Laboratorio di Calcolo Numerico
"Moti ordinati e Metodi discreti"

EnneKappa

Corso di Fisica Matematica A.A. 2005-2006
Università di Padova - Ingegneria Informatica.

Sommario

In questo lavoro si vogliono studiare alcuni sistemi dinamici elementari utilizzando algoritmi di integrazione one-step quali Eulero Cauchy, Heun e Runge Kutta Gill.
Dopo una breve introduzione sulla teoria dei metodi di integrazione adottati si presentano i risultati delle elaborazioni in forma grafica. Si fanno anche delle considerazioni relative agli errori che nascono dalla applicazione dei vari metodi di integrazione e si accenna alle problematiche che emergono considerando gli algoritmi come sistemi dinamici. Per implementare gli algoritmi si è utilizzato il linguaggio FORTRAN perchè mi ha permesso di definire la precisione di calcolo e l'ambiente MatLab per la rappresentazione grafica dei risultati ottenuti.
Gli algoritmi di integrazione sono testati su alcuni tipi di sistemi dinamici presentati durante il corso di Fisica Matematica, quali Pendolo, Pendolo con smorzamento, Pendolo con molla e Van Der Pol. I grafici vengono generati a partire da un'opportuna serie di condizioni iniziali scelta in base al sistema in studio.
In conclusione si propone un confronto tra gli algoritmi, valutandone le differenze di approssimazione.

La figura in copertina è stata generata da un sistema Pickover:

$$\begin{cases} \dot{x} = sin(ay) - zcos(bx) \\ \dot{y} = zsin(cx) - cos(dy) \\ \dot{z} = sin(x) \end{cases}$$

con parametri:

$a = -1.22818791946309$
$b = 2.76952348993289$
$c = 1.846759$
$d = 1.53020134228188$

Indice

1 Premessa

Per semplicità saranno considerati solo sistemi di equazioni differenziali autonomi di tipo meccanico nella forma:

$$\begin{cases} \dot{x} = v \\ \dot{v} = f(x, v) \end{cases} \tag{1.1}$$

In generale non esistono metodi analitici per la risoluzione di equazioni differenziali e si ricorre quindi a metodi di integrazione numerica.
La ricerca di una soluzione numerica approssimata di un problema differenziale si basa sulla costruzione di un problema non differenziale approssimato e sulla risoluzione numerica di quest'ultimo.
Dato un determinato intervallo $I = [a, b]$ in cui esiste ed è unica (*Teorema di Cauchy*) la soluzione y_x definiamo il passo di integrazione:

$$\tau = \frac{(b-a)}{N} > 0 \text{ tale che } a = x_0 \; b = x_N$$
$$x_n = x_0 + n\tau \text{ , } n = 0, ..., N \text{ con N numero di passi su I.}$$

Se τ è variabile si parla di metodi a passo variabile.
Integrare numericamente un'equazione differenziale significa sostituire un processo continuo con uno discreto:

$$y'(x_n) = \frac{\Delta(x_n)}{\tau} + O(\tau)$$
$$\text{con } O(\tau) \text{ errore di troncamento locale.}$$

Da questo processo di integrazione numerica vengono generati due tipi di errore: l'*errore di troncamento* dovuto alla sostituzione dell'equazione differenziale con un'equazione alle differenze; l'*errore di arrotondamento* dovuto all'uso dell'aritmetica finita dei calcolatori.
La cosa importante è che la propagazione degli errori non sia amplificata a tal punto da distruggere la validità del risultato, cioè che l'algoritmo sia **stabile**.
In letteratura si possono trovare molti metodi di integrazione numerica e si possono catalogare in due classi:

1. **Metodi a passo singolo** (*one-step* Runge Kutta) dove ogni valore viene calcolato unicamente a partire dal precedente.

2. **Metodi a passo multiplo** (*multistep* predictor-corrector) dove ogni valore viene calcolato utilizzando le k approssimazioni precedenti.

Un'altra classe di metodi è quella dei **Metodi Simplettici a scorrimento** che vengono usati per la discretizzazione di sistemi conservativi.

2 Metodi con le serie Taylor

I metodi one-step si basano sullo sviluppo in serie di Taylor della soluzione dell'equazione differenziale. Il primo passo consiste nello sviluppare in serie di Taylor la soluzione $y(x_n + 1)$ nel punto x_n :

$$y(x_{n+1}) = \sum_{k=0}^{n} \frac{y^{(k)}(x_n)}{k!}(x - x_n)^k + O((x - x_n)^k)$$

considero $x - x_n = \tau$ e $x_i - \tau = x_{i-1}$

$$y(x_{n+1}) = \sum_{k=0}^{n} \frac{y^{(k)}(x_n)}{k!}(\tau)^k + O((\tau)^k) \tag{2.1}$$

Lo sviluppo di Taylor troncato al primo ordine è la base per il metodo di Eulero; considerando invece un troncamento al secondo ordine per la regola di derivazione delle funzioni composte si ottiene:

$$y(x_{n+1}) = y(x_n) + \tau f_n + \frac{\tau^2}{2}[f_x + f_y f]_n + O(\tau^3) \tag{2.2}$$

Quindi, conoscendo la derivata nel punto x_{i-1} della soluzione, si può ottenere il valore della soluzione nel punto successivo $x(i)$.
Utilizzando questa equazione si possono dedurre le soluzioni approssimate di un sistema di N equazioni differenziali del primo ordine.
Se però si ha bisogno di una approssimazione migliore bisogna considerare derivate di ordine superiore al primo nell'equazione (2.1) complicandone la valutazione .
In questi casi conviene allora utilizzare metodi come il Runge-Kutta che applicano uno sviluppo in serie di Taylor su ogni punto di campionamento, di cui si parlerà in seguito.

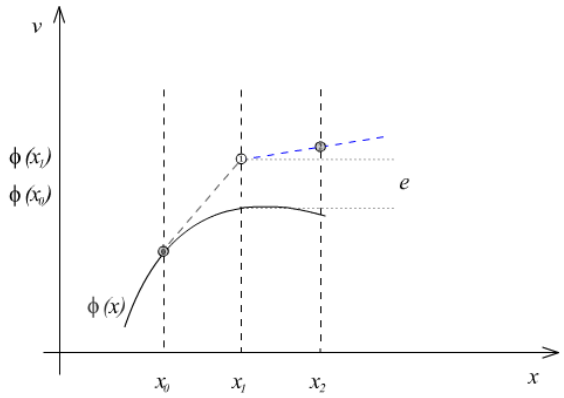
2.1 Metodo di Eulero Cauchy

L'algoritmo di Eulero o metodo della tangente si ottiene troncando lo sviluppo di Taylor al primo ordine nell'equazione (2.1).
Nel nostro caso sarà applicato al problema meccanico in forma della (1.1):

$$\begin{cases} x_i = x_{(i-1)} + \tau v_{(i-1)} \\ v_i = v_{(i-1)} + \tau f(x_{(i-1)}, v_{(i-1)}) \end{cases} \tag{2.3}$$

Graficamente il metodo consiste nello spostarsi lungo le tangenti nel punto successivo a partire dal punto precedente e questo causa una propagazione dell'errore abbastanza consistente come si vede in Figura 2.

Il metodo di Eulero presenta un errore di troncamento locale dell'ordine di $O(\tau^2)$ mentre l'errore di troncamento globale è $O(\tau)$, motivo per cui è inutilizzabile per i problemi trattati.




```
!Algoritmo di Eulero
!per un sistema di EDO
!di tipo meccanico
!...
!input
x(1)=x0
v(1)=v0
!...
\\inizio ciclo
 DO i=2,n
    \\metodo di Eulero semplice
    x(i) = x(i-1) + T*v(i-1)
    v(i) = v(i-1) + T*f(x(i-1))
 END DO
!...
!output
```

Algoritmo FORTRAN

Figura 1: Interpretazione grafica del metodo di Eulero su una generica funzione $\phi(x)$

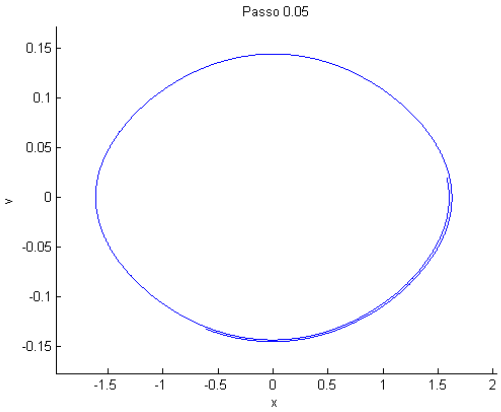


Figura 2: Sistema Pendolo. Errore dell'ordine di $O(\tau)$ con l'algoritmo di Eulero

3 Metodi Runge Kutta

I metodi di Runge Kutta si ottengono sostituendo alle derivate della funzione $f(x, y)$ dell'equazione (2.2) delle funzioni più semplici che le approssimano.

$$y_{n+1} = y_n + \omega_1 k_1 + \omega_2 k_2 + ... + \omega_m k_m \tag{3.1}$$

dove ω_i sono dei coefficienti numerici tali che $\sum_{i=1}^{m} \omega_i = 1$
e le k_i con $i = 1, \ldots, m$ sono le funzioni di Runge Kutta:

$$\begin{aligned} k_1 &= \tau f(x, y) \\ k_2 &= \tau f(x + \alpha_2 \tau, y + \beta_{2,1} k_1(x, y)) \\ k_3 &= \tau f(x + \alpha_3 \tau, y + \beta_{3,1} k_1 + \beta_{3,2} k_2(x, y)) \\ &\ldots \\ k_m &= \tau f(x + \alpha_m \tau, y + \tau \sum_{i=i}^{m-1} b_{m,i} k_i(x, y)) \end{aligned} \tag{3.2}$$

L'ordine del metodo di Runge Kutta è l'esponente che presenta τ sull'ultima funzione k_m.
Nei casi qui trattati stadio e ordine coincidono e saranno considerati:

1. **Metodi a due stadi** Metodo di Heun e metodo di Eulero Modificato.
2. **Metodi a quattro stadi** Runge Kutta classico e Runge Kutta Gill.

Il Runge Kutta a due stadi è definito dall' equazione (3.1) con $m = 2$:

$$y_{n+1} = y_n + \omega_1 \tau f(x_n, y_n) + \omega_2 \tau f(x_n + \alpha\tau, y_n + \beta\tau f(x_n, y_n)) \tag{3.3}$$

I quattro coefficienti $\omega_1, \omega_2, \alpha, \beta$ si calcolano in modo che la (3.3) coincida con la (2.2) cioè con la soluzione sviluppata in serie di Taylor fino al secondo ordine:

$$\begin{cases} \omega_1 + \omega_2 = 1 \\ \alpha = \beta = \dfrac{1}{2\omega_2} \text{ con } (\omega_2 \neq 0) \end{cases} \tag{3.4}$$

Le soluzioni di questo sistema sono ∞^1. Ogni scelta di ω_1, ω_2, α e β che soddisfano il sistema (3.4) genera un metodo per l'integrazione numerica più o meno efficiente.
Il Runge Kutta a quattro stadi si basa sulla stessa equazione (3.1) ma in questo caso $m = 4$ e attraverso la scelta di opportuni coefficienti si ottengono diversi metodi come quello classico e quello di Gill.

3.1 Metodo di Heun

Il metodo di Heun o metodo di Eulero Migliorato è un Runge Kutta a due stadi e risolve il sistema (3.4) con questa configurazione di coefficienti:

$$\begin{cases} \omega_1 = \omega_2 = \dfrac{1}{2} \\ \alpha = \beta = 1 \end{cases}$$

$$\implies y_{n+1} = y_n + \frac{\tau}{2}[f(x_n, y_n) + f(x_n + \tau, y_n + \tau f(x_n, y_n))] \tag{3.5}$$

e nel caso di sistema di equazioni del tipo (1.1):

$$\begin{cases} x_{n+1} = x_n + \dfrac{\tau}{2}[(v_n + \tau) + v_n] \\ \\ v_{n+1} = v_n + \dfrac{\tau}{2}[f(x_n, y_n) + f(x_n + \tau, y_n + \tau f(x_n, y_n))] \end{cases}$$

Graficamente l'algoritmo utilizza il metodo della tangente due volte, riporta la seconda tangente nel punto iniziale e calcola la somma vettoriale tra le due. (Vedi Figura 3)

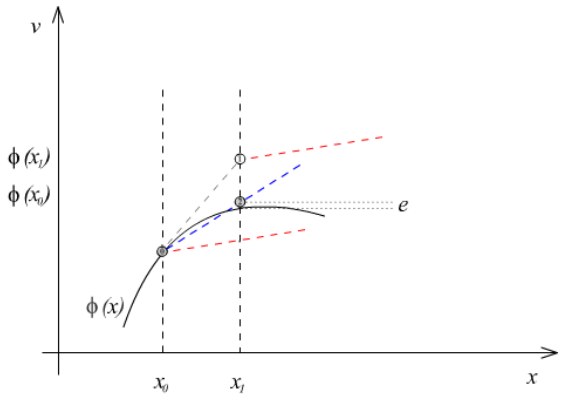


Figura 3: Interpretazione grafica del metodo di Heun su una generica funzione $\phi(x)$

```
!input
x(1)=x0
v(1)=v0
!...
\\inizio ciclo
 DO i=2,n
    \\metodo di Eulero
    x(i) = x(i-1) + T*v(i-1)
    v(i) = v(i-1) + T*f(x(i-1))
    t_x=x(i); \\variabile temporanea
    \\metodo di Heun
    x(i)=x(i-1) + (T/2)*(v(i-1)+v(i))
    v(i)=v(i-1) + (T/2)*(f(x(i-1))+f(t_x))
 END DO
!...
!output
```

Algoritmo FORTRAN

3.2 Metodo di Eulero Modificato

Un altro metodo Runge Kutta a due stadi è il metodo di Eulero modificato che utilizza questi coefficienti:

$$\begin{cases} \omega_1 = 0 \\ \omega_2 = 1 \\ \alpha = \beta = \dfrac{1}{2} \end{cases}$$

$\Longrightarrow$

$$y_{n+1} = y_n + \tau f[x_n + \frac{\tau}{2}, y_n + \frac{\tau}{2} f(x_n, y_n)] \tag{3.6}$$

e nel caso di sistema di equazioni del tipo (1.1):

$$\begin{cases} x_{n+1} = x_n + \tau[v_n + \dfrac{\tau}{2} v_n] \\ v_{n+1} = v_n + \tau f[x_n + \dfrac{\tau}{2}, v_n + \dfrac{\tau}{2} f(x_n, v_n)] \end{cases}$$

3.3 Metodo di Runge Kutta classico

Uno dei metodi one-step di integrazione numerico più usati è il Runge Kutta a $m = 4$ stadi definito con:

$$y_{n+1} = y_n + \frac{1}{6}[k_1 + 2k_2 + 2k_3 + k_4] \tag{3.7}$$

con

$$\begin{aligned} k_1 &= \tau f(x_n, y_n) \\ k_2 &= \tau f(x_n + \frac{\tau}{2}, y_n + \frac{k_1}{2}) \\ k_3 &= \tau f(x_n + \frac{\tau}{2}, y_n + \frac{k_2}{2}) \\ k_4 &= \tau f(x_n + \tau, y_n + k_3) \end{aligned}$$

```
!input x(1)=x0 v(1)=v0 !...
\\inizio ciclo
 DO i=2,n
    h1=v(i-1)
    h2=v(i-1)+T/2*h1
    h3=v(i-1)+T/2*h2
    h4=v(i-1)+T*h3
    x(i)=x(i-1)+T/6*(h1+2*h2+2*h3+h4)

    k1 = f(x(i-1),v(i-1))
    k2 = f(x(i-1)+t/2,v(-i)+T/2*k1)
    k3 = f(x(i-1)+t/2,v(-i)+T/2*k2)
    k4 = f(x(i)+t,v(-i)+T/2*k3)
    v(i)=v(i-1)+T/6*(k1+2*k2+2*k3+k4)
 END DO
!... !output
```

Algoritmo FORTRAN

3.4 Metodo Runge Kutta Gill

$$y_{n+1} = y_n + \frac{1}{6}[k_1 + (2-\sqrt{2})k_2 + (2+\sqrt{2})k_3 + k_4] \tag{3.8}$$

con

$$\begin{aligned}
k_1 &= \tau f(x_n, y_n) \\
k_2 &= \tau f(x_n + \frac{\tau}{2}, y_n + \frac{k_1}{2}) \\
k_3 &= \tau f[x_n + \frac{\tau}{2}, y_n + \frac{1}{2}(-1+\sqrt{2})k_1 + (1-\frac{1}{2}\sqrt{2})k_2] \\
k_4 &= \tau f[x_n + \tau, y_n - \frac{1}{2}\sqrt{2}k_2 + (1+\frac{1}{2}\sqrt{2})k_3]
\end{aligned}$$

Algoritmo FORTRAN

```
!input x(1)=x0 v(1)=v0 !...
\\inizio ciclo
 DO i=2,n
    h1=v(i-1)
    h2=(v(i-1)+T/2*h1)
    h3=(v(i-1)+T*(-1.+sqrt(2.))*h1+T*(1.-1/2*sqrt(2.))*h2)
    h4=(v(i-1)-T/2*sqrt(2.)*h2+T*(1.+1/2*sqrt(2.))*h3)
    x(i)=x(i-1)+T/6*(h1+(2.-sqrt(2.))*h2+(2-sqrt(2.))*h3+h4)

    k1 = f(x(i-1),v(i-1))
    k2 = f(x(i-1)+1/2*T,v(i-1)+T/2*h1)
    k3 = f(x(i-1)+1/2*T,v(i-1)+T*(-1.+sqrt(2.))*h1+T*(1.-1/2*sqrt(2.))*h2)
    k4 = F(x(i-1)+T,v(i-1)-T/2*sqrt(2.)*h2+T*(1.+1/2*sqrt(2.))*h3)
    v(i)=v(i-1)+T/6*(k1+(2.-sqrt(2.))*k2+(2.-sqrt(2.))*k3+k4)
 END DO
!... !output
```

4 Grafici

In questa appendice sono raccolti tutti i grafici generati da MatLab con l'algoritmo di Heun.

4.1 Pendolo

Ritratto in fase di un pendolo utilizzando l'algoritmo di Heun.

$$\begin{cases} \dot{x} = v \\ \dot{v} = -\omega^2 sin(x) \end{cases} \tag{4.1}$$

Nella Figura 4 si può notare come le linee delle rotazioni si interrompono a

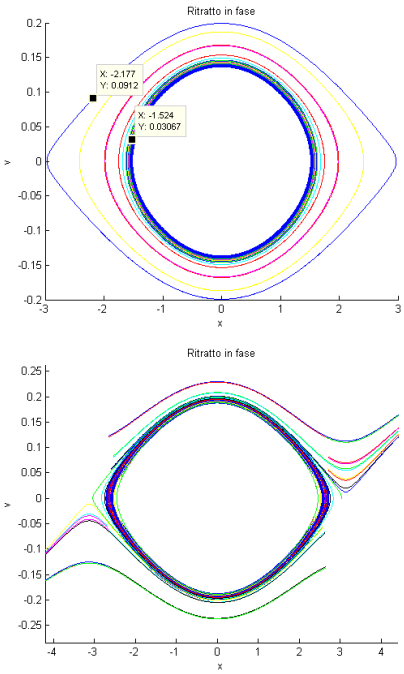


Figura 4: $\tau = 0.05$, $\omega = 0.1$, $x_0 = \pi/2$ $v_0 = 0.01$

causa della differenza di periodo rispetto alle librazioni: il programma esegue lo stesso numero di iterazioni per ogni moto e quindi i moti di periodo più grande risulteranno interrotti.

`Per il codice vedere pendolo.m in Appendice B`

4.2 Pendolo smorzato

Ritratto in fase di un pendolo smorzato utilizzando l'algoritmo di Heun.

$$\begin{cases} \dot{x} = v \\ \dot{v} = -\omega^2 sin(x) - 2\mu v \end{cases} \tag{4.2}$$

Anche in Figura 5 si può notare un'interruzione dei moti dovuto al fatto che la

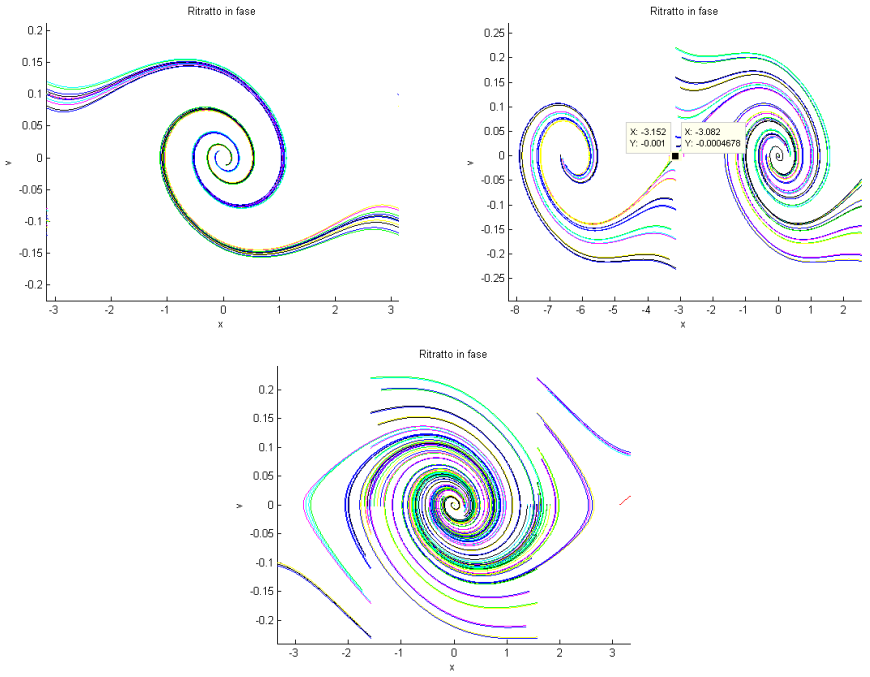


Figura 5: $\tau = 0.05$, $\omega = 0.1$, $\mu = 0.02$, $x_0 = \pi/2e\pi$, $v_0 = 0.01$

figra non è periodica.
Interessanti sono i moti sulle separatrici nei pressi del valore 2π per il quale i moti tendono per tempi infiniti.

`Per il codice vedere pendolosm.m in Appendice B`

4.3 Pendolo con molla

Ritratto in fase di un pendolo con molla utilizzando l'algoritmo di Heun.

$$\begin{cases} \dot{x} = v \\ \dot{v} = -\omega^2 sin(x) + \Omega^2 sin(x) cos(x) \end{cases} \tag{4.3}$$

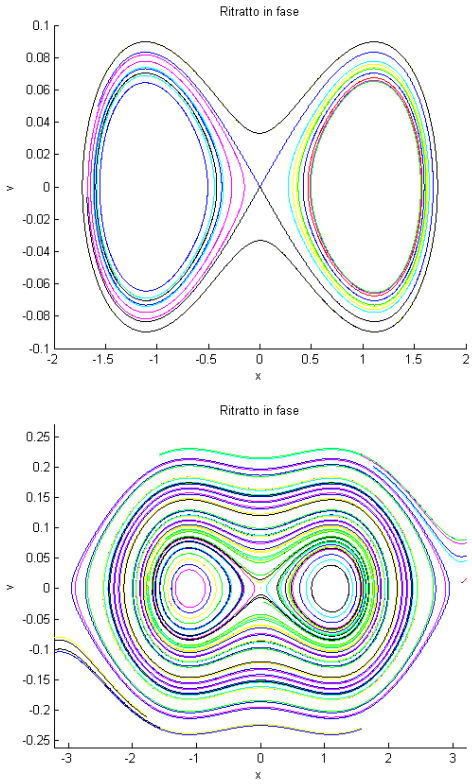


Figura 6: $\tau = 0.05$, $\omega = 0.1$, $\Omega = 0.15$, $x_0 = \pi/2$, $v_0 = 0.01$

Ho ritenuto opportuno inserire solo i ritratti in fase (Figura 6) in nel caso in cui $\frac{\omega^2}{\Omega^2} \leq 1$ cioè in presenza di doppia buca dovuta alla biforcazione a forchetta.

`Per il codice vedere pendolom.m in Appendice`

4.4 Van Der Pol

Ritratto in fase del sistema Van Der Pol utilizzando l'algoritmo di Eulero Migliorato.

$$\begin{cases} \dot{x} = \beta[y - (\frac{1}{3}x^3 - x)] \\ \dot{y} = -\beta^{-1}x \end{cases} \tag{4.4}$$

In Figura 7 si può vedere il moto per $\beta << 1$, cioè in presenza di ciclo limite.

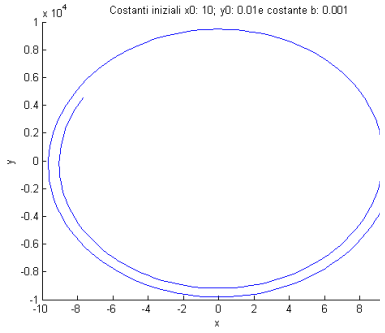


Figura 7: $\tau = 0.05$, $\beta = 0.01$, $x_0 = 0.01$, $y_0 = 2$

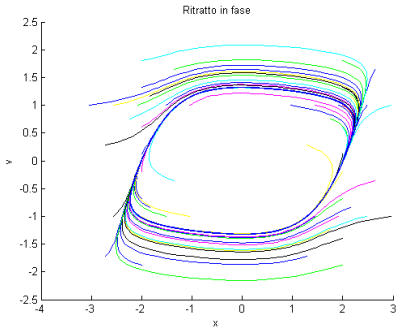


Figura 8: $\tau = 0.05$, $\beta = 0.01$, $x_0 = 0.01$, $y_0 = 2$

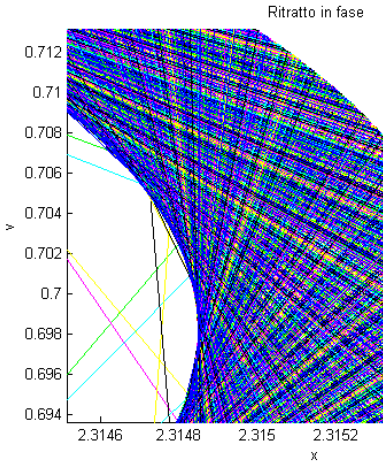


Figura 9: $\tau = 0.05$, $\beta = 3$, $x_0 = 0.001$, $y_0 = 0.02$

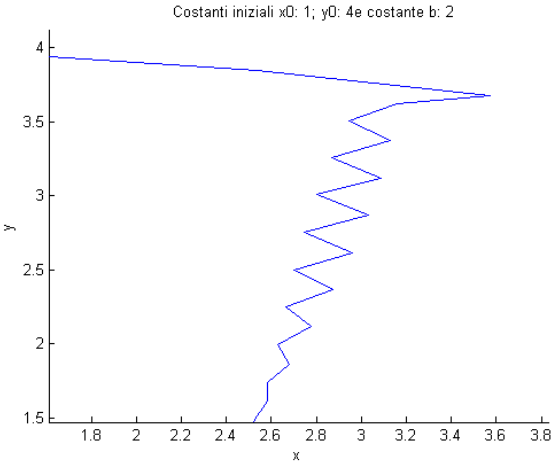


Figura 10: Effetti dell'errore di troncamento con passo di integrazione grande.

Nelle Figure 9 e 10 si vogliono sottolineare gli effetti dell'errore di troncamento sui problemi *stiff* e le conseguenze di un passo di integrazione troppo grande.

`Per il codice vedere vander.m in Appendice`

5 Errori

Anche se tutti gli algoritmi trattati in questo lavoro sono stabili, essi si comportano in modo diverso a seconda del problema.
Dalla Figura 12 può notare come gli algoritmi Runge Kutta classico al quarto ordine e il Runge Kutta Gill sono assolutamente equivalenti per il sistema pendolo semplice in quanto l'errore valutato mediante differenza tra le energie calcolate con i due algoritmi è zero.

$$\begin{cases} \dot{x} = v \\ \dot{v} = -\omega^2 sin(x) \end{cases} \tag{5.1}$$

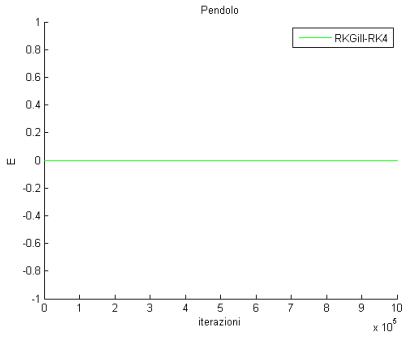


Figura 11: Differenza tra l'errore di RK4 e quello di RKGill

Invece per il sistema Van Der Pol sono abbastanza diversi anche se la loro differenza non esplode essendo l'errore in entrambi i casi dell'ordine di $O(\tau^4)$.

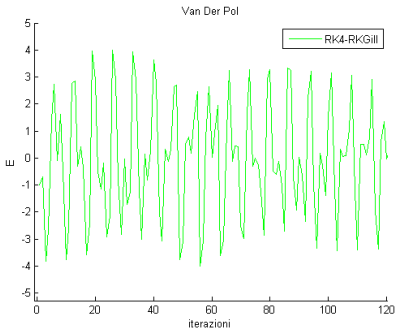


Figura 12: Differenza tra l'errore di RK4 e quello di RKGill

Le differenze tra gli algoritmi si possono valutare sia effettuando una differenza tra le posizioni e le velocità, sia sfruttando la conservazione dell'energia quando possibile:

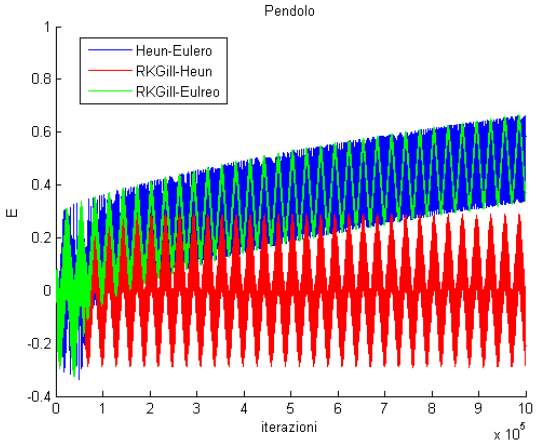


Figura 13: Differenze sulla posizione e velocità nel sistema Pendolo

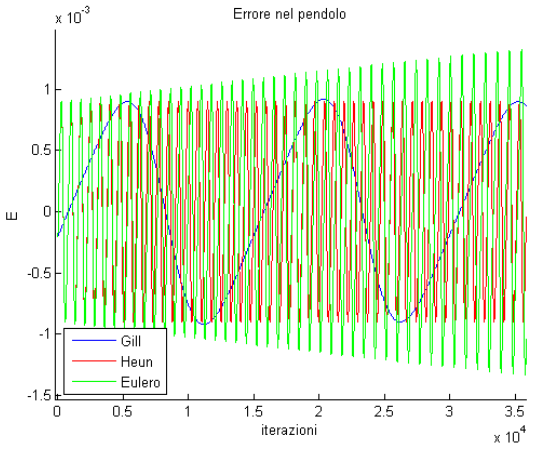


Figura 14: Differenze sull'energia nel sistema pendolo

In entrambi i casi si può notare un'esplosione delle differenze con l'algoritmo di Eulero Cauchy e un mantenimento costante della differenza tra Runge Kutta Gill e metodo di Heun. Questo avviene perchè l'errore di troncamento globale del metodo di Eulero è dell'ordine di $O(\tau)$.
Questo effetto si verifica anche nel sistema Pendolo con molla:

$$\begin{cases} \dot{x} = v \\ \dot{v} = -\omega^2 sin(x) - \Omega^2 sin(x) cos(x) \end{cases} \tag{5.2}$$

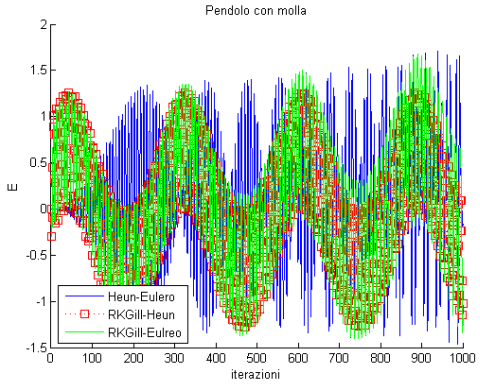


Figura 15: Differenze sulla posizione e velocità nel sistema Pendolo con molla

Applicando il metodo Eulero al sistema Van Der Pol si può notare che il ritratto in fase è qualitativamente diverso da quello che produce un altro algoritmo.

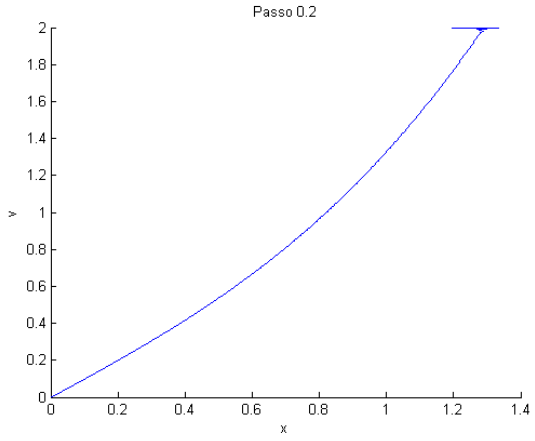


Figura 16: Algoritmo di Eulero su Van Der Pol

Un altro effetto che si può notare è la curva a zigzag in alto a destra della Figura 9 del quale parlerò nel paragrafo successivo. Un'altra analisi qualitativa degli algoritmi può essere fatta utilizzando il sistema Pendolo semplice di equazione (4.1), osservando il risultato numerico ottenuto applicando gli algoritmi trattati. Nel sistema pendolo i moti sono periodici e quindi il ritratto in fase presenterà delle traiettorie chiuse; la mancata chiusura delle traiettorie è dovuta all'errore accumulato durante l'operazione di integrazione numerica.

6 Algoritmi

Può essere interessante lo studio degli algoritmi come sistemi dinamici.
Essi infatti rispondono diversamente a seconda dell'input cioè a seconda di quale sistema si vuole approssimare.
In queste condizioni possono emergere i problemi *stiff* che si presentano quando si vuole studiare un sistema con un transitorio molto veloce dopo il quale il sistema si stabilizza.
Usando il sistema di Van Der Pol con $\beta >> 1$:

$$\begin{cases} \dot{x} = \beta(y - \gamma(x)) \\ \dot{y} = -\frac{1}{\beta}x \end{cases} \tag{6.1}$$

è interessante vedere come i vari algoritmi si adattano al sistema al superamento verticale della cubica $\gamma(x) = \frac{x^3}{3} - x$ vicino cui il moto rimane confinato (si tratta una regione molto piccola pari a β^-2).
La curva a 'zigzag' è dovuta alla presenza di moti opposti nelle vicinanze della cubica $\gamma(x)$ come trattato nel paragrafo 5: l'algoritmo con passo grande prima si trova nella regione di moti verso destra, poi 'salta' la cubica e si trova nella regione di moti verso sinistra e prosegue a 'zigzag' senza mai riuscire ad allinearsi.
Si può notare che per passi di discretizzazione uguali gli algoritmi presentano uno 'smorzamento' diverso.

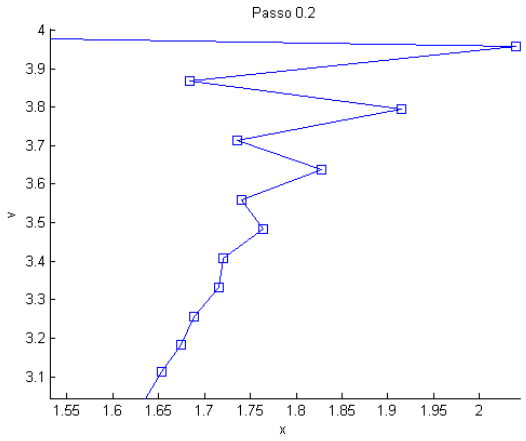


Figura 17: Algoritmo RK4 su Van Der Pol

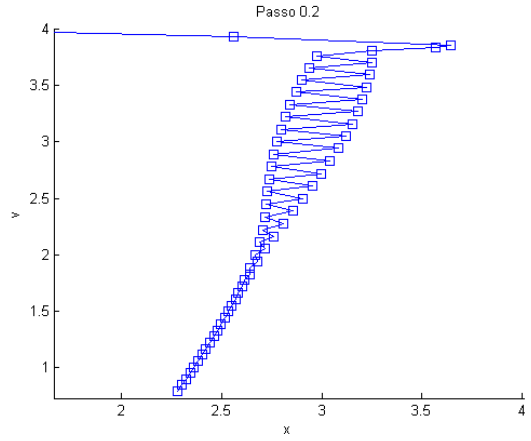


Figura 18: Algoritmo di Eulero su Van Der Pol

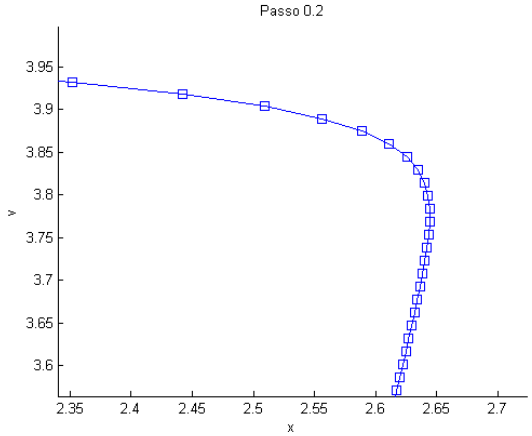


Figura 19: Algoritmo di Heun su Van Der Pol

Questo non avviene se si usano passi di integrazione molto piccoli in modo che la soluzione non abbia 'sovraelongazioni' troppo accentuate.

Appendice

In questa appendice sono raccolti i listati dei programmi e delle subroutine in Fortran e in MatLab.
Per il 'plottaggio' dei grafici per il controllo dell'errore si è utilizzato il linguaggio FORTRAN 95 in modo da poter decidere la precisione ed il motore grafico di MatLab.
Per il 'plottaggio' degli altri grafici è stato utilizzato interamente MatLab con il metodo di Heun e una routine che genera una serie di condizioni iniziali intorno a quelle definite dall'utente.
L'utente può decidere quante nuove condizioni iniziali saranno generate e quanto dovranno essere numericamente vicine alle sue.

MatLab

genInit.m e inputGen.m

Funzioni per l'input e la generazione di condizioni iniziali.
genInit.m

```
%Generazione array di condizioni iniziali intorno a quelle inserite dall'utente:
%A : distanza tra le cond.iniziali.
%B : numero di cond. iniziali generate per ogni condizione iniziale
%    inserita dall'utente.
%num: numero di cond. iniziali inserite dall'utente.
%x0,v0 : array di cond. iniziali dell'utente.
function [x1,v1,color,count] = genInit(A,B,num,x0,v0)
 %creo array colori
for i=1:6:num*B,
    color(i)='b';
    color(i+1)='g';
    color(i+2)='c';
    color(i+3)='m';
    color(i+4)='y';
    color(i+5)='k';

end
%creo array condizioni iniziali
count=1;
x1=[x0];
v1=[v0];
for j=1:num
    for i=1:B
        C=A*i;
        if (i==(B/2))
        x1(count)=x0(j);
        v1(count)=v0(j);
        else
         if(mod(i,6)==0)
            x1(count)=x0(j)+C;
            v1(count)=v0(j);
            x1(count+1)=-x0(j)+C;
            v1(count+1)=v0(j);
            x1(count+2)=-x0(j)+C;
            v1(count+2)=-v0(j);
            x1(count+3)=x0(j)+C;
            v1(count+3)=-v0(j);
         end
         if(mod(i,6)==1)
            x1(count)=x0(j)-C;
            v1(count)=v0(j);
            x1(count+1)=-x0(j)-C;
            v1(count+1)=v0(j);
            x1(count+2)=-x0(j)-C;
```

```
            v1(count+2)=-v0(j);
            x1(count+3)=x0(j)-C;
            v1(count+3)=-v0(j);
        end
        if(mod(i,6)==2)
            x1(count)=x0(j)+C;
            v1(count)=v0(j)+C;
            x1(count+1)=-x0(j)+C;
            v1(count+1)=v0(j)+C;
            x1(count+2)=-x0(j)+C;
            v1(count+2)=-v0(j)+C;
            x1(count+3)=x0(j)+C;
            v1(count+3)=-v0(j)+C;
        end
        if(mod(i,6)==3)
            x1(count)=x0(j)-C;
            v1(count)=v0(j)-C;
            x1(count+1)=-x0(j)-C;
            v1(count+1)=v0(j)-C;
            x1(count+2)=-x0(j)-C;
            v1(count+2)=-v0(j)-C;
            x1(count+3)=x0(j)-C;
            v1(count+3)=-v0(j)-C;
        end
        if(mod(i,6)==4)
            x1(count)=x0(j);
            v1(count)=v0(j)+C;
            x1(count+1)=-x0(j);
            v1(count+1)=v0(j)+C;
            x1(count+2)=-x0(j);
            v1(count+2)=-v0(j)+C;
            x1(count+3)=x0(j);
            v1(count+3)=-v0(j)+C;
        end
        if(mod(i,6)==5)
            x1(count)=x0(j);
            v1(count)=v0(j)-C;
            x1(count+1)=-x0(j);
            v1(count+1)=v0(j)-C;
            x1(count+2)=-x0(j);
            v1(count+2)=-v0(j)-C;
            x1(count+3)=x0(j);
            v1(count+3)=-v0(j)-C;
        end
      end
      count=count+4;
    end
end
end
```

inputGen.m

```
function [T,A,B,iteraz0,num]=inputGen()
    T= input ('Inserire il passo di discretizzazione:[0.05]');
    if isempty(T) T=0.05;
    end
    num= input ('Inserire il numero di condizioni iniziali:[1]');
    if isempty(num)num=1;
    end
    B = input('Inserire il numero di moti che saranno generati per ogni valore iniziale:[20]');
    if isempty(B)B=20;
    end
    A = input('Inserire la distanza tra un moto e l''altro[0.01]');
    if isempty(A)A=0.01;
    end
    iteraz0 = input('Inserire il numero di iterazioni:[2000]');
    if isempty(iteraz0)iteraz0=2000;
    end
end
```

heun.m e subroutines

Funzioni per il 'plottaggio' dei ritratti in fase dei vari sistemi trattati.
heun.m

```
%Laboratorio Calcolo Numerico A.A. 2005-2006
%Nicola Carlon
clear;
clc;
%user input
q=input('Selezionare il sistema: Pendolo semplice [1]
Pendolo con smorzamento [2]
Pendolo con molla [3] Van Der Pol [4].');
if isempty(q) q=1; end
if(q==1)sys='PENDOLO'
end
if(q==2)sys='PENDOLO CON SMORZAMENTO'
end
if(q==3)sys='PENDOLO CON MOLLA'
end
if(q==4)sys='VAN DER POL'
end
[T,A,B,iteraz0,num]=inputGen();
 for i=1:num
        x0(i) = input('Posizione iniziale x0:');
        v0(i) = input('Velocità iniziale v0:');

 end
 [x1,v1,color,count]=genInit(A,B,num,x0,v0);
 hold on
figure(1)
%PENDOLO
if(q==1)
    w=input('Inserire la pulsazione del sistema:[0.1]');
    if isempty(w)w=0.1;end
[iteraz]=pendolo(A,B,w,T,num,iteraz0,x0,v0,x1,v1,color)
end

%PENDOLO CON SMORZAMENTO
if(q==2)
    w=input('Inserire la pulsazione del sistema:[0.1]');
    if isempty(w)w=0.1;end
      m=input('Inserire coefficiente di smorzamento:[0.2]');
    if isempty(m)m=0.2;end
[iteraz]=pendolosm(A,B,w,m,T,num,iteraz0,x0,v0,x1,v1,color)
end

%PENDOLO CON MOLLA
if(q==3)
    w=input('Inserire la pulsazione del sistema:[0.1]');
    if isempty(w)w=0.1;end
      O=input('Inserire coefficiente elasticità/massa:[0.15]');
    if isempty(O)O=0.15;end
[iteraz]=pendolom(A,B,w,O,T,num,iteraz0,x0,v0,x1,v1,color)
end

%VAN DER POL
if(q==4)
    beta=input('Inserire il coefficiente beta:[0.1]');
    if isempty(beta)beta=0.1;end
[iteraz]=vander(A,B,beta,T,num,iteraz0,x0,v0,x1,v1,color)
end
```

pendolo.m

```
%Algoritmo di Heun: PENDOLO
%T: passo di discretizzazione
%A: 'distanza' tra una cond.iniz. e l'altra
%B: numero condizioni iniziali generate
%w: pulsazione del sistema
%num: numero di condizioni iniziali
```

```
%x0: array posizione iniziale utente
%v0: array velocità inizaile utente
%x1,v1: array cond.iniz.
%color: array colori
%iteraz0: numero iterazioni per Heun
function [iteraz] = pendolo(A,B,w,T,num,iteraz0,x0,v0,x1,v1,color)

%Moti utente
for j=1:num
xa=[x0(j)];
va=[v0(j)];
%modifico il numero di interazioni
if(v0(j)>x0(j))
    iteraz=iteraz0+abs(v0(j)/x0(j));
end
if(x0(j)>v0(j))
    iteraz=iteraz0+abs(x0(j)/v0(j));
end
%HEUN
for i=2:iteraz,
    xa(i) = xa(i-1) + T*(va(i-1)) ;
    va(i) = va(i-1) + T*(-(w.^2)*sin(xa(i-1)));
    t_x=xa(i);
    xa(i)=xa(i-1) + (T/2)*(va(i-1)+va(i));
    va(i)=va(i-1) + (T/2)*(-(w.^2)*sin(xa(i-1))-(w.^2)*sin(t_x));
end
%NUEH
plot(xa,va,'r')
end
%Moti intermedi
for j=1:num*B;
    x=[x1(j)];
    v=[v1(j)];
%genero i punti
if(v1(j)>x1(j))
    iteraz=iteraz0+abs(v1(j)/x1(j));
end
if(x1(j)>v1(j))
    iteraz=iteraz0+abs(x1(j)/v1(j));
end
%HEUN
for i=2:iteraz,
    x(i) = x(i-1) + T*(v(i-1)) ;
    v(i) = v(i-1) + T*(-(w.^2)*sin(x(i-1)));
    t_x=x(i);
    x(i)=x(i-1) + (T/2)*(v(i-1)+v(i));
    v(i)=v(i-1) + (T/2)*(-(w.^2)*sin(x(i-1))-(w.^2)*sin(t_x));
end
%NUEH
%Plotto i grafici
plot(x,v,color(j));
end
xlabel('x')
ylabel('v')
title(['Pendolo con w=',num2str(w)])
end
```

pendolosm.m

```
%Algoritmo di Heun: PENDOLO CON SMORZAMENTO
%T: passo di discretizzazione
%A: 'distanza' tra una cond.iniz. e l'altra
%B: numero condizioni iniziali generate
%w: pulsazione del sistema
%m: coefficiente del sistema
%num: numero di condizioni iniziali
%x0: array posizione iniziale utente
%v0: array velocità inizaile utente
%x1,v1: array cond.iniz.
%color: array colori
%iteraz0: numero iterazioni per Heun
function [iteraz] = pendolo(A,B,w,m,T,num,iteraz0,x0,v0,x1,v1,color)
%Moti utente
```

```
for j=1:num
xa=[x0(j)];
va=[v0(j)];
%modifico il numero di interazioni
if(v0(j)>x0(j))
    iteraz=iteraz0+abs(v0(j)/x0(j));
end
if(x0(j)>v0(j))
    iteraz=iteraz0+abs(x0(j)/v0(j));
end
for j=1:num
xa=[x0(j)];
va=[v0(j)];
if(v0(j)>x0(j))
    iteraz=iteraz0+abs(v0(j)/x0(j));
end
if(x0(j)>v0(j))
    iteraz=iteraz0+abs(x0(j)/v0(j));
end
%HEUN
for i=2:iteraz,
    xa(i) = xa(i-1) + T*(va(i-1)) ;
    va(i) = va(i-1) + T*(-(w.^2)*sin(xa(i-1))-2*m*va(i-1));
    t_x=xa(i);
    xa(i)=xa(i-1) + (T/2)*(va(i-1)+va(i));
    va(i)=va(i-1) + (T/2)*(-(w.^2)*sin(xa(i-1))-2*m*va(i-1)-(w.^2)*sin(t_x)-2*m*va(i-1));
end
%NUEH
plot(xa,va,'r')
end
%plotto i moti intermedi
for j=1:num*B;
    x=[x1(j)];
    v=[v1(j)];
%genero i punti
if(v1(j)>x1(j))
    iteraz=iteraz0+abs(v1(j)/x1(j));
end
if(x1(j)>v1(j))
    iteraz=iteraz0+abs(x1(j)/v1(j));
end
%HEUN
for i=2:iteraz,
    x(i) = x(i-1) + T*(v(i-1)) ;
    v(i) = v(i-1) + T*(-(w.^2)*sin(x(i-1))-2*m*v(i-1));
    t_x=x(i);
    x(i)=x(i-1) + (T/2)*(v(i-1)+v(i));
    v(i)=v(i-1) + (T/2)*(-(w.^2)*sin(x(i-1))-2*m*v(i-1)-(w.^2)*sin(t_x)-2*m*v(i-1));
end
%NUEH
%Plotto i grafici
plot(x,v,color(j));
end
xlabel('x')
ylabel('v')
title(['Pendolo con molla con w=',num2str(w),'m=',num2str(m)])
end
```

pendolom.m

```
%Algoritmo di Heun: PENDOLO CON MOLLA
%T: passo di discretizzazione
%A: 'distanza' tra una cond.iniz. e l'altra
%B: numero condizioni iniziali generate
%w: pulsazione del sistema
%O: coefficiente elasticità/massa
%num: numero di condizioni iniziali
%x0: array posizione iniziale utente
%v0: array velocità inizaile utente
%x1,v1: array cond.iniz.
%color: array colori
%iteraz0: numero iterazioni per Heun
function [iteraz] = pendolom(A,B,w,O,T,num,iteraz0,x0,v0,x1,v1,color)
```

```
%Moti utente
for j=1:num
xa=[x0(j)];
va=[v0(j)];
%modifico il numero di interazioni
if(v0(j)>x0(j))
    iteraz=iteraz0+abs(v0(j)/x0(j));
end
if(x0(j)>v0(j))
    iteraz=iteraz0+abs(x0(j)/v0(j));
end
for j=1:num
xa=[x0(j)];
va=[v0(j)];
if(v0(j)>x0(j))
    iteraz=iteraz0+abs(v0(j)/x0(j))
end
if(x0(j)>v0(j))
    iteraz=iteraz0+abs(x0(j)/v0(j))
end
%HEUN
for i=2:iteraz,
 xa(i) = xa(i-1) + T*(va(i-1)) ;
 va(i) = va(i-1) + T*(-(w.^2)*sin(xa(i-1))+0.^2*sin(xa(i-1))*cos(xa(i-1)));
 t_x=xa(i);
 xa(i)=xa(i-1) + (T/2)*(va(i-1)+va(i));
 va(i)=va(i-1) + (T/2)*(-(w.^2)*sin(xa(i-1))+0.^2*sin(xa(i-1))*cos(xa(i-1))-(w.^2)*sin(t_x)+0.^2*sin(t_x)*cos(t_x));
end
%NUEH
plot(xa,va,'r')
end
%plotto i moti intermedi
for j=1:num*B;
    x=[x1(j)];
    v=[v1(j)];
%genero i punti
if(v1(j)>x1(j))
    iteraz=iteraz0+abs(v1(j)/x1(j))
end
if(x1(j)>v1(j))
    iteraz=iteraz0+abs(x1(j)/v1(j))
end
%HEUN
for i=2:iteraz,
 x(i) = x(i-1) + T*(v(i-1)) ;
 v(i) = v(i-1) + T*(-(w.^2)*sin(x(i-1))+0.^2*sin(x(i-1))*cos(x(i-1)));
 t_x=x(i);
 x(i)=x(i-1) + (T/2)*(v(i-1)+v(i));
 v(i)=v(i-1) + (T/2)*(-(w.^2)*sin(x(i-1))+0.^2*sin(x(i-1))*cos(x(i-1))-(w.^2)*sin(t_x)+0.^2*sin(t_x)*cos(t_x));
end
%NUEH
%Plotto i grafici
plot(x,v,color(j));
end
xlabel('x')
ylabel('v')
title(['Pendolo con molla con w=',num2str(w),'k/m=',num2str(0)])
end
```

vander.m

```
%Algoritmo di Heun: VAN DER POL
%T: passo di discretizzazione
%A: 'distanza' tra una cond.iniz. e l'altra
%B: numero condizioni iniziali generate
%b: coefficiente beta del sistema
%num: numero di condizioni iniziali
%x0: array posizione iniziale utente
%v0: array velocità inizaile utente
%x1,v1: array cond.iniz.
%color: array colori
%iteraz0: numero iterazioni per Heun
function [iteraz] = pendolo(A,B,b,T,num,iteraz0,x0,v0,x1,v1,color)
```

```
%Moti utente
for j=1:num
xa=[x0(j)];
va=[v0(j)];
%modifico il numero di interazioni
if(v0(j)>x0(j))
    iteraz=iteraz0+abs(v0(j)/x0(j));
end
if(x0(j)>v0(j))
    iteraz=iteraz0+abs(x0(j)/v0(j));
end
for j=1:num
xa=[x0(j)];
va=[v0(j)];
if(v0(j)>x0(j))
    iteraz=iteraz0+abs(v0(j)/x0(j))
end
if(x0(j)>v0(j))
    iteraz=iteraz0+abs(x0(j)/v0(j))
end
%HEUN
for i=2:iteraz,
    xa(i) = xa(i-1) + T*(b*(va(i-1)-(1/3*xa(i-1)^3+xa(i-1)))) ;
    va(i) = va(i-1) + T*(-1/b*xa(i-1));
    t_x=xa(i);
    t_v=va(i);
    xa(i)=xa(i-1) + (T/2)*(b*(va(i-1)-1/3*xa(i-1)^3+xa(i-1))+b*(va(i)-1/3*t_x^3+t_x));
    va(i)=va(i-1) + (T/2)*(-1/b*xa(i-1)-1/b*t_x);
end
%NUEH
plot(xa,va,'r')
end
%plotto i moti intermedi
for j=1:num*B;
    x=[x1(j)];
    v=[v1(j)];
%genero i punti
if(v1(j)>x1(j))
    iteraz=iteraz0+abs(v1(j)/x1(j))
end
if(x1(j)>v1(j))
    iteraz=iteraz0+abs(x1(j)/v1(j))
end
%HEUN
for i=2:iteraz,
    x(i) = x(i-1) + T*(b*(v(i-1)-(1/3*x(i-1)^3+x(i-1)))) ;
    v(i) = v(i-1) + T*(-1/b*x(i-1));
    t_x=x(i);
    t_v=v(i);
    x(i)=x(i-1) + (T/2)*(b*(v(i-1)-1/3*x(i-1)^3+x(i-1))+b*(v(i)-1/3*t_x^3+t_x));
    v(i)=v(i-1) + (T/2)*(-1/b*x(i-1)-1/b*t_x);
end
%NUEH
%Plotto i grafici
plot(x,v,color(j));
end
xlabel('x')
ylabel('v')
title(['Van Der Pol con beta=', num2str(b)])
end
```

fort2mat.m

Programma per 'plottare' i grafici generati dal FORTRAN in MatLab.

```
%Interfaccia per FORTRAN
clear;
clc;
z=[1:1:50000];
load x.dat;
load v.dat;
hold on
```

```
figure(1)
plot(x,v,'b')
xlabel('x')
ylabel('v')
```

FORTRAN

ODE.f95

```
PROGRAM ODE
!Questo programma applica gli algoritmi di Eulero Cauchy,
!Heun, Runge Kutta classico (RK4) e Runge Kutta Gill
!per calcolare i moti di un sistema di due equazioni differenziali
!del primo ordine.
!I sistemi studiati sono:
!Pendolo semplice
!Pendolo smorzato
!Pendolo con molla
!Van Der Pol
!
!Input:
!       scelta del sistema
!       scelta dell'algoritmo
!       inserimento parametri
!Output:
!       x.dat
!       v.dat
IMPLICIT NONE

REAL :: w,T,m,O,b
REAL :: x0,v0
REAL :: punti
REAL, DIMENSION(50000) :: x,v
INTEGER :: q,a
x=0
w=0
WRITE(*,*) "Questo programma calcola numericamente"
WRITE(*,*) "i moti di alcuni sistemi di due equazioni differenziali"
WRITE(*,*) "studiati durante il corso di Fisica Matematica A.A. 2005-06"
WRITE(*,*) "utilizzando alcuni algoritmi di integrazione."
WRITE(*,*) ""
!Init
WRITE(*,*) "Selezionare il tipo di sistema:"
WRITE(*,*) ""
WRITE(*,*) "    Pendolo semplice [1]"
WRITE(*,*) "    Pendolo smorzato [2]"
WRITE(*,*) "    Pendolo con molla[3]"
WRITE(*,*) "    Van Der Pol      [4]"
WRITE(*,*) ""
READ(*,*)  q
CALL input(a,x0,v0,T,punti)
WRITE(*,*) ""
WRITE(*,*)  "Inserire i parametri del sistema:"
WRITE(*,*) ""
!PENDOLO
IF (q==1) THEN
WRITE(*,*)  "Pulsazione:"
READ(*,*) w
WRITE(*,*) ""
CALL pendolo(a,x0,v0,w,t,punti,x,v)
END IF

!PENDOLO CON SMORZAMENTO
IF (q==2) THEN
WRITE(*,*)  "Pulsazione:"
READ(*,*) w
WRITE(*,*)  "Coefficiente di smorzamento:"
READ(*,*) m
WRITE(*,*) ""
CALL pendolosm(a,x0,v0,w,m,t,punti,x,v)
END IF
```

```
!PENDOLO CON MOLLA
IF (q==3) THEN
WRITE(*,*)  "Pulsazione:"
READ(*,*) w
WRITE(*,*)  "Coefficiente di elasticità/massa:"
READ(*,*) O
WRITE(*,*) ""
CALL pendolom(a,x0,v0,w,O,t,punti,x,v)
END IF

!VAN DER POL
IF (q==4) THEN
WRITE(*,*)  "Parametro beta:"
READ(*,*) b
CALL vander(a,x0,v0,b,t,punti,x,v)
END IF

WRITE(*,*)  ""
WRITE(*,*)  "Creati i file x.dat e v.dat dei valori calcolati."
WRITE(*,*)  "La precisione è di 22 cifre decimali."
WRITE(*,*) ""
!Genero i file dat
 OPEN (UNIT=8, FILE='x.dat',STATUS='REPLACE')
 WRITE(8,100) x
 100 FORMAT (F27.22)
 OPEN (UNIT=9, FILE='v.dat',STATUS='REPLACE')
 WRITE(9,200) v
 200 FORMAT (F27.22)
END PROGRAM
!FINE PROGRAMMA PRINCIPALE

!INPUT
SUBROUTINE input(a,x0,v0,T,punti)
INTEGER, INTENT(OUT)::a
REAL, INTENT(OUT)::x0,v0,T,punti
WRITE(*,*) "Selezionare il tipo di algoritmo:"
WRITE(*,*) "Eulero Cauchy       [1]"
WRITE(*,*) "Heun                [2]"
WRITE(*,*) "Runge Kutta Classico[3]"
WRITE(*,*) "Runge Kutta Gill    [4]"
WRITE(*,*) ""
READ(*,*) a
WRITE(*,*) "Inserire parametri dell' algoritmo:"
WRITE(*,*) ""
WRITE(*,*)  "Passo di discretizzazione:"
READ(*,*) T
WRITE(*,*)  "Numero di iterazioni:[max=50000]"
READ(*,*) punti
IF (punti>50000) THEN
  WRITE(*,*) "Troppe iterazioni"
  STOP
END IF
WRITE(*,*) ""
WRITE(*,*) "Inserire le condizioni iniziali"
WRITE(*,*) "Posizione x0:"
READ(*,*) x0
WRITE(*,*) "Velocità v0 o Posizione y0:"
READ(*,*) v0
WRITE(*,*) ""
RETURN
END SUBROUTINE input

!VAN DER POL
SUBROUTINE vander (a,x0,v0,b,T,punti,x,v)
INTEGER, INTENT(IN) :: a
REAL, INTENT(IN) :: b,T,punti,x0,v0
REAL :: h1,h2,h3,h4,k1,k2,k3,k4
REAL, DIMENSION(50000), INTENT(OUT) :: x,v
x(1)=x0
v(1)=v0
WRITE(*,*) "Sistema Van Der Pol"
WRITE(*,*) "| x'=b(y-(x^3/3-x))"
WRITE(*,*) "| y'=1/b (x)"
```

```
WRITE(*,*) ""
IF(a==1) THEN
  WRITE(*,*) "Algoritmo di Eulero Cauchy"
   DO i=2,punti
    x(i) = x(i-1) + T*(b*(v(i-1)-(1./3.*x(i-1)**3+x(i-1))))
    v(i) = v(i-1) + T*(-1./b*x(i-1))
   END DO
END IF
IF(a==2) THEN
  WRITE(*,*) "Algoritmo di Heun"
  DO i=2,punti
    x(i) = x(i-1) + T*(b*(v(i-1)-(1./3.*x(i-1)**3+x(i-1))))
    v(i) = v(i-1) + T*(-1./b*x(i-1))
    t_x=x(i)
    x(i)=x(i-1) + (T/2.)*(b*(v(i-1)-1./3.*x(i-1)**3+x(i-1))+b*(v(i)-1./3.*t_x**3+t_x));
    v(i)=v(i-1) + (T/2.)*(-1./b*x(i-1)-1./b*t_x);
 END DO
END IF
IF(a==3) THEN
  WRITE(*,*) "Algoritmo di Runge Kutta Classico del quarto ordine"
  DO i=2,punti
    h1=T*(b*(v(i-1)-(1./3.*x(i-1)**3-x(i-1))))
    h2=T*(b*(v(i-1)+T/2.*h1-(1./3.*(x(i-1)+T/2.)**3-(x(i-1)+T/2.))))
    h3=T*(b*(v(i-1)+T/2.*h2-(1./3.*(x(i-1)+T/2.)**3-(x(i-1)+T/2.))) )
    h4=T*(b*(v(i-1)+T/2.*h3-(1./3.*x(i-1)**3-x(i-1))))
    x(i)=x(i-1)+T/6.*(h1+2.*h2+2.*h3+h4)
    k1 =T*( -1./b*x(i-1))
    k2 =T*( -1./b*(x(i-1)+T/2.))
    k3 = k2
    k4 =T*( -1./b*(x(i-1)+T))
    v(i)=v(i-1)+T/6.*(k1+2.*k2+2.*k3+k4)
 END DO
END IF
IF(a==4) THEN
 WRITE(*,*) "Algoritmo di Runge Kutta Gill"
 DO i=2,punti
    h1=T*(b*(v(i-1)-(1./3.*x(i-1)**3-x(i-1))))
    h2=T*(b*(v(i-1)+T/2.*h1-(1./3.*(x(i-1)+T/2.)**3-(x(i-1)+T/2.))))
    h3=T*b*(v(i-1)+T*(-1.+SQRT(2.))*h1+T*(1.-1./2.*SQRT(2.))*h2-(1./3.*(x(i-1)+T/2.)**3-(x(i-1)+T/2.)))
    h4=T*b*(v(i-1)-T/2*SQRT(2.)*h2+T*(1.+1./2.*SQRT(2.))*h3-(1./3.*(x(i-1)+T)**3-(x(i-1)+T)))
    x(i)=x(i-1)+T/6.*(h1+(2.-SQRT(2.))*h2+(2.-SQRT(2.))*h3+h4)
    k1 =T*( -1./b*x(i-1))
    k2 =T*( -1./b*(x(i-1)+T/2.))
    k3 = k2
    k4 =T*( -1./b*(x(i-1)+T))
    v(i)=v(i-1)+T/6.*(k1+(2.-SQRT(2.))*k2+(2.-SQRT(2.))*k3+k4)
 END DO
END IF
WRITE(*,*) ""
WRITE(*,*) "Parametri:"
WRITE(*,*) "x0=",x0," v0=",v0," b=",b," T=",T
RETURN
END SUBROUTINE vander

!PENDOLO CON MOLLA
SUBROUTINE pendolom (a,x0,v0,w,O,T,punti,x,v)
INTEGER, INTENT(IN) :: a
REAL, INTENT(IN) :: w,O,T,punti,x0,v0
REAL :: h1,h2,h3,h4,k1,k2,k3,k4
REAL, DIMENSION(50000), INTENT(OUT) :: x,v
INTEGER :: i
x(1)=x0
v(1)=v0
WRITE(*,*) "Sistema Pendolo con molla"
WRITE(*,*) "| x'=v"
WRITE(*,*) "| v'=-w^2sin(x)-O^2cos(x)sin(x)"
WRITE(*,*) ""
IF(a==1) THEN
  WRITE(*,*) "Algoritmo di Eulero Cauchy"
   DO i=2,punti
    x(i) = x(i-1) + T*(v(i-1))
    v(i) = v(i-1) + T*(-(w**2)*sin(x(i-1))+O**2*sin(x(i-1))*cos(x(i-1)))
   END DO
END IF
```

```
IF(a==2) THEN
  WRITE(*,*) "Algoritmo di Heun"
 DO i=2,punti
    x(i) = x(i-1) + T*(v(i-1))
    v(i) = v(i-1) + T*(-(w**2)*sin(x(i-1))+0**2*sin(x(i-1))*cos(x(i-1)))
    t_x=x(i)
    x(i)=x(i-1) + (T/2.)*(v(i-1)+v(i))
    v(i)=v(i-1) + (T/2.)*(-(w**2)*sin(x(i-1))+0**2*sin(x(i-1))*cos(x(i-1))-(w**2)*sin(t_x)+0**2*sin(t_x)*cos(t_x))
 END DO
END IF
IF(a==3) THEN
  WRITE(*,*) "Algoritmo di Runge Kutta Classico del quarto ordine"
  DO i=2,punti
    h1=T*v(i-1)
    h2=T*(v(i-1)+T/2.*h1)
    h3=T*(v(i-1)+T/2.*h2)
    h4=T*(v(i-1)+T*h3)
    x(i)=x(i-1)+T/6.*(h1+2.*h2+2.*h3+h4)
    k1 =T*( -(w**2)*SIN(x(i-1))+0**2*sin(x(i-1))*cos(x(i-1)))
    k2 =T*( -(w**2)*SIN(x(i-1)+T/2.)+0**2*sin(x(i-1)+T/2.)*cos(x(i-1)+T/2.))
    k3 = k2
    k4 =T*( -(w**2)*SIN(x(i-1)+T)+0**2*sin(x(i-1)+T)*cos(x(i-1)+T))
    v(i)=v(i-1)+T/6.*(k1+2.*k2+2.*k3+k4)
 END DO
END IF
IF(a==4) THEN
 WRITE(*,*) "Algoritmo di Runge Kutta Gill"
 DO i=2,punti
    h1=T*v(i-1)
    h2=T*(v(i-1)+T/2.*h1)
    h3=T*(v(i-1)+T*(-1.+SQRT(2.))*h1+T*(1.-1./2.*SQRT(2.))*h2)
    h4=T*(v(i-1)-T/2*SQRT(2.)*h2+T*(1.+1./2.*SQRT(2.))*h3 )
    x(i)=x(i-1)+T/6.*(h1+(2.-SQRT(2.))*h2+(2.-SQRT(2.))*h3+h4)
    k1 =T*( -(w**2)*SIN(x(i-1))+0**2*sin(x(i-1))*cos(x(i-1)))
    k2 =T*( -(w**2)*SIN(x(i-1)+T/2.)+0**2*sin(x(i-1)+T/2.)*cos(x(i-1)+T/2.))
    k3 = k2
    k4 =T*( -(w**2)*SIN(x(i-1))+0**2*sin(x(i-1))*cos(x(i-1)) )
    v(i)=v(i-1)+T/6.*(k1+(2.-SQRT(2.))*k2+(2.-SQRT(2.))*k3+k4)
 END DO
END IF
WRITE(*,*) ""
WRITE(*,*) "Parametri:"
WRITE(*,*) "x0=",x0," v0=",v0," w=",w," 0=",0," T=",T
RETURN
END SUBROUTINE pendolom

!PENDOLO CON SMORZAMENTO
SUBROUTINE pendolosm (a,x0,v0,w,m,T,punti,x,v)
INTEGER, INTENT(IN) :: a
REAL, INTENT(IN) :: w,m,T,punti,x0,v0
REAL :: k1,k2,k3,k4,h1,h2,h3,h4
REAL, DIMENSION(50000), INTENT(OUT) :: x,v
INTEGER :: i
x(1)=x0
v(1)=v0
WRITE(*,*) "Sistema Pendolo Smorzato"
WRITE(*,*) "| x'=v"
WRITE(*,*) "| v'=-w^2sin(x)-2mv"
WRITE(*,*) ""
IF(a==1) THEN
  WRITE(*,*) "Algoritmo di Eulero Cauchy"
   DO i=2,punti
    x(i) = x(i-1) + T*(v(i-1))
    v(i) = v(i-1) + T*(-(w**2)*sin(x(i-1))-2.*m*v(i-1))
   END DO
END IF
IF(a==2) THEN
  WRITE(*,*) "Algoritmo di Heun"
  DO i=2,punti
    x(i) = x(i-1) + T*(v(i-1))
    v(i) = v(i-1) + T*(-(w**2)*sin(x(i-1))-2.*m*v(i-1))
    t_x=x(i)
    x(i)=x(i-1) + (T/2.)*(v(i-1)+v(i))
    v(i)=v(i-1) + (T/2.)*(-(w**2)*SIN(x(i-1))-2.*m*v(i-1)-(w**2)*SIN(t_x)-2.*m*v(i))
```

```
 END DO
END IF
IF(a==3) THEN
  WRITE(*,*) "Algoritmo di Runge Kutta Classico del quarto ordine"
  DO i=2,punti
    h1=T*v(i-1)
    h2=T*(v(i-1)+T/2.*h1)
    h3=T*(v(i-1)+T/2.*h2)
    h4=T*(v(i-1)+T*h3)
    x(i)=x(i-1)+T/6.*(h1+2.*h2+2.*h3+h4)
    k1 =T*( -(w**2)*SIN(x(i-1))-2.*m*v(i-1))
    k2 =T*( -(w**2)*SIN(x(i-1)+T/2.)-2.*m*(v(i-1)+T/2*k1))
    k3 =T*( -(w**2)*SIN(x(i-1)+T/2.)-2.*m*(v(i-1)+T/2*k2))
    k4 =T*( -(w**2)*SIN(x(i-1)+T)-2.*m*(v(i-1)+T*k3))
    v(i)=v(i-1)+T/6.*(k1+2.*k2+2.*k3+k4)
 END DO
END IF
IF(a==4) THEN
 WRITE(*,*) "Algoritmo di Runge Kutta Gill"
 DO i=2,punti
    h1=T*v(i-1)
    h2=T*(v(i-1)+T/2.*h1)
    h3=T*(v(i-1)+T*(-1.+SQRT(2.))*h1+T*(1.-1./2.*SQRT(2.))*h2)
    h4=T*(v(i-1)-T/2.*SQRT(2.)*h2+T*(1.+1./2.*SQRT(2.))*h3)
    x(i)=x(i-1)+T/6.*(h1+(2.-SQRT(2.))*h2+(2.-SQRT(2.))*h3+h4)
    k1 =T*( -(w**2)*SIN(x(i-1))-2.*m*v(i-1))
    k2 =T*( -(w**2)*SIN(x(i-1)+T/2.)-2.*m*(v(i-1)+T/2.*k1))
    k3 =T*( -(w**2)*SIN(x(i-1)+T/2.)-2.*m*(v(i-1)+T/2.*k2))
    k4 =T*( -(w**2)*SIN(x(i))-2.*m*(v(i-1)+T*k3) )
    v(i)=v(i-1)+T/6.*(k1+(2.-SQRT(2.))*k2+(2.-SQRT(2.))*k3+k4)
 END DO
END IF
WRITE(*,*) ""
WRITE(*,*) "Parametri:"
WRITE(*,*) "x0=",x0," v0=",v0," w=",w," m=",m," T=",T
RETURN
END SUBROUTINE pendolosm

!PENDOLO
SUBROUTINE pendolo (a,x0,v0,w,T,punti,x,v)
INTEGER, INTENT(IN) :: a
REAL, INTENT(IN) :: w,T,punti,x0,v0
REAL :: k1,k2,k3,k4,h1,h2,h3,h4
REAL, DIMENSION(50000), INTENT(OUT) :: x,v
INTEGER :: i
x(1)=x0
v(1)=v0
WRITE(*,*) "Sistema Pendolo"
WRITE(*,*) "| x'=v"
WRITE(*,*) "| v'=-w^2sin(x)"
WRITE(*,*) ""
IF(a==1) THEN
  WRITE(*,*) "Algoritmo di Eulero Cauchy"

   DO i=2,punti
    x(i) = x(i-1) + T*(v(i-1))
    v(i) = v(i-1) + T*(-(w**2)*sin(x(i-1)))
   END DO
END IF
IF(a==2) THEN
  WRITE(*,*) "Algoritmo di Heun"
  DO i=2,punti
    x(i) = x(i-1) + T*(v(i-1))
    v(i) = v(i-1) + T*(-(w**2)*sin(x(i-1)))
    t_x=x(i)
    x(i)=x(i-1) + (T/2)*(v(i-1)+v(i))
    v(i)=v(i-1) + (T/2)*(-(w**2)*SIN(x(i-1))-(w**2)*SIN(t_x))
 END DO
END IF
IF(a==3) THEN
  WRITE(*,*) "Algoritmo di Runge Kutta Classico del quarto ordine"
  DO i=2,punti
    h1=T*v(i-1)
    h2=T*(v(i-1)+T/2.*h1)
```

```
    h3=T*(v(i-1)+T/2.*h2)
    h4=T*(v(i-1)+T*h3)
    x(i)=x(i-1)+T/6.*(h1+2.*h2+2.*h3+h4)
    k1 =T*( -(w**2)*SIN(x(i-1)))
    k2 =T*( -(w**2)*SIN(x(i-1)+T/2.))
    k3 = k2
    k4 =T*( -(w**2)*SIN(x(i-1)+T))
    v(i)=v(i-1)+T/6.*(k1+2.*k2+2.*k3+k4)
 END DO
END IF
IF(a==4) THEN
 WRITE(*,*) "Algoritmo di Runge Kutta Gill"
 DO i=2,punti
    h1=T*(v(i-1))
    h2=T*(v(i-1)+T/2.*h1)
    h3=T*(v(i-1)+T*(-1.+SQRT(2.))*h1+T*(1.-1./2.*SQRT(2.))*h2)
    h4=T*(v(i-1)-T/2.*SQRT(2.)*h2+T*(1.+1./2.*SQRT(2.))*h3)
    x(i)=x(i-1)+T/6.*(h1+(2.-SQRT(2.))*h2+(2.-SQRT(2.))*h3+h4)
    k1 = T*(-(w**2)*SIN(x(i-1)) )
    k2 =T*( -(w**2)*SIN(x(i-1)+T/2.))
    k3 = k2
    k4 = T*(-(w**2)*SIN(x(i-1)+T))
    v(i)=v(i-1)+T/6.*(k1+(2.-SQRT(2.))*k2+(2.-SQRT(2.))*k3+k4)
 END DO
END IF
WRITE(*,*) ""
WRITE(*,*) "Parametri:"
WRITE(*,*) "x0=",x0," v0=",v0," w=",w," T=",T
RETURN
END SUBROUTINE pendolo
```

Bibliografia

Libri:
Morandi Checchi, Introduzione al Calcolo Numerico, Editrice Esculapio, 2001.
Michela Redivo Zaglia, Calcolo Numerico: metodi ed algoritmi, Edizioni Libreria Progetto Padova, 2005.
G. Zilli, Lezioni di Calcolo Numerico, Edizione Libreria Progetto Padova, 2003.

Risorse Web:
http://www.dmmm.uniroma1.it/ iafrati/stdinfo/index.html
http://www.matematicamente.it/
http://mathworld.wolfram.com/topics/ODESolving.html

Programmi utilizzati:
MatLab 7 SP2
Chaoscope (www.chaoscope.org)
Kile LaTeX e WinEdit
Stalford FORTRAN compiler and Plato