errore di arrotondamento è trascurabile rispetto a quello locale.

In realtà l'errore di arrotondamento può diventare importante qualora il numero complessivo di passi di integrazione sia elevato. Ad esempio quando l'errore locale è elevato occorre effettuare dei passi di integrazione piccoli e ciò fa aumentare il numero di passi di integrazione. Si ha lo stesso problema quando viene richiesto di conoscere l'andamento delle variabili dipendenti per un elevato numero di punti intermedi (tipicamente problemi di grafica). Esistono comunque degli algoritmi (**Multivalue**) che aggirano questo problema.

Comportamento dei sistemi ODE

Il comportamento locale della soluzione di un sistema differenziale in un opportuno intorno del punto $(t_p, \mathbf{y}_p)$ può essere analizzato espandendo in serie di Taylor rispetto a t e $\mathbf{y}$ la funzione $\mathbf{f}(t, \mathbf{y})$ del sistema $\mathbf{y}' = \mathbf{f}(t, \mathbf{y})$:

$$\mathbf{f}(t,\mathbf{y}) = \mathbf{f}(t_p,\mathbf{y}_p) + \alpha(t - t_p) + \mathbf{J}(\mathbf{y} - \mathbf{y}_p) + \dots$$

$$\alpha = \frac{\partial \mathbf{f}}{\partial t}(t_p,\mathbf{y}_p) \qquad \mathbf{J} = \frac{\partial \mathbf{f}}{\partial \mathbf{y}}(t_p,\mathbf{y}_p)$$

Si noti che se il sistema differenziale è autonomo allora $\alpha = 0$.

$\mathbf{J}$ è la matrice Jacobiana contenente le derivate parziali del sistema $\mathbf{f}(t, \mathbf{y})$ rispetto alle variabili $\mathbf{y}$ e quindi ha dimensioni $n \times n$.

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial y_1} & \frac{\partial f_1}{\partial y_2} & \cdots & \frac{\partial f_1}{\partial y_n} \\ \frac{\partial f_2}{\partial y_1} & \frac{\partial f_2}{\partial y_2} & \cdots & \frac{\partial f_2}{\partial y_n} \\ \vdots & & \ddots & \vdots \\ \frac{\partial f_n}{\partial y_1} & \frac{\partial f_n}{\partial y_2} & \cdots & \frac{\partial f_n}{\partial y_n} \end{bmatrix}$$

Comportamento dei sistemi ODE

L'influenza di $\mathbf{J}$ sul comportamento locale è determinata dalla soluzione del sistema lineare di equazioni differenziali ordinarie: $\mathbf{y}' = \mathbf{J}\,\mathbf{y}$

ottenuto dal troncamento al primo termine dello sviluppo in serie di Taylor e ricordando che un generico sistema ODE può essere sempre ricondotto ad un sistema autonomo tramite introduzione di una variabile dipendente ausiliaria.

Siano: $\lambda_k = \mu_k + i\nu_k$ gli autovalori di $\mathbf{J}$ e $\mathbf{L} = diag(\lambda_k)$ la matrice diagonale degli autovalori. Se esiste un insieme di autovettori, $\mathbf{V}$, linearmente indipendenti, allora: $\mathbf{J} = \mathbf{V}\,\mathbf{L}\,\mathbf{V}^{-1}$

La trasformazione lineare: $\mathbf{V}\,\mathbf{x} = \mathbf{y}$ produce un insieme locale di equazioni differenziali disaccoppiate nelle singole componenti: $\dot{x}_k = \lambda_k x_k$ le cui soluzioni sono:

$$x_k(t) = e^{\lambda_k (t - t_p)} x(t_p)$$

Comportamento dei sistemi ODE

La singola componente $x_k(t)$:

1. cresce con t se μ_k è positivo;
2. decresce con t se μ_k è negativo;
3. oscilla se ν_k è non nullo;
4. Se μ_k è nullo e $\nu_k \neq 0$ allora la componente $x_k(t)$ è un oscillatore puro.

N.B.: dato che un sistema differenziale è costituito da tante componenti, ognuna di queste può avere un comportamento peculiare (crescente, decrescente e/o oscillante).

Sistemi stiff

www.hyperdictionary.com

Stiff = incapable of or resistant to bending; "a rigid strip of metal"; "a table made of rigid plastic"; "a palace guardsman stiff as a poker“ (poker = a metal bar or rod used in stirring a fire of coals)

Evidenza

In molte applicazioni pratiche l’integrazione di sistemi ben condizionati, effettuata con metodi tradizionali, richiede un passo di integrazione straordinariamente piccolo per garantire la stabilità dell’algoritmo utilizzato.

Sistemi stiff

Esempio: è dato il seguente sistema differenziale stiff

$$\begin{cases} y_1' = \dfrac{\lambda_1 + \lambda_2}{2} y_1 + \dfrac{\lambda_1 - \lambda_2}{2} y_2 \\ y_2' = \dfrac{\lambda_1 - \lambda_2}{2} y_1 + \dfrac{\lambda_1 + \lambda_2}{2} y_2 \end{cases}$$

che ha soluzione generale:

$$y_1(t) = C_1 e^{\lambda_1 t} + C_2 e^{\lambda_2 t}$$
$$y_2(t) = C_1 e^{\lambda_1 t} - C_2 e^{\lambda_2 t}$$

utilizzando il metodo di Eulero esplicito affinché l'algoritmo sia stabile occorre che:

$$\left|1 + h\lambda_1\right| < 1 \quad \text{e} \quad \left|1 + h\lambda_2\right| < 1$$

se $\lambda_1 = -1$ e $\lambda_2 = -10000$ il sistema è ben condizionato. Il termine $C_2 \exp(\lambda_2 t)$ ha importanza soltanto per t molto piccolo. Ciononostante il passo di integrazione viene condizionato dall'autovalore λ_2 anche quando t è grande in quanto deve essere soddisfatta la relazione: $\left|1 + h\lambda_2\right| < 1$

L'utilizzo del metodo di Eulero forward richiede paradossalmente l'adozione di un passo piccolissimo per un termine di nessuna importanza.

Lo stesso accade adottando un metodo di Runge-Kutta del IV ordine.

Sistemi stiff

Definizione di Sistema Stiff

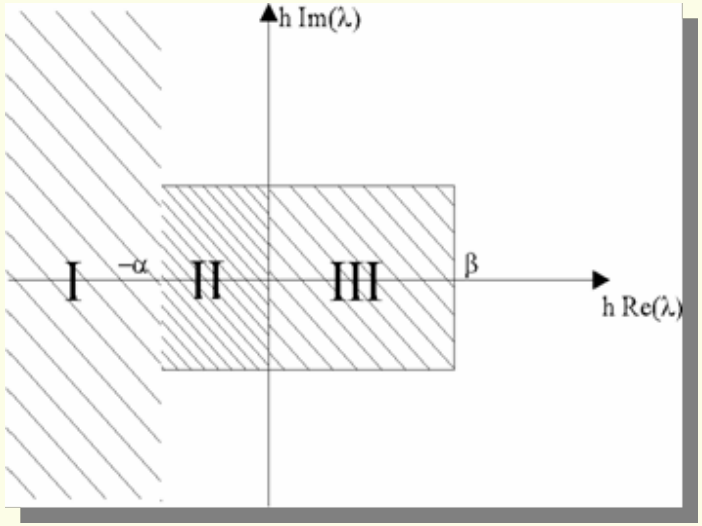
Un sistema differenziale ben condizionato risulta **stiff** se il modulo del rapporto tra il massimo ed il minimo autovalore della matrice Jacobiana, $\mathbf{J}$, è grande.

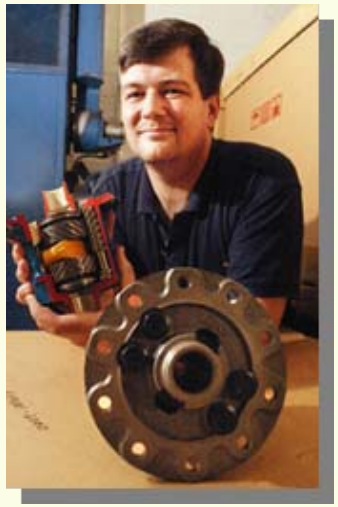
Nella pratica un sistema è stiff qualora nel processo fisico descritto siano presenti equazioni differenziali con diverse scale di tempo (ad es.: cinetica chimica con specie molecolari e radicaliche; sistemi dinamici con costanti di tempo significativamente differenti; ...).

Esempio: l'equazione differenziale $y' = -y \qquad y(0) = 1$ non presenta problemi se l'intervallo di integrazione è relativamente piccolo. Se però il tempo finale di integrazione vale ad esempio 10^6, gli algoritmi che non siano fortemente A-stabili o che non siano adatti a risolvere problemi stiff dovrebbero utilizzare passi di integrazione molto piccoli anche se hanno un errore locale praticamente nullo.

Sistemi stiff

Secondo l'analisi di **Gear** (1971) un algoritmo è adatto a risolvere sistemi stiff se gode delle seguenti proprietà:

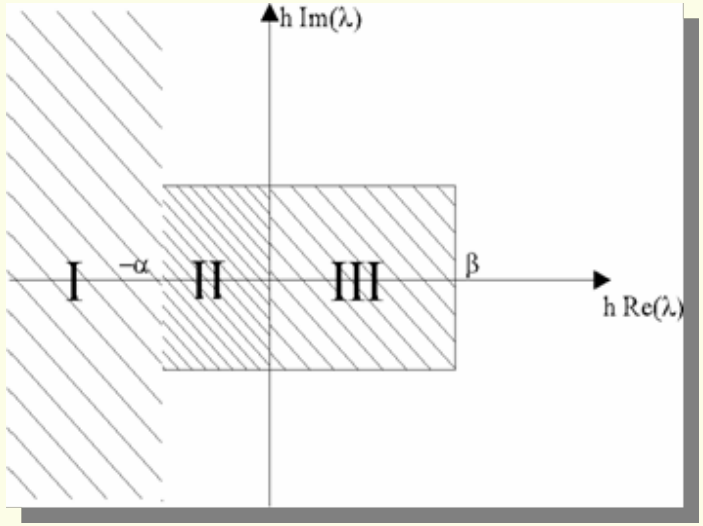




??? C. W. GEAR ???
Wanted dead or alive

Regione I: in questa regione risulta $h\mathrm{Re}(\lambda) < -\alpha$ e l'algoritmo deve essere **stabile**. Se $\alpha = 0$ allora il metodo è A-stabile. Gear, rilassando l'ipotesi di A-stabilità, ha permesso di determinare algoritmi Multi Step di ordine superiore a due (per valori piccoli, negativi ma non nulli di α).

Sistemi stiff



Regione II: in questa regione risulta $-\alpha < h\mathrm{Re}(\lambda) < 0$ e l'algoritmo deve risultare **stabile ed accurato**.

Regione III: in questa regione risulta $0 < h\mathrm{Re}(\lambda) < \beta$ e il sistema è moderatamente mal condizionato. L'algoritmo deve risultare **accurato**.

Metodi di Runge-Kutta espliciti

I metodi di Runge-Kutta si basano su formule ricorrenti con coefficienti numerici scelti in modo da soddisfare al meglio un'espansione in serie di Taylor della soluzione $\mathbf{y}(t)$.

Carl Runge
(1856-1927)

Esempio: il metodo di Runge-Kutta del **IV ordine, *classico***, è il più noto e per anni è stato anche il più utilizzato

$$\mathbf{k}_1 = h\mathbf{f}\left(\mathbf{y}_n, t_n\right)$$
$$\mathbf{k}_2 = h\mathbf{f}\left(\mathbf{y}_n + 0.5\mathbf{k}_1, t_n + 0.5h\right)$$
$$\mathbf{k}_3 = h\mathbf{f}\left(\mathbf{y}_n + 0.5\mathbf{k}_2, t_n + 0.5h\right)$$
$$\mathbf{k}_4 = h\mathbf{f}\left(\mathbf{y}_n + \mathbf{k}_3, t_n + h\right)$$
$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4}{6}$$

Wilhelm Kutta
(1867-1944)

N.B.: dato che fino all'ordine quattro sono richieste tante valutazioni del sistema ODE quanto è l'ordine stesso mentre per ordini maggiori occorre effettuare $p+1$ valutazioni, il metodo RK IV rappresenta un ottimo compromesso fra precisione, sforzo computazionale e stabilità.

Metodi di Runge-Kutta espliciti

PRO

1. Sono efficienti anche quando la soluzione non viene ben approssimata con polinomi
2. Sono di solito poco sensibili a eventuali discontinuità delle funzioni del sistema
3. Richiedono poca memoria
4. È facile cambiare il passo di integrazione in un momento qualsiasi
5. Il tempo di calcolo **dell'algoritmo** non è elevato. Quindi per sistemi ODE che richiedano poche risorse di calcolo, il tempo complessivo di integrazione (CPU time) risulta basso

Metodi di Runge-Kutta espliciti

CONTRO

1. Il numero di calcoli del sistema ODE è generalmente maggiore rispetto a quello di altre alternative (metodi Multi Step e Multi Value)
2. Non sono adatti per sistemi stiff
3. L'errore locale non è facile da calcolare
4. Non è possibile risolvere sistemi in cui le derivate $\mathbf{y}'$ siano presenti in forma implicita
5. Non permettono di risolvere sistemi algebrico-differenziali
6. Non si prestano a risolvere problemi in cui siano richiesti valori delle variabili dipendenti, $\mathbf{y}$, molto ravvicinati, ossia con valori di h decisamente inferiori a quelli richiesti dalla precisione dell'algoritmo

Metodi RK: gestione del passo di integrazione

Con riferimento all'**errore locale**, E, e alla precisione richiesta, ε, se risulta che:

$$E = C\,h^{p+1} > \varepsilon$$

occorre ridurre il passo di integrazione, h, e ripetere l'integrazione.

Il nuovo passo di integrazione dovrà soddisfare la condizione:

$$C\,h_{new}^{p+1} \le \varepsilon$$

ovvero: $h_{new}^{p+1} = h^{p+1}\sqrt{\dfrac{\varepsilon}{E}}$

In genere si preferisce la seguente espressione: $h_{new}^{p+1} = \alpha\, h^{p+1}\sqrt{\dfrac{\varepsilon_A + \varepsilon_R |y_n|}{E}}$

N.B.: è buona norma prevedere un controllo affinché il passo di integrazione non venga ridotto sotto valori così piccoli che eviterebbero l'effettiva fattibilità dell'integrazione. In tali casi occorre bloccare l'integrazione. Parimenti occorre controllare che il passo di integrazione non superi opportuni valori limite.

Metodi RK: stima dell'errore locale

Algoritmo di Runge-Kutta-Merson

Merson ha proposto un algoritmo di quarto ordine che utilizzando cinque calcoli di funzione permette di stimare l'errore locale.

Algoritmi embedded

Esistono dei gradi di libertà nella individuazione dei metodi RK che permettono di identificare degli algoritmi di grado crescente che utilizzano porzioni comuni di formule. In questo modo è possibile utilizzare l'algoritmo di ordine più elevato per stimare l'errore locale di quello di ordine inferiore.

Il più utilizzato tra gli algoritmi embedded è quello di **Runge-Kutta-Fehlberg** del quarto ordine controllato da uno di quinto ordine. Questa versione del metodo RK risulta di solito leggermente migliore delle altre.

Metodi RK impliciti e semiimpliciti

Un metodo RK costituito da r termini può essere rappresentato tramite due vettori $\mathbf{w}$ e $\mathbf{c}$ e la matrice $\mathbf{A}$:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + w_1\mathbf{k}_1 + w_2\mathbf{k}_2 + \ldots + w_r\mathbf{k}_r$$

$$\mathbf{k}_i = h\mathbf{f}\left(\mathbf{y}_n + \sum_{j=1}^{r} a_{ij}\mathbf{k}_j, t_n + c_i h\right) \qquad i = 1, \ldots, r$$

Considerando la matrice $\mathbf{A}$ come la somma di tre sottomatrici: $\mathbf{A}_\mathrm{L}$, $\mathbf{A}_\mathrm{D}$, $\mathbf{A}_\mathrm{R}$ è possibile effettuare la seguente distinzione:

- se le matrici $\mathbf{A}_\mathrm{D}$ e $\mathbf{A}_\mathrm{R}$ sono nulle allora il metodo è **esplicito**
- se la matrice $\mathbf{A}_\mathrm{R}$ è nulla allora il metodo è **semiimplicito**
- se anche la matrice $\mathbf{A}_\mathrm{R}$ è non-nulla allora il metodo è **implicito**

Metodi RK impliciti e semiimpliciti

Nei metodi RK **espliciti** non vi sono problemi a calcolare i termini $\mathbf{k}_i$ in quanto le formule sono appunto esplicite. Al contempo nessun metodo esplicito è A-stabile.

Nei metodi **semiimpliciti** occorre risolvere un sistema lineare. Alcuni sono anche fortemente A-stabili.

Nei metodi RK **impliciti** occorre risolvere un sistema non lineare. Molti metodi impliciti sono A-stabili ed alcuni anche fortemente A-stabili.

Nei metodi **semiimpliciti** la matrice Jacobiana interviene **direttamente** nella risoluzione del sistema lineare. È tale matrice ad essere fattorizzata per determinare la soluzione. Occorre quindi calcolare con estrema precisione lo Jacobiano del sistema altrimenti la soluzione del sistema lineare non è corretta. Inoltre lo Jacobiano del sistema ODE deve essere calcolato ad ogni passo in quanto deve essere fattorizzato per risolvere il sistema lineare. Ciò comporta un notevole esborso di risorse di calcolo (CPU time) che conduce alla pratica non applicabilità di tali metodi se confrontati con i più efficienti Multi Step e Multi Value.

Al contrario i metodi **impliciti** utilizzano lo Jacobiano indirettamente, come strumento necessario alla risoluzione del sistema non lineare (ad esempio metodo di Newton) e quindi possono *accontentarsi* anche di una approssimazione di tale matrice.

Metodi Multi Step

I metodi **Multi Step** utilizzano i valori delle variabili dipendenti, $\mathbf{y}$, accumulati negli intervalli precedenti con **passo di integrazione costante**.

La formula generica di un metodo multi step è:

$$\mathbf{y}_{n+1} = a_0\mathbf{y}_n + a_1\mathbf{y}_{n-1} + \ldots + a_k\mathbf{y}_{n-k} + \\ + h\left[b_{-1}\mathbf{f}\left(\mathbf{y}_{n+1}, t_{n+1}\right) + b_0\mathbf{f}\left(\mathbf{y}_n, t_n\right) + b_1\mathbf{f}\left(\mathbf{y}_{n-1}, t_{n-1}\right) + \ldots + b_k\mathbf{f}\left(\mathbf{y}_{n-k}, t_{n-k}\right)\right]$$

- se $b_{-1} = 0$ il metodo è **esplicito**
- se $b_{-1} \neq 0$ il metodo è **implicito** (quindi occorre risolvere ad ogni passo un sistema di equazioni non lineari)

PRO

- Forniscono senza difficoltà una stima dell'errore locale
- Richiedono un minor numero di calcoli di funzione per passo di integrazione

Metodi Multi Step

Un metodo **Runge-Kutta** di ordine p richiede ad ogni passo di integrazione almeno p calcoli del sistema ODE. Un metodo **Multi Step** esplicito di ordine p richiede un solo calcolo di sistema.

Gli algoritmi di **Adams-Bashforth** sono **espliciti**

Gli algoritmi di **Adams-Moulton** sono **impliciti**

CONTRO

- Tutti i metodi Multi Step **espliciti** sono **instabili** quindi occorre utilizzare quelli impliciti.
- Gli algoritmi Multi Step **non** sono **autosufficienti**. Occorre cioè disporre di altri algoritmi per **avviare** l'integrazione e calcolare i punti coinvolti nella formula.
- Le formule sono adatte ad un passo costante. Ogniqualvolta si modifica il passo di integrazione occorre reinizializzare l'integrazione per poter disporre nuovamente dei punti necessari alla formula stessa (si usa ad esempio un metodo RK).

Metodi Multi Value

Proprio a causa dei problemi presentati dai metodi Multi Step relativamente al cambiamento del passo di integrazione si sono sviluppati i metodi **Multi Value**. Per questo motivo i metodi **Multi Step** sono **obsoleti**.

I metodi **MV** sono una versione più moderna degli stessi metodi **MS**.

Il punto cardine del cambiamento di approccio tra MV e MS è la modalità di rappresentazione di un polinomio.

- Negli **algoritmi MS** vengono memorizzati i punti di supporto del polinomio:

$$(y_n, t_n), (y_{n-1}, t_{n-1}), (y_{n-2}, t_{n-2}), \ldots$$

- Viceversa negli **algoritmi MV** lo **stesso** polinomio viene memorizzato come valore dell'ordinata e delle sue derivate in **un unico punto** t_n:

$$(y_n, t_n), (y'_n, t_n), (y''_n, t_n), \ldots$$

Metodi Multi Value

Si supponga per il momento di avere già innescato il metodo MV e di essere al punto di integrazione t_n. La storia precedente è quindi nota e memorizzata in un opportuno vettore $\mathbf{z}$ la cui struttura è stata proposta da **Nordsieck**.

Supponendo di riferirci alla versione **MV** del metodo di **Adams-Moulton del quarto ordine**, il **vettore di Nordsieck** è così composto:

$$\mathbf{z}_0 = \mathbf{y}_n$$
$$\mathbf{z}_1 = h\,\mathbf{y}'_n$$
$$\mathbf{z}_2 = \frac{h^2\mathbf{y}''_n}{2}$$
$$\mathbf{z}_3 = \frac{h^3\mathbf{y}'''_n}{6}$$
$$\mathbf{z}_4 = \frac{h^4\mathbf{y}^{(4)}}{24}$$

Si noti che quando si risolve un sistema differenziale ogni componente del vettore $\mathbf{z}$ è a sua volta un vettore, sicché $\mathbf{z}$ è un vettore di vettori (ovvero una matrice).

Metodi Multi Value

Se conoscessimo le componenti esatte, prive di errori, del vettore $\mathbf{z}$ nel punto t_n si potrebbe scrivere il seguente **sviluppo in serie di Taylor** per ogni componente:

$$\mathbf{y}(t_{n+1}) = \mathbf{y}(t_n) + h\mathbf{y}'(t_n) + \frac{h^2\mathbf{y}''(t_n)}{2} + \frac{h^3\mathbf{y}'''(t_n)}{6} + \frac{h^4\mathbf{y}^{(4)}(t_n)}{24} + \frac{h^5\mathbf{y}^{(5)}(t_n)}{120} + \dots$$

$$h\mathbf{y}'(t_{n+1}) = h\mathbf{y}'(t_n) + 2\frac{h^2\mathbf{y}''(t_n)}{2} + 3\frac{h^3\mathbf{y}'''(t_n)}{6} + 4\frac{h^4\mathbf{y}^{(4)}(t_n)}{24} + 5\frac{h^5\mathbf{y}^{(5)}(t_n)}{120} + \dots$$

$$\frac{h^2\mathbf{y}''(t_{n+1})}{2} = \frac{h^2\mathbf{y}''(t_n)}{2} + 3\frac{h^3\mathbf{y}'''(t_n)}{6} + 6\frac{h^4\mathbf{y}^{(4)}(t_n)}{24} + 10\frac{h^5\mathbf{y}^{(5)}(t_n)}{120} + \dots$$

$$\frac{h^3\mathbf{y}'''(t_{n+1})}{6} = \frac{h^3\mathbf{y}'''(t_n)}{6} + 4\frac{h^4\mathbf{y}^{(4)}(t_n)}{24} + 10\frac{h^5\mathbf{y}^{(5)}(t_n)}{120} + \dots$$

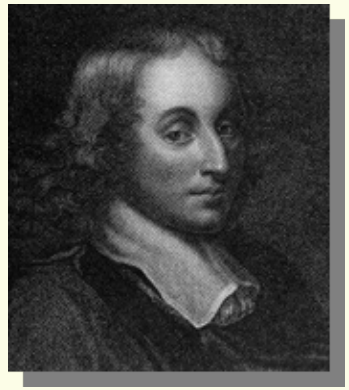
$$\frac{h^4\mathbf{y}^{(4)}(t_{n+1})}{24} = \frac{h^4\mathbf{y}^{(4)}(t_n)}{24} + 5\frac{h^5\mathbf{y}^{(5)}(t_n)}{120} + \dots$$

Brook Taylor
(1685-1731)

Metodi Multi Value

In realtà il vettore $\mathbf{z}$ permette di calcolare un vettore $\mathbf{v}$ che è una approssimazione di tale sviluppo in serie di Taylor:

$$\mathbf{v}_0 = \mathbf{z}_0 + \mathbf{z}_1 + \mathbf{z}_2 + \mathbf{z}_3 + \mathbf{z}_4$$
$$\mathbf{v}_1 = \mathbf{z}_1 + 2\mathbf{z}_2 + 3\mathbf{z}_3 + 4\mathbf{z}_4$$
$$\mathbf{v}_2 = \mathbf{z}_2 + 3\mathbf{z}_3 + 6\mathbf{z}_4$$
$$\mathbf{v}_3 = \mathbf{z}_3 + 4\mathbf{z}_4$$
$$\mathbf{v}_4 = \mathbf{z}_4$$



Blaise Pascal
(1623-1662)

o in forma matriciale: $\mathbf{v} = \mathbf{D}\mathbf{z}$ con $\mathbf{D}$ la matrice triangolare di Pascal:

$$d_{i,j} = \frac{j!}{(j-i)!\, i!} \qquad j \geq i \geq 0$$

N.B.: il vettore $\mathbf{v}$ viene ottenuto tramite semplici somme vettoriali. $\mathbf{v}$ può essere considerato come una previsione, costruita con il vettore $\mathbf{z}$ noto nel punto t_n, dello stesso vettore $\mathbf{z}$ nel nuovo punto t_{n+1}.

Metodi Multi Value

La previsione, $\mathbf{v}$, viene corretta utilizzando un opportuno vettore $\mathbf{b}$ in modo da ottenere l'effettivo nuovo vettore $\mathbf{z}$ relativo al punto t_{n+1}:

$$\mathbf{z}_0 = \mathbf{v}_0 + r_0 \mathbf{b}$$
$$\mathbf{z}_1 = \mathbf{v}_1 + r_1 \mathbf{b}$$
$$\mathbf{z}_2 = \mathbf{v}_2 + r_2 \mathbf{b}$$
$$\mathbf{z}_3 = \mathbf{v}_3 + r_3 \mathbf{b}$$
$$\mathbf{z}_4 = \mathbf{v}_4 + r_4 \mathbf{b}$$

Ogni metodo MV di ordine p è caratterizzato da uno specifico vettore $\mathbf{r}$ utilizzato per correggere la previsione $\mathbf{v}$. Il vettore $\mathbf{r}$ è scelto in modo da rendere l'algoritmo **stabile**, **accurato** ed **esatto** per soluzioni polinomiali di ordine p.

Nel caso del **metodo Adams-Moulton di quarto ordine**:

$$\mathbf{r}^{\mathrm{T}} = \left[\frac{3}{8}, 1, \frac{11}{12}, \frac{1}{3}, \frac{1}{24}\right]$$

Metodi Multi Value

Il vettore $\mathbf{b}$ viene calcolato in modo che il vettore $\mathbf{z}$ nel nuovo punto soddisfi il sistema differenziale. Ecco come:

$$\mathbf{y}' = \mathbf{f}(\mathbf{y}, t) \Rightarrow h\mathbf{y}' = \mathbf{z}_1 = h\mathbf{f}(\mathbf{y}, t) \Rightarrow \mathbf{v}_1 + \mathbf{b} = h\mathbf{f}(\mathbf{v}_0 + r_0\mathbf{b}, t_{n+1})$$

N.B.: per ogni metodo **MV** il coefficiente $r_1 = 1$.

Forma implicita in $\mathbf{b}$

Grazie alla formulazione dei metodi MV è possibile affrontare in modo del tutto naturale problemi che NON siano nella forma speciale $\mathbf{y}' = \mathbf{f}(\mathbf{y}, t)$. Ad esempio avendo il sistema in forma implicita: $\mathbf{f}\left(\mathbf{y}^{(n)}, \mathbf{y}^{(n-1)}, \ldots, \mathbf{y}', \mathbf{y}, t\right) = \mathbf{0}$ è sufficiente sostituire ad ogni derivata il corrispondente componente del vettore di Nordsieck per avere un sistema non lineare nella sola incognita $\mathbf{b}$.

Metodi Multi Value

Un altro caso molto importante è rappresentato da: $\mathbf{A}\mathbf{y}' = \mathbf{f}(\mathbf{y}, t)$ dove $\mathbf{A}$ è una matrice assegnata. Il sistema non lineare diventa:

$$\mathbf{A}\left[\mathbf{v}_1 + \mathbf{b}\right] = h\mathbf{f}\left(\mathbf{v}_0 + r_0\mathbf{b}, t_{n+1}\right)$$

N.B.: se la matrice $\mathbf{A}$ è **singolare** il sistema risulta **algebrico-differenziale**. Questa formulazione ha quindi anche il pregio di affrontare tale **fondamentale** problema.

N.B.: apparentemente, tramite la formulazione MV basata sul vettore di Nordsieck, si è in grado di risolvere problemi in cui le derivate di qualunque ordine sono implicite o il problema stesso è algebrico-differenziale. Nella pratica però tutte le varianti al problema proposto nella forma standard $\mathbf{y}' = \mathbf{f}(\mathbf{y}, t)$ sono **molto** più **difficili** da risolvere a causa del **sistema non lineare** che si ottiene. Nel caso della forma standard, infatti, il sistema non lineare da risolvere gode di alcune peculiarità favorevoli.

Metodi MV: calcolo dell'errore locale

L'errore locale di un metodo MV di ordine p è: $\mathbf{e}_p = E_p \dfrac{\mathbf{y}^{(p+1)}(\xi) h^{p+1}}{(p+1)!}$ $\qquad E_p = \text{noto}$

Per poter stimare l'errore locale di un metodo MV di ordine p è necessario conoscere un valore approssimato della derivata di ordine $p+1$.

Ciò è molto semplice utilizzando la memorizzazione nel vettore di Nordsieck. L'ultima componente di tale vettore vale:

$$\mathbf{z}_p = \frac{h^p \mathbf{y}_n^{(p)}}{p!}$$

se si conosce tale componente in due punti successivi, $t_{n\text{-}1}$ e t_n allora:

$$\frac{\mathbf{y}^{(p+1)}(\xi) h^{p+1}}{(p+1)!} \approx \left(\frac{h^p \mathbf{y}_n^{(p)}}{p!} - \frac{h^p \mathbf{y}_{n-1}^{(p)}}{p!} \right) \frac{p!}{(p+1)!} = \left(\mathbf{z}_{p,n} - \mathbf{z}_{p,n-1} \right) \frac{p!}{(p+1)!}$$

dato che: $\mathbf{y}^{(p+1)}(\xi) \approx \dfrac{\mathbf{y}_n^{(p)} - \mathbf{y}_{n-1}^{(p)}}{h}$

Metodi MV: variazione passo integrazione

• Il passo di integrazione **deve essere ridotto** quando l'errore locale è maggiore della precisione richiesta dall'utente. In tale caso occorre rifare l'integrazione.

• È possibile anche **aumentare** il passo di integrazione qualora l'errore locale sia molto piccolo.

Tramite l'approccio MV è molto semplice modificare il passo di integrazione da h a h_{new}. Il nuovo vettore $\mathbf{z}$ diventa:

$$\mathbf{z}_i = \left(\frac{h_{new}}{h} \right)^i \mathbf{z}_i \qquad i \geq 0$$

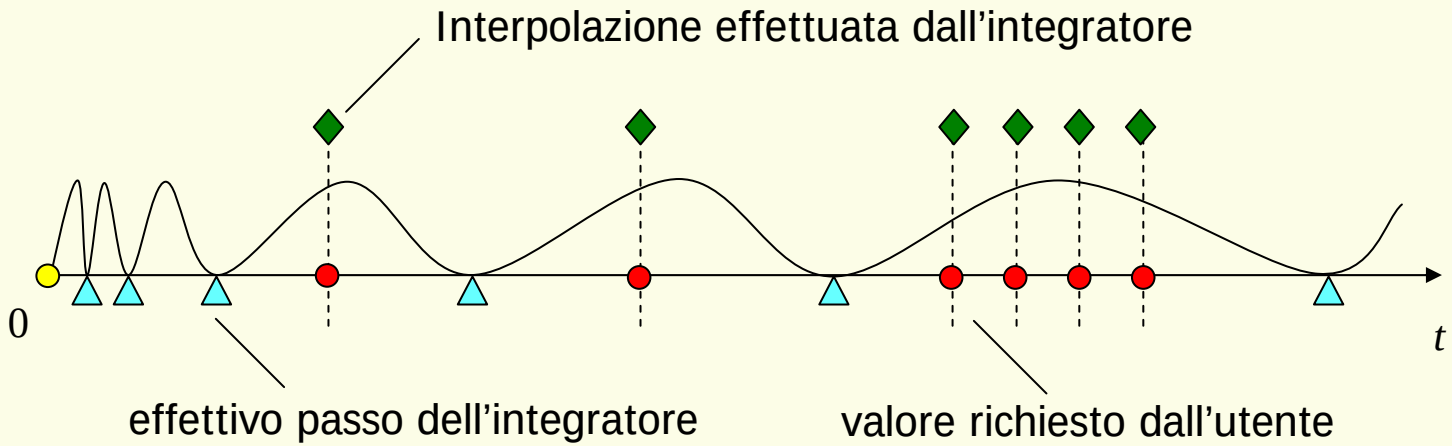
N.B.: questa proprietà è anche sfruttabile per rendere il passo di integrazione **indipendente** dai punti rispetto cui l'utente desidera conoscere la risposta del sistema.

Metodi MV: variazione passo integrazione

Il vettore di Nordsieck permette di effettuare un'interpolazione polinomiale nell'intorno del punto corrente t_n. La previsione richiesta diviene:

$$\mathbf{v}_0 = \sum_{i=0}^{p} \mathbf{z}_i \left(\frac{h_{new}}{h} \right)^i$$

N.B.: l'interpolazione è fattibile con la precisione richiesta soltanto per valori di h_{new} compresi tra t_{n-1} e t_n. Quindi se l'ultimo passo di integrazione oltrepassa il punto richiesto è sufficiente operare un'interpolazione all'indietro per ottenere una risposta corretta entro i margini di errore assegnati.



Metodi MV: variazione ordine del metodo

In molti casi risulta più conveniente modificare, oltre al passo di integrazione, anche l'ordine del metodo.

Utilizzando un metodo di ordine p è possibile stimare le componenti del vettore $\mathbf{z}$ di ordine maggiore. Ad esempio:

$$z_{p+1} = \frac{\left(z_p\right)_n - \left(z_p\right)_{n-1}}{p+1}$$

analogamente per stimare $\mathbf{z}_{p+2}$ occorre aver memorizzato l'ultima componente del vettore $\mathbf{z}$ nei punti t_{n-2}, t_{n-1}, t_n.

N.B.: è quindi possibile stimare tutte le componenti del vettore $\mathbf{z}$ fino all'ordine massimo previsto (dopo un numero sufficiente di passi di integrazione) a partire da un algoritmo di ordine qualsiasi quindi anche dall'ordine 1.

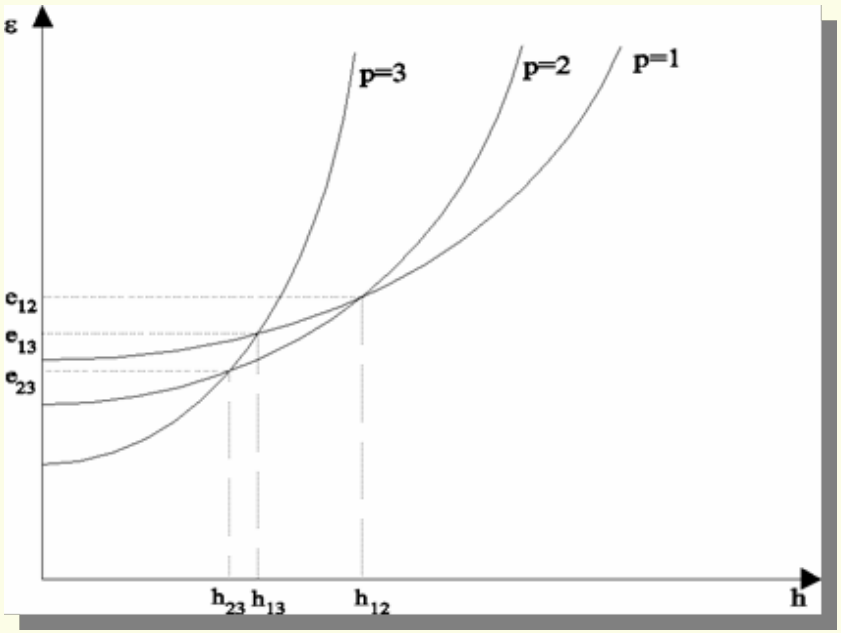
Metodi MV: variazione ordine del metodo

Esempio: si supponga di avere una sola equazione differenziale e di avere fissato l'errore massimo ε accettabile. Allora è necessario che:

$$E_p \frac{y^{(p+1)} h^{p+1}}{(p+1)!} \leq \varepsilon$$

Tramite il vettore di Nordsieck è possibile conoscere $K_p = \left| E_p \frac{y^{(p+1)} h^{p+1}}{(p+1)!} \right|$

e quindi diagrammare le curve $\varepsilon = K_p h^{p+1}$ per vari possibili ordini da adottare.



se $\varepsilon > e_{12}$ vince il primo ordine

se $e_{13} < \varepsilon < e_{12}$ vince il secondo ordine

se $\varepsilon < e_{23}$ vince il terzo ordine

N.B.: per ogni assegnato errore massimo accettabile esiste un passo di integrazione ed un ordine del metodo che risultano ottimali.

Metodi MV: variazione ordine del metodo

Tramite il vettore di Nordsieck e le componenti $\mathbf{z}_p$, $\mathbf{z}_{p+1}$ e $\mathbf{z}_{p+2}$ è possibile stimare gli errori locali commessi con tre algoritmi diversi di ordine $p-1$, p e $p+1$:

$$\mathbf{e}_{p-1} = E_{p-1}\mathbf{z}_p$$
$$\mathbf{e}_p = E_p\mathbf{z}_{p+1}$$
$$\mathbf{e}_{p+1} = E_{p+1}\mathbf{z}_{p+2}$$

Esempio: si consideri una sola equazione differenziale. I passi di integrazione relativi ai tre ordini $p-1$, p e $p+1$ sono:

$$h_{new,p-1} = \alpha_1 h\left(\frac{\varepsilon_A + \varepsilon_R |y_n|}{E_{p-1}z_p}\right)^{\frac{1}{p}}$$

$$h_{new,p} = \alpha_2 h\left(\frac{\varepsilon_A + \varepsilon_R |y_n|}{E_p z_{p+1}}\right)^{\frac{1}{p+1}}$$

$$h_{new,p+1} = \alpha_3 h\left(\frac{\varepsilon_A + \varepsilon_R |y_n|}{E_{p+1}z_{p+2}}\right)^{\frac{1}{p+2}}$$

N.B.: di solito si sceglie $\alpha_3<\alpha_1<\alpha_2<1$ in modo da favorire il metodo con l'ordine corrente quindi quello con ordine inferiore ed infine quello con ordine maggiore.

Nel caso di problemi **stiff** è ancora più importante penalizzare i metodi di ordine elevato in quanto sono meno stabili.

Metodi MV: inizializzazione

Contrariamente ai metodi MS, quelli MV **non necessitano** di altri metodi per essere inizializzati.

L'integrazione può partire utilizzando il metodo del primo ordine appartenente alla famiglia MV prescelta.

Ciò permette di calcolare dopo i primi due passi di integrazione la componente $\mathbf{z}_2$ del vettore di Nordsieck.

Al passo successivo può essere stimata la componente $\mathbf{z}_3$ e così via...

Metodi MV: scelta del primo passo

Si consideri per semplicità una sola equazione differenziale. Poiché l'integrazione ha inizio utilizzando un metodo del primo ordine, la previsione è:

$$y_1 = y_0 + h\,y_0'$$

con un errore locale proporzionale a: $y(t_1) - y_1 \propto \frac{y'' h^2}{2}$

quindi l'errore locale può essere grossolanamente stimato in funzione del passo di integrazione, h, nel seguente modo:

$$y(t_1) - y_1 = y(t_1) - y_0 - h\,y' \approx y_0 - y_0 - h\,y' = -h\,y'$$

Assegnato dall'utente l'errore ε ammissibile e supponendo $y_0' \neq 0$ si ottiene:

$$h = \frac{\varepsilon}{|y_0'|}$$

Selezione dei metodi Multi Valore

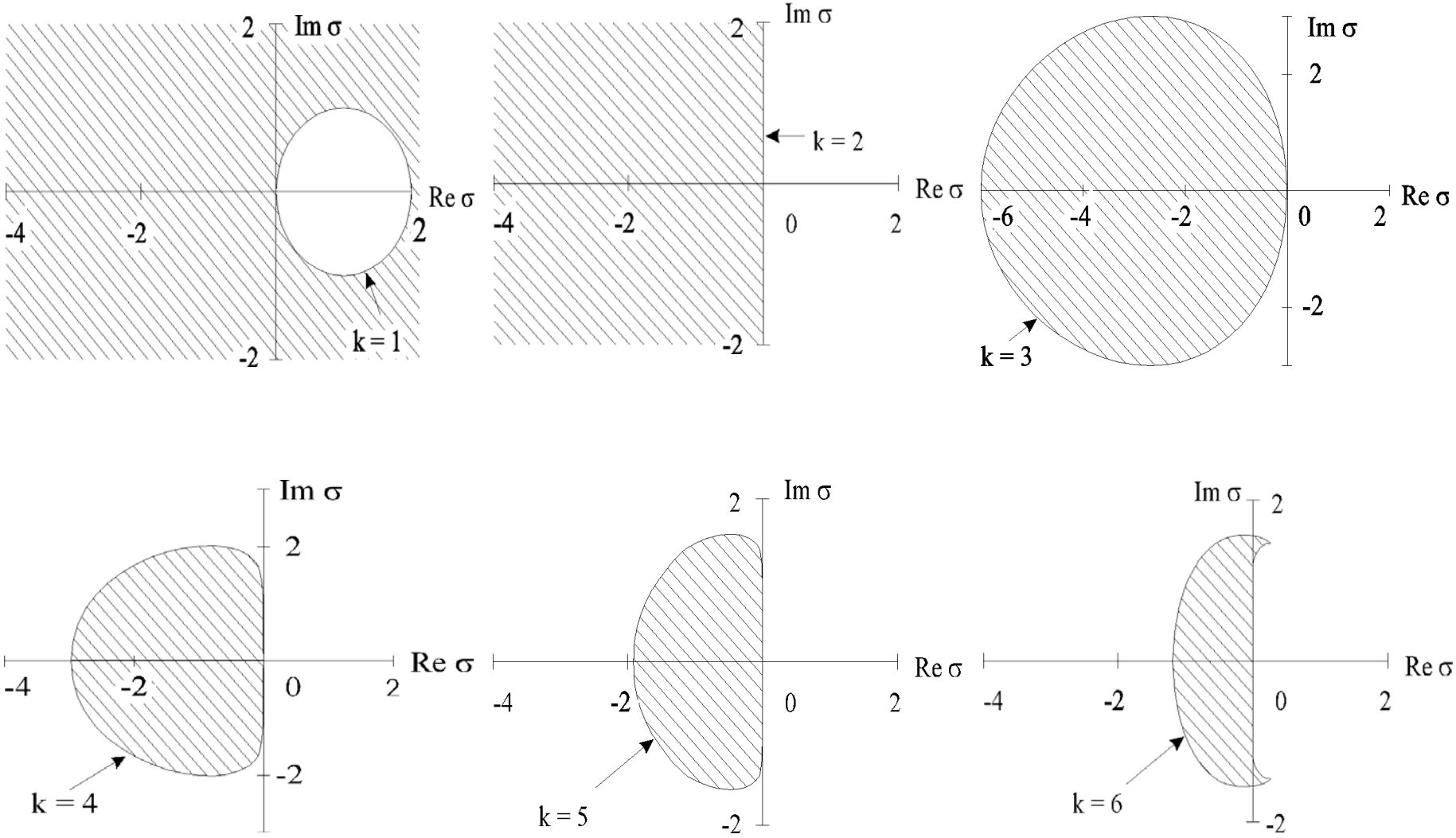
Nella pratica vengono utilizzate due famiglie di metodi MV:

- **Adams-Moulton** per problemi **normali**
- **Gear** per problemi **stiff**

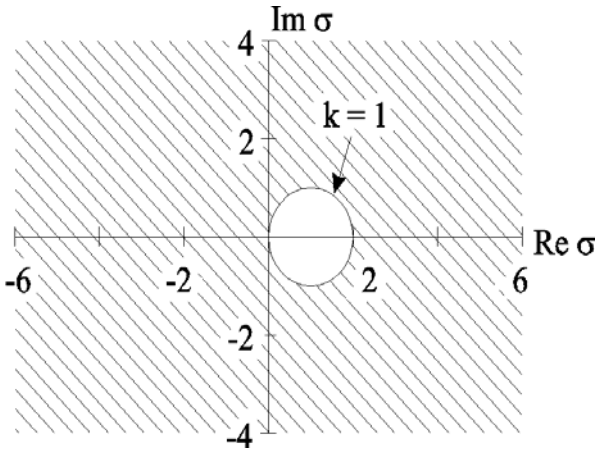
Alcune caratteristiche:

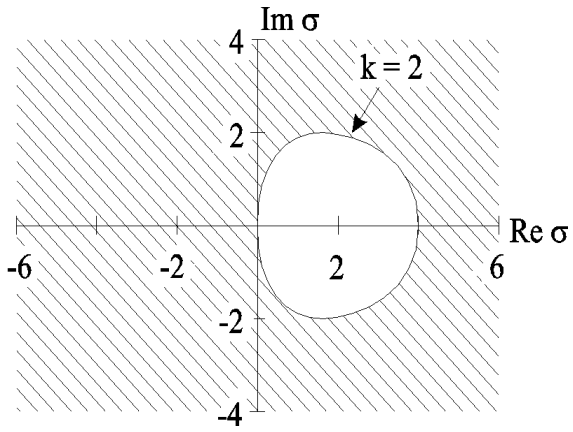
- Gli algoritmi di **Gear** hanno un **errore locale** circa tre volte maggiore di quello degli algoritmi **Adams-Moulton**.
- Gli algoritmi di **Gear** sono **stabili** per problemi **stiff** mentre quelli di **Adams-Moulton** non lo sono per ordini superiori al secondo.
- Tutti i metodi **MV** sono **impliciti** perché soltanto essi risultano stabili.

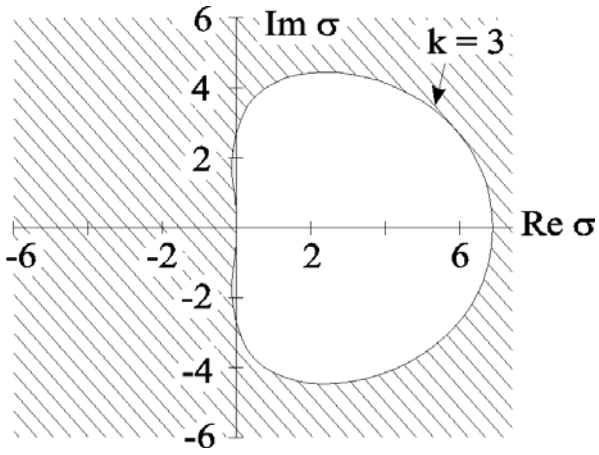
Regione di stabilità metodi Adams-Moulton

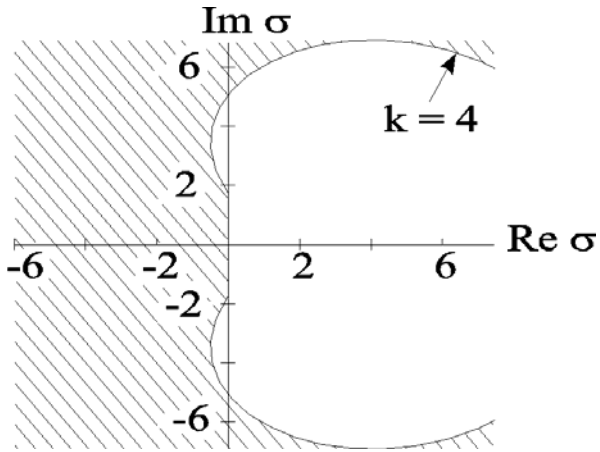


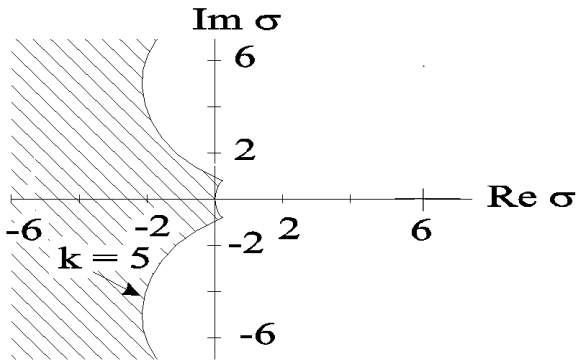
Regione di stabilità metodi Gear

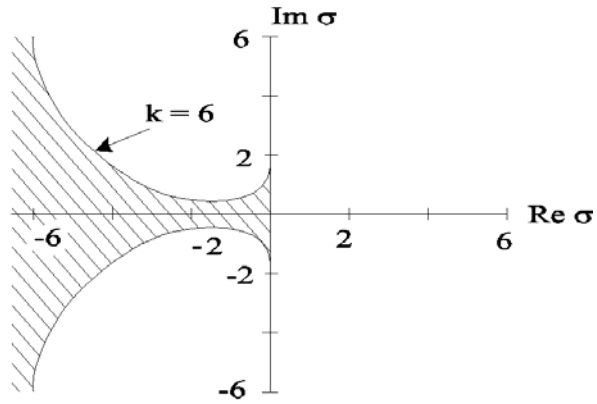












Soluzione del sistema non lineare

Dato che i metodi MV stabili sono **impliciti**, occorre risolvere ad ogni passo di integrazione un **sistema non lineare** nella incognita $\mathbf{b}$:

$$\mathbf{v}_1 + \mathbf{b} = h\mathbf{f}\left(\mathbf{v}_0 + r_0\mathbf{b}, t_{n+1}\right)$$

Si hanno due possibilità:

- utilizzare un metodo iterativo per la soluzione (**Predictor-Corrector**)
- utilizzare una versione del **metodo di Newton** per sistemi non lineari

Soluzione SNL: metodo Predictor-Corrector

Si applica un algoritmo **esplicito Predictor** e quindi un algoritmo **implicito Corrector** la cui formula è ottenuta esplicitamente e quindi viene iterata, teoricamente fino a convergenza, per identificare il punto unito della trasformazione. Se la formula **corrector** non viene iterata fino a convergenza il metodo non può essere definito, quantomeno per via teorica, implicito e quindi si perdono i vantaggi del metodo implicito relativi alla stabilità dell'algoritmo.

Come **previsione** viene utilizzato il vettore $\mathbf{v}$ della formulazione di Nordsieck.

Come **correttore** si calcola $\mathbf{b}$ tramite formula iterativa esplicita:

$$\mathbf{b}_{i+1} = h\mathbf{f}\left(\mathbf{v}_0 + r_0\mathbf{b}_i, t_{n+1}\right) - \mathbf{v}_1 \qquad \text{con} \quad \mathbf{b}_0 = \mathbf{0}$$

Una volta determinato il vettore $\mathbf{b}$ esso viene utilizzato per ricavare il vettore di Nordsieck nel punto $n+1$.

N.B.: ogni iterazione per la ricerca del vettore $\mathbf{b}$ richiede una valutazione del sistema ODE. Il metodo Predictor-Corrector, se funziona, è molto **efficiente** in termine di **tempi di CPU** e richiede **poca memoria**.

Soluzione SNL: metodo Predictor-Corrector

N.B.: nelle applicazioni moderne il numero massimo di iterazioni per la ricerca di $\mathbf{b}$ è limitato a due o tre al massimo. Ciò porta ad una convergenza non molto spinta.

Se però le iterazioni non sono portate a fondo il sistema non è risolto in modo accurato e l'algoritmo risultante è esplicito e perciò non gode delle proprietà di stabilità del metodo implicito.

Nel caso di problemi **non stiff** la scelta di non sprecare calcoli nel portare a convergenza il sistema non lineare è **ragionevole** in quanto in questo caso non è necessario un metodo estremamente stabile.

Nel caso di problemi **stiff** se si dovesse utilizzare il metodo di sostituzione semplice (corrector) sarebbe necessario adottare un passo così piccolo per la convergenza che si perderebbero i vantaggi del metodo stiff (che probabilmente permetterebbe, lato stabilità, di effettuare passi di integrazione decisamente più ampi).

Soluzione SNL: metodo di Newton

Nel caso di sistemi **stiff**, il metodo Predictor-Corrector non funziona.

È opportuno sfruttare la struttura specifica del problema non lineare scaturente dalla **forma normale** del sistema ODE: $\mathbf{y}' = \mathbf{f}(\mathbf{y},t)$.

La previsione resta la stessa, $\mathbf{v}$, quindi si converge alla correzione, $\mathbf{b}$, tramite il metodo di Newton.

N.B.: la previsione $\mathbf{v}$ può essere resa accurata quanto basta per far convergere il metodo di Newton. Infatti se il passo di integrazione, h, tende a zero le componenti $\mathbf{v}_0$ e $\mathbf{v}_1$ tendono a $\mathbf{z}_0 = \mathbf{y}_n$ e $\mathbf{z}_1 = h\mathbf{y}'_n = h\mathbf{f}(\mathbf{y}_n, t_n)$. Quindi al tendere di h a zero la correzione $\mathbf{b}$ deve tendere anch'essa a zero.

Quindi se nascono problemi di convergenza sul sistema non lineare è sufficiente ridurre il passo di integrazione e ripetere il metodo di Newton.

Se la previsione è soddisfacente, il metodo di Newton può convergere anche con passi molto grandi.

Soluzione SNL: metodo di Newton

Se il sistema ODE è nella forma *normale*, il sistema non lineare da risolvere è:

$$\mathbf{v}_1 + \mathbf{b} = h\mathbf{f}\left(\mathbf{v}_0 + r_0\mathbf{b}, t_{n+1}\right)$$

allora lo **Jacobiano del sistema non lineare** da risolvere col metodo di Newton è:

$$\mathbf{G} = \mathbf{I} - h\,r_0\,\mathbf{J}$$

in questo caso il metodo di Newton richiede la soluzione del **sistema lineare**:

$$\mathbf{G}\mathbf{d}_i = \left(\mathbf{I} - h\,r_0\,\mathbf{J}\right)\mathbf{d}_i = -\mathbf{v}_1 - \mathbf{b}_i + h\mathbf{f}\left(\mathbf{v}_0 + r_0\mathbf{b}_i, t_{n+1}\right)$$

da cui si ricava la nuova correzione:

$$\mathbf{b}_{i+1} = \mathbf{b}_i + \mathbf{d}_i$$

N.B.: nel corso di queste iterazioni viene mantenuto costante sia $\mathbf{J}$ che $\mathbf{G}$ e quindi anche la fattorizzazione di $\mathbf{G}$ necessaria per la soluzione del sistema lineare. In generale si mantiene costante $\mathbf{J}$ di passo in passo fintantoché ci sono problemi di convergenza. Se cambia il metodo (r_0) o il passo (h) o $\mathbf{J}$ allora occorre fattorizzare nuovamente $\mathbf{G}$.

Soluzione SNL: metodo di Newton

Le iterazioni del metodo di Newton vengono fermate quando la correzione, $\mathbf{d}$, è inferiore alla tolleranza assegnata alle variabili dipendenti di integrazione.

Si noti che soltanto nel caso di sistema ODE in forma normale, si ha la certezza che diminuendo il passo, h, il metodo di Newton converge in quanto la matrice $\mathbf{G}$ tende alla matrice identità: $\mathbf{I}$.

$$\lim_{h\to 0} \mathbf{G} = \lim_{h\to 0} \left(\mathbf{I} - h\, r_0\, \mathbf{J} \right) = \mathbf{I}$$

Nel caso di **sistemi algebrico-differenziali** nella forma: $\mathbf{A}\mathbf{y}' = \mathbf{f}\left(\mathbf{y}, t\right)$ quindi con la matrice $\mathbf{A}$ singolare, la matrice $\mathbf{G}$ diventa:

$$\mathbf{G} = \mathbf{A} - h\, r_0\, \mathbf{J}$$

con evidenti problemi di convergenza.

Per questo motivo, nei casi più generali, non si può forzare la convergenza riducendo unicamente il passo di integrazione, bensì è necessario **ricorrere a programmi** di soluzione di sistemi non lineari più **sofisticati**.

Efficienza in problemi stiff

La porzione di algoritmo che più richiede risorse in termini di tempo di CPU per l'integrazione di un sistema ODE stiff è la soluzione numerica del **sistema non lineare**:

$$\mathbf{v}_1 + \mathbf{b} - h\mathbf{f}\left(\mathbf{v}_0 + r_0\mathbf{b}, t_{n+1}\right) = \mathbf{0}$$

che richiede a sua volta la soluzione iterativa del sistema lineare:

$$\mathbf{G}\mathbf{d}_i = \left(\mathbf{I} - h\,r_0\,\mathbf{J}\right)\mathbf{d}_i = -\mathbf{v}_1 - \mathbf{b}_i + h\mathbf{f}\left(\mathbf{v}_0 + r_0\mathbf{b}_i, t_{n+1}\right)$$

Quattro sono i **punti** più **delicati**

1. Quando fattorizzare la matrice G

La matrice $\mathbf{G}$ deve essere ricalcolata e quindi fattorizzata ogni volta che viene modificato lo **Jacobiano**, $\mathbf{J}$, o il **passo di integrazione**, h, o l'**ordine** dell'algoritmo e quindi r_0.

Per questo motivo è opportuno non modificare troppo spesso il passo di integrazione o l'ordine dell'algoritmo.

Efficienza in problemi stiff

2. Come fattorizzare la matrice G

In molti problemi lo Jacobiano, $\mathbf{J}$, e quindi la matrice $\mathbf{G}$ sono **sparsi** o **strutturati**. Occorre quindi utilizzare un metodo di fattorizzazione che sfrutti la struttura della matrice $\mathbf{G}$.

3. Quando aggiornare lo Jacobiano J

Finché non viene **compromessa la convergenza** del metodo di Newton, conviene mantenere costante lo Jacobiano dato che il suo calcolo (il più delle volte per via numerica tramite differenze finite) richiede tempi elevati di CPU.

4. Come aggiornare lo Jacobiano J

Se il problema è **sparso**, è possibile sfruttarne la struttura per ridurre il numero di chiamate del sistema differenziale, raggruppando e facendo variare le variabili che sono indipendenti l'una dall'altra all'interno delle singole equazioni differenziali.

Principali routine ODE e ODE/DAE

DIFSUB (Gear, 1971)

GEAR (Hindmarsh, 1974)

LSODE (Hindmarsh, 1980, LSODES, LSODI, LSODAR, ...)

VODE (Brown, Byrne, Dean, 1989, CVODE, VODEPK, ...)

Ivpag (IMSL®, 1995)

BzzOde (Buzzi Ferraris, 1998, BzzOdeStiff, BzzOdeNonStiff, BzzOdeBanded, BzzOdeBlock, ...)

DASSL (Petzold, 1983, DASAC)

DASPK (Brown, Hindmarsh, Petzold, 1992)

SPRINT (Berzins, Furzeland, 1985)

LIMEX (Nowak, Zugck, Deuflhard, 1985)

DAESOL (Bock, Bauer 1997)

BzzDae (Buzzi, 2001, BzzDaeBanded, BzzDae4Blocks, ...)

BzzOde e Mixed Language

È oggi possibile utilizzare la classe in C++ BzzOde per risolvere sistemi ODE con condizioni iniziali all'interno di programmi in Fortran. BzzOde appartiene al pacchetto numerico BzzMath (Buzzi Ferraris, 1998).

Ecco un semplice esempio:

```
call BzzOde(iState,nEq,tIn,tOut,y)
```

È anche possibile fornire ulteriori informazioni quali:

- vincoli sulle variabili dipendenti: **yMin** e **yMax**;
- vincolo sulla variabile indipendente: tCrit;
- tolleranze relative ed assolute: **rTol** e **aTol**.

```
call BzzOde(iState,nEq,tIn,tOut,y,iMinMax,yMin,yMax,iTol,aTol,rTol,iCrit,tCrit)
```

L'utente non deve più preoccuparsi di dimensionare i vettori ausiliari di lavoro e definirne le dimensioni (IWORK, RWORK, LIW, LRW).

```
PROGRAM MixedODE

   implicit real*8 (a-h,o-z)
   parameter(MAX_EQ=10)
   dimension y(MAX_EQ),yMin(MAX_EQ),yMax(MAX_EQ)
   dimension aTol(MAX_EQ),rTol(MAX_EQ)

   nEq = 2          !  Numero equazioni del sistema ODE
   y(1) = 1.d0      !  Condizioni iniziali
   y(2) = 0.5d0

   iMinMax = 1      !  1=yMin e yMax def. utente, 0=No yMin e yMax
   iTol = 1         !  1=tolA e tolR def. utente, 0=Tolleranze di default
   iCrit = 0        !  1=Rispettare tCrit con tCrit >= tOut, 0=No tCrit

   do i = 1, nEq
      aTol(i) = 1.d-10;     rTol(i) = 1.d-6
      yMin(i) = -1.d20;     yMax(i) = 1.d20
   enddo

   tIn = 0.d0
   tOut = 0.d0
   tTotale = 100.d0
   nIntervalli = 10
   deltaTempo = tTotale / nIntervalli

   do i = 1, nIntervalli
      tOut = i * deltaTempo

      if(i == 1)then
         iState = 1  ! Inizio ad integrare un nuovo problema
      else
         iState = 2  ! Continuo ad integrare lo stesso problema
      endif

      if(iCrit == 1)then    !  Gestione dell'eventuale tCrit
         tCrit = tOut
         iState = 1
      endif

      call BzzOdeSolver(iState,nEq,tIn,tOut,y,iMinMax,yMin,yMax,     &
                        iTol,aTol,rTol,iCrit,tCrit)

      if(iCrit == 1)tIn = tOut    !  Gestione dell'eventuale tCrit
   enddo

END
```

```
SUBROUTINE Sisdif(nEq, t, y, f)

   implicit real*8 (a-h,o-z)

   dimension y(nEq), f(nEq)

   f(1) = -Cos(y(1)) + y(2)
   f(2) = - y(2) + Sin(y(1))

   return

END

SUBROUTINE Printo(nEq,y,t)

   !  Questa subroutine viene chiamata ogni volta che BzzOde
   !  ha fatto un passo con successo...

   implicit real*8 (a-h,o-z)

   dimension y(nEq)

   write(*,*)'t , y1, y2 = ',Sngl(t),Sngl(y(1)),Sngl(y(2))

   return

END
```

Bibliografia

- **Guido Buzzi Ferraris, "Metodi Numerici e Software in C++", Addison Wesley, (1998)**
- Atkinson K. E., "Elementary Numerical Analysis", John Wiley & Sons, (1993)
- Brenan K. E., S. L. Campbell and L. R. Petzold, "*Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*", North-Holland, New-Holland, New York, (1989)
- Brown P. N., G. D. Byrne and A. C. Hindmarsh, "VODE: A Variable Coefficient ODE Solver", Siam J. Sci. Stat. Comput., **10**, 1038-1051, (1989)
- Brown P. N., A. C. Hindmarsh and L. R. Petzold, "*A Description of DASPK: A Solver for Large-Scale Differential-Algebraic Systems*", Lawrence Livermore National Report UCRL, (1992)
- Buzzi-Ferraris G., D. Manca, "BzzOde : A New C++ Class for the Solution of Stiff and Non-Stiff Ordinary Differential Equation Systems", Computers Chem. Engng., Vol. 22, **11**, 1595-1621, (1998)
- Byrne G. and A. M. Dean, "The Numerical Solution of Some Kinetics Models with VODE and CHEMKIN II", Comp. Chem., **17**, 297-302, (1993)
- Chan Y. N. I., I. Birnbaum and L. Lapidus, "Solution of Stiff Differential Equations and the Use of Imbedding Techniques", Ind. Eng. Chem. Fundam., **17**, 133-149, (1978)
- Cohen S. D. and A. C. Hindmarsh, "CVODE User Guide", Lawrence Livermore National Laboratory report UCRL-MA-118618, Sept. 1994.

Bibliografia

- Curtis A. R., M. J. D. Powell and J. K. Reid, “On the estimation of sparse Jacobian matrices”, J. Inst. Maths. Applns., **13**, 117-120, (1974)
- Gear C. W., “Algorithm 407, DIFSUB for Solution of Ordinary Differential Equations”, Comm. ACM, vol. 14, 3, 185-190, (1971)
- Hindmarsh A. C., “GEAR: Ordinary Differential Equation System Solver”, Report UCID-30001, Rev. 3, Lawrence Livermore Laboratory, Livermore, CA, (1974)
- Hindmarsh A. C. “LSODE and LSODI, Two New Initial Value Ordinary Differential Equation Solvers”, ACM SIGNUM Newsletter, **15**, 10-11, (1980)
- Manca D., T. Faravelli, G. Pennati, G. Buzzi Ferraris and E. Ranzi, “*Numerical Integration of Large Kinetic System*”, ICheaP-2 Conf. Ser., ERIS, 115-121, (1995)
- Manca D., G. Buzzi-Ferraris, T. Faravelli and E. Ranzi, “Numerical Problems in the Solution of Oxidation and Combustion Models”, Combustion Theory and Modelling, 5, 185-199, (2001)
- Moler C., “Numerical Computing with MATLAB”, The Mathworks, (2004)
- Petzold L., “Automatic selection of Methods for Solving Stiff and Non-stiff Systems of Ordinary Differential Equations”, Siam J. Sci. Stat. Comput., **4**, 136-148, (1983)
- Petzold L. R., "A Description Of DASSL: A Differential/Algebraic System Solver", Scientific Computing, R. S. Stepleman et al. (Eds.), North-Holland, Amsterdam, pp. 65-68, (1983)

Bibliografia

- Pitz W. J. and C. K. Westbrook, “Chemical Kinetics of the High Pressure Oxidation of n-butane and its Relation to Engine Knock”, Combust. Flame, **63**, 113-133, (1986)
- Radhakrishnan K, A. C. Hindmarsh, “Description and Use of LSODE, the Livermore Solver for Ordinary Differential Equations”, NASA Ref. Publ. 1327, Livermore, (1993)
- Shampine L. F., “What is Stiffness? Stiff Computation”, R. C. Aiken ed., Oxford University Press, New York, 1-16, (1985)
- Shampine L. F., C. W. Gear, “A User’s View of Solving Stiff Ordinary Differential Equations”, SIAM Rev. Vol. 21, 1-17, (1979)
- Sincovec R. F., A. M. Erisman, E. L. Yip and M. A. Epton, “Analysis of Descriptor Systems Using Numerical Algorithms”, IEEE Trans. on Aut. Contr., 26, 139-147, (1981)
- http://scienceworld.wolfram.com/biography/topics/BranchofScience.html
- http://www.mathworks.com/moler