Sistema **32.b** ***Muraca Giuseppe*** Matr. ***47553***

3° TESINA

MECCANISMI ATTIVATORI-INIBITORI

USO DEI METODI NUMERICI PER LA RISOLUZIONE DEI PROBLEMI FIN QUI AFFRONTATI

INDICE

PARTE PRIMA

Adesso iniziamo a lavorare sul sistema non lineare determinando numericamente i punti fissi del sistema cioè i punti che soddisfano:

$$\begin{cases} \dfrac{du}{dt} = a - u + u^2 v = f(u,v) = 0 \\ \dfrac{du}{dt} = b - u^2 v = g(u,v) = 0 \end{cases}$$

ed esaminando per semplicità i due casi esaminati nella 1° tesina.

Siccome il sistema non è lineare non possiamo applicare direttamente l'algoritmo di Gauss. Procediamo invece sviluppando, in serie di Taylor troncata al primo ordine, separatamente le due equazioni attorno ad un punto qualsiasi risolvendo il sistema risultante con l'algoritmo di Gauss per poi sviluppare attorno al nuovo punto trovato nuovamente il sistema in modo ciclico fino ad arrivare a trovare sempre lo stesso punto che è il punto che interessa noi. Questo metodo si rifà a quello di Newton per le equazioni non lineari.

Siccome in questa procedura abbiamo bisogno di risolvere ad ogni iterazione un sistema lineare definiamo una funzione 'Gauss' che presi in ingresso la matrice A dei coefficienti e il vettore colonna B dei termini noti restituisce il punto Z.

Ricordiamo che un sistema del tipo:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3 \end{cases}$$

può essere scritto in forma vettoriale come

$$AX = B$$

dove

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$B = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Il metodo di Gauss è un metodo che sfrutta alcune proprietà delle matrici dei coefficienti per triangolarizzare la matrice stessa. Sfrutta quindi le combinazione lineari tra le varie righe per azzerare i coefficienti che stanno al di sotto della diagonale principale.

Se vogliamo ad esempio azzerare l'elemento a(i,j) della nostra matrice scegliamo un valore 'pivot'

$$l_{i,j} = \frac{a_{i,j}}{a_{1,j}}$$

quindi facciamo una combinazione lineare tra la i-esima riga e la prima

$$\xrightarrow{\text{i-esima riga}} \begin{bmatrix} \dots & \dots & \dots & \dots & \dots \\ a(i-1,j) & \dots & \dots & \dots & \dots \\ a(i,j) - l_{i,j}a(1,j) & a(i,j+1) - l_{i,j}a(1,j) & \dots & \dots & \dots \\ a(i+1,j) & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix}$$

da cui $a(i,j) - l_{i,j}a(1,j) = a(i,j) - \frac{a(i,j)}{a(1,j)}a(1,j) = 0$

Sfruttando questo metodo annulliamo tutti gli elementi della prima colonna che non stanno sulla diagonale cioè gli a(i,1) con i=2….n. Fatto ciò escludiamo la 1° colonna e la 1° riga e consideriamo la sottomatrice restante. Possiamo ripetere il procedimento anche per questa nuova matrice e cosi via fino ad ottenere alla fine una matrice triangolare.
Una volta ottenuta la matrice triangolare diventa facile determinare le soluzioni del nostro sistema. Infatti sarà:

$$x_n = \frac{b(n)}{a(n,n)}$$

$$x_{n-1} = \frac{b(n-1) - \sum_{i=n-1}^{n} a(n-1,i)x_i}{a(n-1,n-1)}$$

….. e così via…

In generale:

$$x_k = \frac{b(k) - \sum_{i=k}^{n} a(k,i)x_i}{a(k,k)}$$

Dobbiamo fare attenzione però al caso in cui risultasse il nostro a(1,j)=0 (avremmo una divisione per 0). Se capita questo dobbiamo operare una permutazione delle righe.

L'algoritmo che fa ciò si trova nel file allegato gauss.m ed qui sotto riportato:

```
% FUNZIONE CHE DATA LA MATRICE DEI COEFFICENTI E IL VETTORE COLONNA
% DEI TERMINI NOTI TROVA GLI ZERI DI UN SISTEMA LINEARE

function Z=gauss(A,B)


r=size(B);
n=r(1);

for m=1:n
   for i=m+1:n

      if (A(m,m)==0);
         C=A(m,:);
         A(m,:)=A(i,:);
         A(i,:)=C;
         D=B(m);
         B(m)=B(i);
         B(i)=D
      end

   l(i,m)=A(i,m)/A(m,m);
   B(i)=B(i)-l(i,m)*B(m);

   for k=m:n
         A(i,k)=A(i,k)-l(i,m)*A(m,k);
      end
   end
end
%disp(A);
%disp(B);

Z(n)=B(n)/A(n,n);

for e=1:n-1

   Z2(n-e)=B(n-e);
   for k=n-e+1:n
      Z2(n-e)=(Z2(n-e)-A(n-e,k)*Z(k));
   end
 Z(n-e)=Z2(n-e)/A(n-e,n-e);
end
```

Caso a=1 b=1

Iniziamo adesso a trattare il caso (a=1 b=1) già trattato analiticamente nella 1° tesina ricordando che il risultato analitico era dato dal punto (2,1/4).

Ricordiamo che lo sviluppo di Taylor arrestato al primo ordine di una funzione di due variabili è:

$$f(u,v) = f(u_0,v_0) + \left[\frac{f'}{\partial u}(u_0,v_0)\right](u-u_0) + \left[\frac{f'}{\partial v}(u_0,v_0)\right](v-v_0)$$

il nostro sistema nel caso (a=1 b=1) è:

$$\begin{cases} \dot{u} = 1 - u + u^2 v = f(u,v) \\ \dot{v} = 1 - u^2 v = g(u,v) \end{cases}$$

scegliamo il punto (arbitrario) attorno al quale fare il primo sviluppo delle nostre equazioni *f(u,v)* e *g(u,v)* uguale a (1,1) . Chiamiamo x e y i vettori in cui memorizziamo via via rispettivamente l'elemento *u* e l'elemento *v* del punto trovato ad ogni iterazione. Quindi (per la scelta che abbiamo fatto) x(1)=1 e y(1)=1. Alla k-esima iterazione abbiamo che:

$$f(u,v) = f(x(k),y(k)) + \left[\frac{f}{\partial u}'(x(k),y(k))\right](u - x(k)) + \left[\frac{f}{\partial v}'(x(k),y(k))\right](v - y(k))$$

$$g(u,v) = g(x(k),y(k)) + \left[\frac{g}{\partial u}'(x(k),y(k))\right](u - x(k)) + \left[\frac{g}{\partial v}'(x(k),y(k))\right](v - y(k))$$

dove

$$\frac{f}{\partial u}'(x(k),y(k)) = -1 + 2x(k)y(k) \qquad \frac{f}{\partial v}'(x(k),y(k)) = x(k)^2$$

$$\frac{g}{\partial u}'(x(k),y(k)) = -2x(k)y(k) \qquad \frac{g}{\partial v}'(x(k),y(k)) = -x(k)^2$$

allora

$$f(u,v) = \underbrace{[-1 + 2x(k)y(k)]}_{\text{cofficiente di u}}u + \underbrace{[x(k)^2]}_{\text{coeff. di v}}v - \underbrace{\{1 - x(k) + x(k)^2 y(k) - [x(k)\cdot(-1 + 2x(k)y(k)] - [y(k)x(k)^2]\}}_{\text{termine noto}}$$

stesso procedimento per *g(u,v)*:

$$g(u,v) = \underbrace{[-2x(k)y(k)]}_{\text{cofficiente di u}}u + \underbrace{[-x(k)^2]}_{\text{coeff. di v}}v - \underbrace{\{1 - x(k)^2 y(k) - [x(k)\cdot(-2x(k)y(k)] - [y(k)(-x(k)^2)]\}}_{\text{termine noto}}$$

queste due equazioni formano un sistema lineare che si risolve con il metodo di Gauss, il punto risultante verrà poi utilizzato come punto di partenza nella iterazione successiva.

L'algoritmo che risolve il sistema non lineare usando il procedimento appena descritto è il seguente:

```
% TROVA GLI ZERI DEL SISTEMA NEL CASO a=1 b=1

function malara(n)

z(1,1)=1;
z(2,1)=1;

for k=1:n
   x(k)=z(1,k);
   y(k)=z(2,k);

   f(k)=1-x(k)+x(k)^2*y(k);
```

```
   fx(k)=-1+2*x(k)*y(k);
   fy(k)=x(k)^2;

   ftn(k)=-(f(k)-x(k)*fx(k)-y(k)*fy(k));
   fu(k)=fx(k);
   fv(k)=fy(k);

g(k)=1-x(k)^2*y(k);
   gx(k)=-2*x(k)*y(k);
   gy(k)=-x(k)^2;

   gtn(k)=-(g(k)-x(k)*gx(k)-y(k)*gy(k));
   gu(k)=gx(k);
   gv(k)=gy(k);

   A=[fu(k) fv(k);gu(k) gv(k)];
   B=[ftn(k);gtn(k)];
   E=gauss(A,B);
   z(1,k+1)=E(1);
   z(2,k+1)=E(2);
end
disp(z')
```

dove *n* rappresenta il numero di volte che vogliamo iterare il processo.
Come si può notare la funzione al suo interno fa uso della funzione gauss che abbiamo descritto sopra.

Questa funzione stampa una matrice (n x 2) che ha sulla prima riga il punto di partenza arbitrario, sull'ultima riga il punto cercato e tra queste due le successive approssimazioni.
Infatti se lanciamo la funzione 'malara(15)' avremo il seguente output:

```
   1.0000    1.0000
   2.0000   -1.0000
   2.0000    0.2500
   2.0000    0.2500
   2.0000    0.2500
   2.0000    0.2500
   2.0000    0.2500
   2.0000    0.2500
   2.0000    0.2500
   2.0000    0.2500
   2.0000    0.2500
   2.0000    0.2500
   2.0000    0.2500
   2.0000    0.2500
   2.0000    0.2500
   2.0000    0.2500
   2.0000    0.2500
```

si nota che già dopo la terza iterazione abbiamo trovato il punto fisso cercato.

Caso a=1 b=0

Vediamo, quindi, il secondo caso. Valgono naturalmente tutte le considerazioni fatte per il caso precedente e il procedimento è identico.

Per esso il risultato analitico era (1,0). Per poter calcolare questo punto dobbiamo ovviamente modificare (anche se di pochissimo) l'algoritmo:

```
% TROVA GLI ZERI DEL SISTEMA NEL CASO a=1 b=0

function malara2(n)

z(1,1)=1;
z(2,1)=1;

for k=1:n
   x(k)=z(1,k);
   y(k)=z(2,k);

   f(k)=1-x(k)+x(k)^2*y(k);
   fx(k)=-1+2*x(k)*y(k);
   fy(k)=x(k)^2;

   ftn(k)=-(f(k)-x(k)*fx(k)-y(k)*fy(k));
   fu(k)=fx(k);
   fv(k)=fy(k);

   g(k)=-x(k)^2*y(k);
   gx(k)=-2*x(k)*y(k);
   gy(k)=-x(k)^2;

   gtn(k)=-(g(k)-x(k)*gx(k)-y(k)*gy(k));
   gu(k)=gx(k);
   gv(k)=gy(k);

   A=[fu(k) fv(k);gu(k) gv(k)];
   B=[ftn(k);gtn(k)];
   E=gauss(A,B);
   z(1,k+1)=E(1);
   z(2,k+1)=E(2);
end
disp(z')
```

Punto iniziale arbitrario

riga modificata

(Si potrebbe anche facilmente modificare l'algoritmo in modo che prenda in input i valori di a,b questo al momento non è però un problema al centro della mia attenzione).

L'output in questo caso è il seguente:

```
   1    1
   1    0
   1    0
   1    0
   1    0
   1    0
   1    0
   1    0
   1    0
   1    0
   1    0
   1    0
   1    0
```

1 0
1 0
1 0
1 0

qui il punto viene trovato alla seconda iterazione.

L'approssimazione (perché sempre di approssimazione si tratta) data dal calcolo numerico si è dimostrata (almeno in questi casi esaminati) buona.

PARTE SECONDA

Sappiamo che se abbiamo una matrice R con $\det(R) \neq 0$, facendo $R^T AR=B$ (trasformazione di similitudine) ottengo la matrice B che ha gli stessi autovalori di A . R in questo caso viene chiamata matrice di cambiamento di base.

In questa seconda parte applichiamo un metodo numerico per la ricerca degli autovalori e degli autovettori del sistema linearizzato. Il metodo applicato è quello di JACOBI. Una restrizione però(non di poco conto), è che la matrice dei coefficienti deve essere una matrice simmetrica. Siccome non sono riuscito a trovare per il nostro sistema in esame valori dei coefficienti *a,b* tali che il sistema linearizzato potesse dare una matrice dei coefficienti simmetrica, prendo in esame due matrici arbitrarie e applico per esse il metodo di Jacobi.

Si può però già presentare l'algoritmo poiché esso è generalizzato e vale per ogni matrice simmetrica. Esso mira a diagonalizzare la matrice dei coefficienti. In tal modo gli elementi restanti sulla diagonale principale saranno proprio gli autovalori.

La procedura usata è la seguente:

- Per prima cosa controlliamo che la matrice immessa sia simmetrica per evitare che l'algoritmo stampi risultati errati.
- Poi si cercano gli indici del valore più grande tra quelli fuori dalla diagonale principale e li chiamiamo p,q.
- Verifichiamo che non si verifichi il caso particolare in cui *a(p,p)=a(q,q)*, e in questo caso assegniamo direttamente a θ il valore π/4.
- Se non si cade nel caso particolare calcoliamo per il metodo di jacobi senθ e cosθ
- Calcoliamo la matrice R di cambiamento di base e la moltiplichiamo per la matrice R dell'iterazione precedente.
- Facciamo le opporune combinazioni lineari (rotazioni) usando senθ e cosθ calcolati sopra

L'algoritmo risultante è il seguente:

```
%FUNZIONE CHE PRENDE IN INPUT UNA MATRICE SIMMETRICA


function Jacobi(A)

e=0.000000000001;
s=1;
r=size(A);
```

```
n=r(1);
R=eye(n,n);


% CONTROLLIAMO CHE LA MATRICE IN INPUT SIA SIMMETRICA
for i=1:n
      for k=i:n
         if (A(i,k)~=A(k,i))
            A=input('La matrice in input non è simmetrica')
            return
         end
      end
   end

while (s>e)
   c=0;

   %  CERCHIAMO GLI INDICI DEL VALORE MASSIMO FUORI DIAGONALE
   for i=1:n-1
      for k=i+1:n
         if (abs(A(i,k))>abs(c))
            c=A(i,k);
            p=i;
            q=k;
         end
      end
   end
   %VERIFICHIAMO CHE NON SI VERIFICHI IL CASO PARTICOLARE a(q,q)=a(p,p)
   %E TROVIAMO T, Sen(teta), cos(teta)
   if (A(p,p)==A(q,q))
      st=1/sqrt(2);
      ct=1/sqrt(2);
   else
   T=(2*A(p,q))/(A(p,p)-A(q,q));
   if (T==0)
      st=1/sqrt(2);
      ct=1/sqrt(2);
   else
      t=(-1+sqrt(1+T^2))/T;
      st=t/sqrt(1+t^2);
      ct=1/sqrt(1+t^2);
   end
end
%CALCOLIAMO IL PRODOTTO DELLE MATRICI Rn CHE CI DA UNA MATRICE CHE HA PER
COLONNE
% GLI AUTOVETTORI
B=eye(n,n);
B(p,p)=ct;
B(q,q)=ct;
B(p,q)=st;
B(q,p)=-st;
R=R*B;
%TRASFORMIAMO LA MATRICE ATTUANDO LA ROTAZIONE DI ANGOLO TETA
for i=1:n
   for j=1:n

        if (and(i~=p,i~=q))
           A1(i,j)=A(i,j);
        else

         A1(p,j)=A(p,j)*ct+A(q,j)*st;
         A1(q,j)=A(q,j)*ct-A(p,j)*st;
```

```
      end
   end
end
for i=1:n
   for j=1:n
      if (and(j~=p,j~=q))
           A2(i,j)=A1(i,j);
else
         A2(i,p)=A1(i,p)*ct+A1(i,q)*st;
         A2(i,q)=A1(i,q)*ct-A1(i,p)*st;
         end
      end
   end


   A=A2;

   s=0;
   for i=1:n-1
     for k=i+1:n
        s=s+A(i,k)^2;
     end
  end
end
%STAMPA I RISULTATI
risultato=A
autovettori=R
```

Esaminiamo adesso due casi particolari.

Caso 1

La prima matrice (2 x 2) scelta è:

$$A = \begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix}$$

da cui:

$$\det(A - lI) = \begin{vmatrix} 1-l & 4 \\ 4 & 1-l \end{vmatrix} = (1-l)^2 - 16 = l^2 - 2l - 15$$

$$l = 1 \pm \sqrt{1+15} = 1 \pm 4$$

$$\begin{cases} l_1 = 5 \\ l_2 = -3 \end{cases}$$

$$(A - 5I) = \begin{bmatrix} -4 & 4 \\ 4 & -4 \end{bmatrix}$$

quindi un primo autovetture è $v = \begin{bmatrix} a \\ -a \end{bmatrix}$ dove α può assumere qualsiasi valore.

$$(A+3I)=\begin{bmatrix}4 & 4\\ 4 & 4\end{bmatrix}$$

quindi un secondo autovetture è $v=\begin{bmatrix}a\\ a\end{bmatrix}$ dove α può assumere qualsiasi valore.

Se eseguiamo l'algoritmo inserendo in input la nostra A otterremo il seguente output:

```
» Jacobi(A)

risultato =

    5.0000         0
         0   -3.0000


autovettori =

    0.7071    0.7071
   -0.7071    0.7071
```

Il risultato analitico e quello numerico coincidono!

Caso 2

Vediamo adesso un secondo caso:

$$A=\begin{bmatrix}2 & 3\\ 3 & 2\end{bmatrix}$$

da cui:

$$\det(A-lI)=\begin{vmatrix}2-l & 3\\ 3 & 2-l\end{vmatrix}=(2-l)^2-9=l^2-4l-5$$

$$l=2\pm\sqrt{4+5}=2\pm 3$$

$$\begin{cases}l_1=5\\ l_2=-1\end{cases}$$

$$(A-5I)=\begin{bmatrix}-3 & 3\\ 3 & -3\end{bmatrix}$$

quindi un primo autovetture è $v=\begin{bmatrix}a\\ -a\end{bmatrix}$ dove α può assumere qualsiasi valore.

$$(A+2I)=\begin{bmatrix}3 & 3\\ 3 & 3\end{bmatrix}$$

quindi un secondo autovetture è $v=\begin{bmatrix}a\\ a\end{bmatrix}$ dove α può assumere qualsiasi valore.

Se eseguiamo l'algoritmo inserendo in input la nostra A otterremo il seguente output:

Jacobi(A)

risultato =

```
    5.0000         0
         0   -1.0000
```

autovettori =

```
    0.7071    0.7071
   -0.7071    0.7071
```

PARTE TERZA

In questa ultima parte usiamo uno dei metodi di Runge-Kutta del 3° ordine per risolvere numericamente il nostro sistema differenziale non lineare.

$$\begin{cases} \dot{u} = a - u + u^2 v = f(u,v) \\ \dot{v} = b - u^2 v = g(u,v) \end{cases}$$

Il metodo che utilizziamo è il metodo di Heun (un miglioramento del metodo di Eulero) dove ogni punto y_{k+1} delle funzioni risultanti viene trovato con il seguente procedimento:

$$u_{k+1} = u_k + \frac{h}{4}(f1 + 3f3)$$

$$v_{k+1} = v_k + \frac{h}{4}(g1 + 3g3)$$

dove:

$$f1 = f(t_k, u_k, v_k)$$

$$g1 = g(t_k, u_k, v_k)$$

$$f2 = f\left(t_k + \frac{h}{3}, u_k + \frac{h}{3} f1, v_k + \frac{h}{3} g1\right)$$

$$g2 = g\left(t_k + \frac{h}{3}, u_k + \frac{h}{3} f1, v_k + \frac{h}{3} g1\right)$$

$$f3 = f\left(t_k + \frac{2h}{3}, u_k + \frac{2h}{3} f2, v_k + \frac{2h}{3} g2\right)$$

$$g3 = f\left(t_k + \frac{2h}{3}, u_k + \frac{2h}{3} f2, v_k + \frac{2h}{3} g2\right)$$

calcolando ciclicamente questi valori costruiremo una approssimazione delle curve che rappresentano la soluzione del sistema.

L'algoritmo elaborato si trova nel file allegato 'RungeK.m' e una volta mandato in esecuzione chiede in input oltre ai dati iniziali e l'intervallo di integrazione, anche i valori dei parametri *a,b*

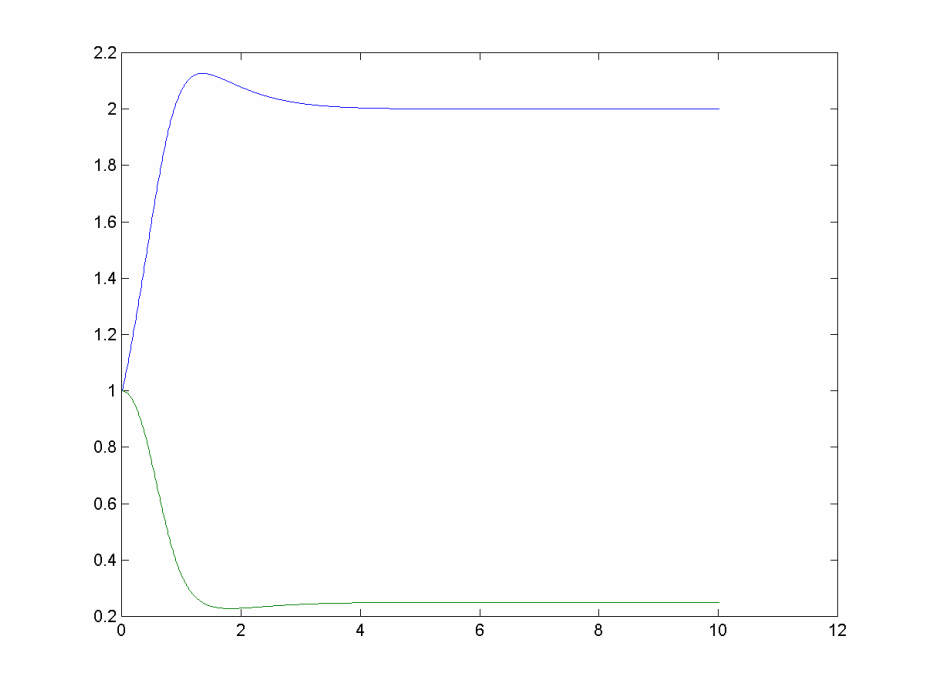
```
function rungeK

clc;
h=0.01;
a=input('Parametro a=');
b=input('Parametro b=');
ti=input('Inserisci istante t iniziale -->');
tf=input('Inserisci istante t finale -->');
p=(tf-ti)/h;
t0=0;
u0=input('Inserisci u(0)=');
v0=input('Inserisci v(0)=');


u(1)=u0;
v(1)=v0;
t(1)=t0;

for k=1:p
   t(k+1)=t0+(k+1)*h;

   f1=a-u(k)+(u(k)^2)*v(k);
   g1=b-(u(k)^2)*v(k);

   f2=a-u(k)+(h/3)*f1+((u(k)+(h/3)*f1)^2)*(v(k)+(h/3)*g1);
   g2=b-((u(k)+(h/3)*f1)^2)*(v(k)+(h/3)*g1);


   f3=a-(u(k)+(2*h/3)*f2)+((u(k)+(2*h/3)*f2)^2)*(v(k)+(2*h/3)*g2);
   g3=b-((u(k)+(2*h/3)*f2)^2)*(v(k)+(2*h/3)*g2);

   u(k+1)=u(k)+(h/4)*(f1+3*f3);
   v(k+1)=v(k)+(h/4)*(g1+3*g3);

end


plot(t,u,t,v);
```

Esaminiamo adesso i soliti due casi già visti: (a=1, b=1) e (a=1, b=0).

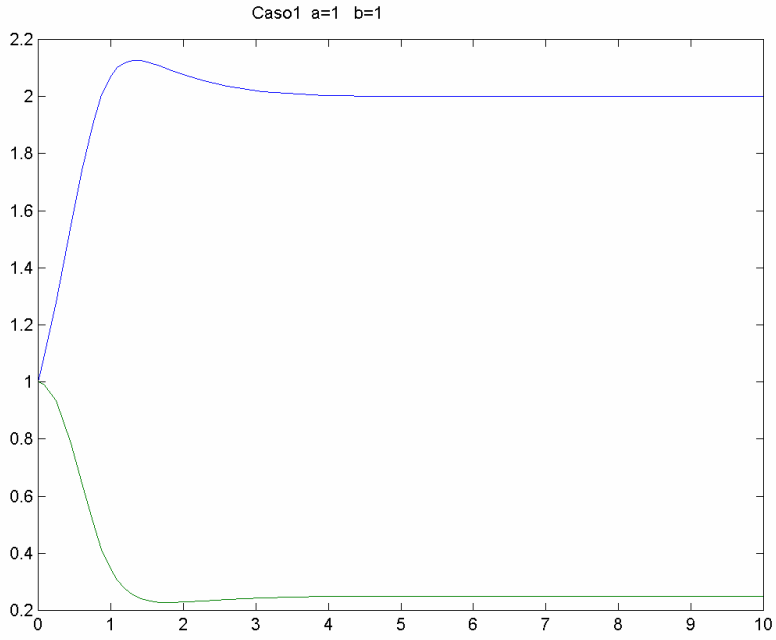
Se richiamiamo questa funzione e diamo in input questi parametri:

Parametro a=1
Parametro b=1
Inserisci istante t iniziale -->0
Inserisci istante t finale -->10
Inserisci u(0)=1
Inserisci v(0)=1

Otteniamo questo grafico:



che possiamo confrontare con la soluzione (anch'essa numerica) ottenuta attraverso il comando ODE23 di Matlab



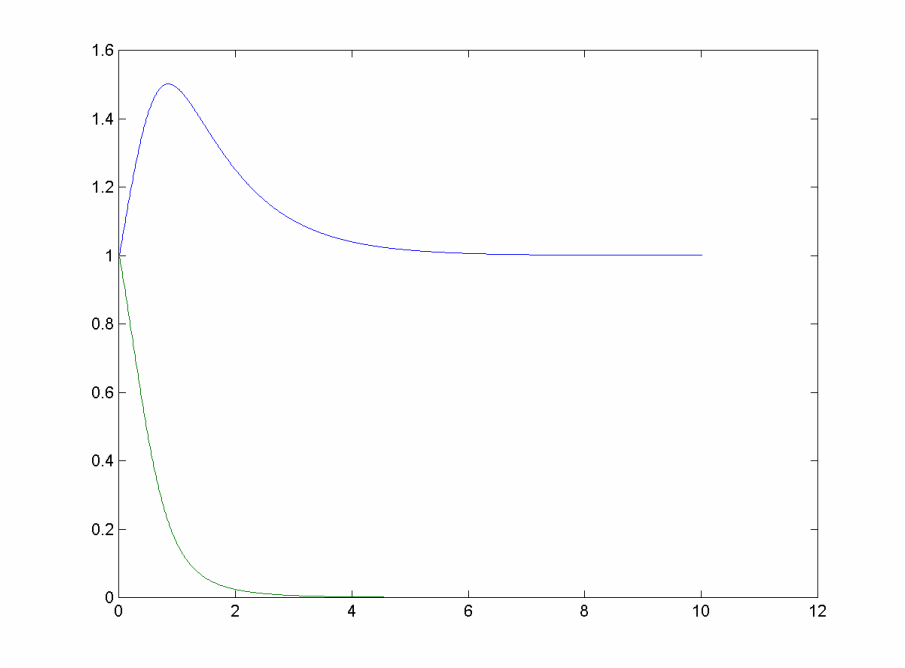
le due curve si comportano allo stesso modo.

Vediamo adesso l'altro caso e inseriamo in input alla nostra funzione

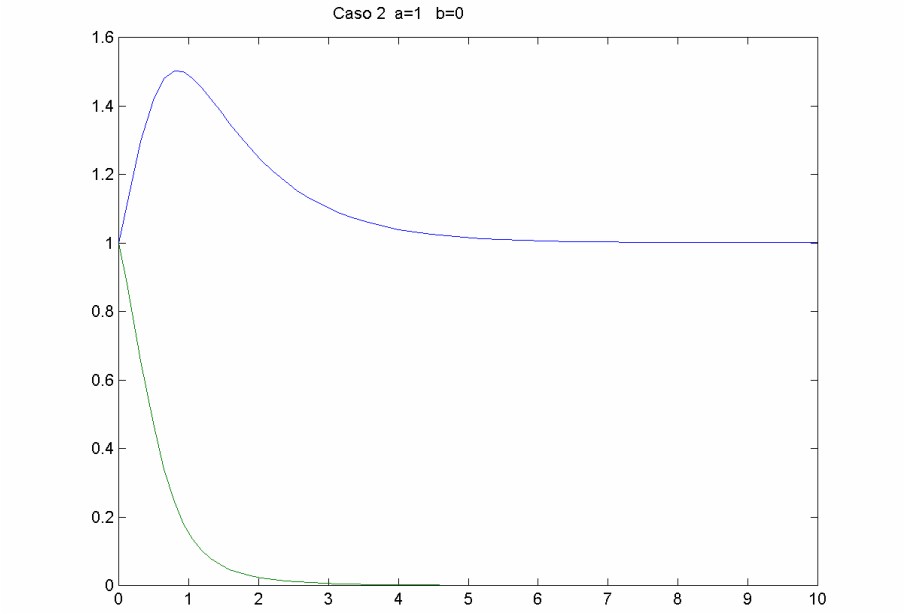
Parametro a=1
Parametro b=0
Inserisci istante t iniziale -->0
Inserisci istante t finale -->10
Inserisci u(0)=1

Inserisci v(0)=1

Otteniamo il seguente grafico:



che anche in questo caso confrontiamo con quello ottenuto con l'ODE23:



e anche in questo caso le due curve si comportano allo stesso modo.