

## Simulare con Matcont

### 1 Modello di Lorenz (1963) <sup>(1)</sup>

Sia dato il sistema di equazioni differenziali

$$\begin{aligned}\dot{x} &= \sigma(y - x) \\ \dot{y} &= \rho x - xz - y \\ \dot{z} &= xy - \beta z\end{aligned}$$

dove  $\sigma = 10$ ,  $\rho = 28$  e  $\beta = 7/3$ .

Si simuli con Matcont l'andamento del sistema a partire dalla condizione iniziale  $x(0) = 0$ ,  $y(0) = 1$  e  $z(0) = 0$ .

Si verifichi la dipendenza sensibile alle condizioni iniziali del comportamento ottenuto.

Si caratterizzi infine l'attrattore mediante una sua sezione di Poincaré e se ne calcoli il suo primo esponente di Liapunov.

NOTA: La classica forma a "farfalla" dell'attrattore di Lorenz può essere visualizzata nello spazio  $(w_1, w_2)$  dove  $w_1 = x - y$  e  $w_2 = z$ .

### 2 Modello di Lorenz (1984) <sup>(2)</sup>

Sia dato il sistema di equazioni differenziali

$$\begin{aligned}\dot{x} &= y^2 + z^2 + ax - aF \\ \dot{y} &= -xy + bxz + y - G \\ \dot{z} &= -bxy - xz + z\end{aligned}$$

dove  $a = 0.25$ ,  $b = 4$ ,  $F = 2$  e  $G = 1.67$ .

Si simuli con Matcont l'andamento del sistema a partire dalla condizione iniziale  $x(0) = 1$ ,  $y(0) = 0$  e  $z(0) = 0.5$ . Si caratterizzi infine l'attrattore mediante una sua sezione di Poincaré e se ne calcoli il suo primo esponente di Liapunov.

### 3 Modello preda-predatore-superpredatore

Sia dato il sistema di equazioni differenziali che caratterizza la dinamica di una catena alimentare a tre livelli trofici

$$\begin{aligned}\dot{x}_1 &= rx_1 \left(1 - \frac{x_1}{K}\right) - \frac{a_1 x_1}{b_1 + x_1} x_2 \\ \dot{x}_2 &= e_2 \frac{a_1 x_1}{b_1 + x_1} x_2 - \frac{a_2 x_2}{b_2 + x_2} x_3 - m_2 x_2 \\ \dot{x}_3 &= e_3 \frac{a_2 x_2}{b_2 + x_2} x_3 - m_3 x_3\end{aligned}$$

dove  $x_1$  è la densità di prede,  $x_2$  la densità di predatori e  $x_3$  la densità di superpredatori.

Si assumano i seguenti valori per i parametri:

$$\begin{array}{llll}r = 1.5 & K = 1 & & \\a_1 = 7/3 & b_1 = 1/3 & a_2 = 0.05 & b_2 = 0.5 \\m_2 = 0.4 & m_3 = 0.01 & e_2 = 1 & e_3 = 1\end{array}$$

Utilizzando Matcont si simuli l'andamento del sistema. Si noti come l'attrattore sia di tipo caotico: lo si caratterizzi quindi mediante una sua sezione di Poincaré e se ne calcoli il suo primo esponente di Liapunov.

Come diventa la dinamica del sistema se  $b_2 = 0.46$  ?

Determinare infine set parametrici in corrispondenza dei quali il sistema tende verso la coesistenza ciclica o stazionaria delle tre specie, la coesistenza ciclica o stazionaria di preda e predatore con estinzione del superpredatore, la presenza della sola preda con estinzione sia del predatore che del superpredatore.

---

#### NOTE

Per mostrare le sezioni di Poincaré e per calcolare gli esponenti di Liapunov si faccia riferimento ai file allegati all'esercitazione.

<sup>(1)</sup> Il modello di Lorenz nasce nel 1963 da alcune semplificazioni delle equazioni di Navier-Stokes sulla descrizione del comportamento dinamico di uno strato di fluido che presenta moti convettivi a causa di una differenza di temperatura.

<sup>(2)</sup> Il modello di Lorenz formulato nel 1984 è un modello semplificato di circolazione atmosferica globale.

# TRACCE DELLE SOLUZIONI

## 1 Modello di Lorenz (63)

Con **Select** → **System** → **New** apriamo la finestra necessaria per l'inserimento del modello. Questa va così compilata

System

Name: lorenz\_model

Coordinates: x,y,z

Parameters: sigma,rho,beta

Time: t

Derivatives: 1st ord 2nd ord 3rd ord 4th ord 5th ord

- numerically ☐ ☐ ☐ ☐ ☐

- from window ☐ ☐ ☐ ☐ ☐

- symbolically ☒ ☒ ☒ ☒ ☒

$x' = \sigma \cdot (y - x)$   
 $y' = \rho \cdot x - x \cdot z$   
 $z' = x \cdot y - \beta \cdot z$

OK Cancel

NOTA: si chiede a MatCont di calcolare simbolicamente le derivate del primo ordine (Jacobiano) perché queste verranno poi utilizzate per la determinazione degli esponenti di Lyapunov.

Con **Type** → **Initial point** → **Point** definiamo le condizioni iniziali e i valori dei parametri del sistema. Si aprono due finestre (**Starter** e **Integrator**) che compiliamo come segue:

Starter

Initial Point

t: 0

x: 0

y: 1

z: 0

sigma: 10

rho: 28

beta: 2.33333

Select Cycle

Integrator

Integration data

Method: ode45

Interval: 200

InitStepsize: <automatic>

MaxStepsize: <automatic>

Rel. Tolerance: 0.0001

Abs. Tolerance: 1e-6

Refine: 1

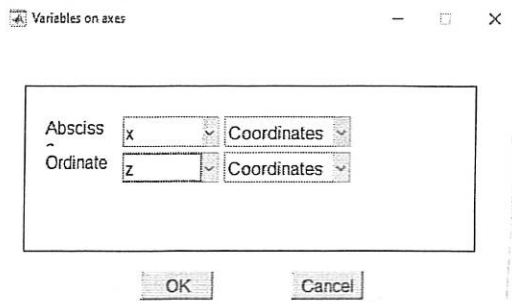
Normcontrol: No

NOTA: La finestra MatCont riporta - Point Type P ⇒ siamo pronti per tracciare una traiettoria  
- Curve Type 0 (0 = orbit)

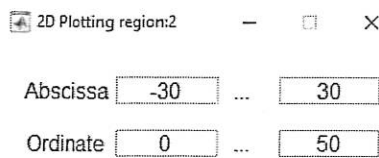
Per visualizzare la traiettoria (per ora la sua proiezione nello spazio  $(x, z)$ ) apriamo una finestra grafica con

Window  $\rightarrow$  Graphic  $\rightarrow$  2Dplot

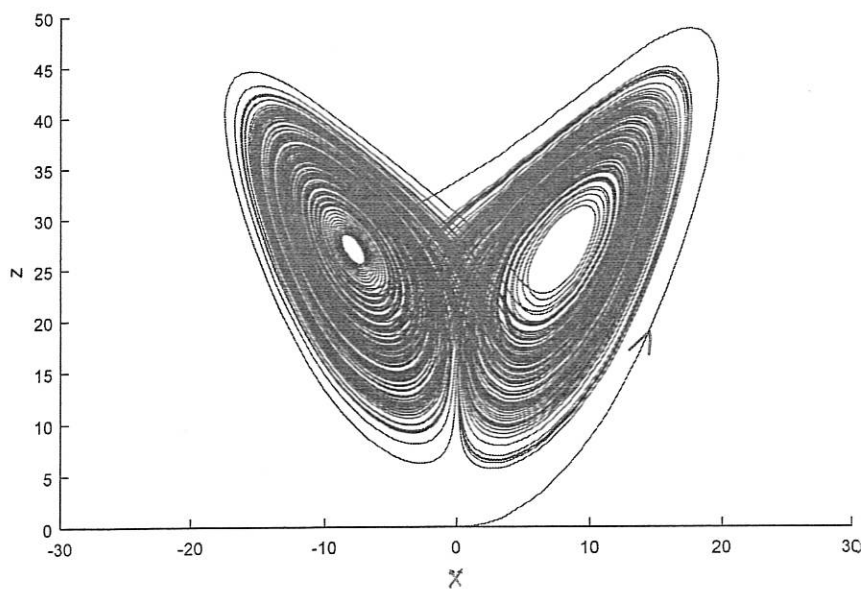
Scegliamo  $x$  e  $z$  come variabili da visualizzare sugli assi:



Con Layout  $\rightarrow$  Plotting region fissiamo gli estremi sugli assi:

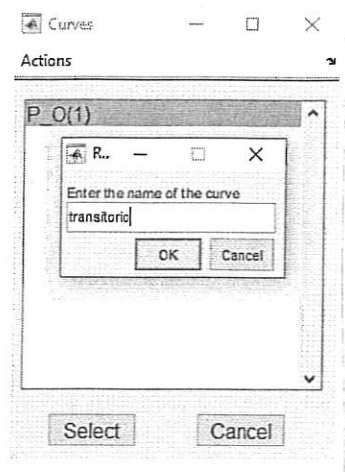


Compute  $\rightarrow$  Forward esegue la simulazione del sistema. La traiettoria converge verso il noto attrattore di Lorenz.



La simulazione viene memorizzata nella curva  $P_0(1)$   
(finestra MatCont  $\rightarrow$  Curve).

Con  $\text{select} \rightarrow \text{Curve} \rightarrow \text{Actions} \rightarrow \text{Rename}$  rinominiamo la curva in "transitorio".



Questa traiettoria mostrava anche il transitorio verso l'attrattore. Per determinare quest'ultimo, finiamo ora come nuova condizione iniziale l'ultimo punto della curva "transitorio".

$\text{select} \rightarrow \text{Initial point} \rightarrow \text{---This is the last---} \rightarrow \text{Select}$

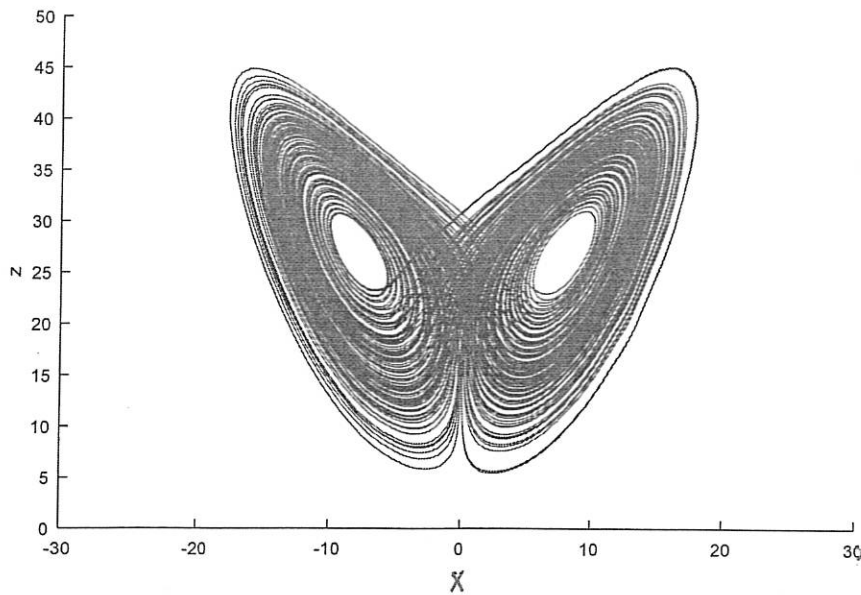


	Initial Point
t	0
x	-13.679983
y	-11.086647
z	36.959969
sigma	10
rho	28
beta	2.33333

Nota: Dopo aver selezionato la nuova condizione iniziale, nella finestra Starter (aggiornata ai nuovi valori) fissare  $t$  al valore 0

Con  $\text{Plot} \rightarrow \text{Clear}$  puliamo la finestra grafica

$\text{Compute} \rightarrow \text{Forward}$  eseguiamo la simulazione dell'attrattore di Lorenz (questa volta senza il transitorio)



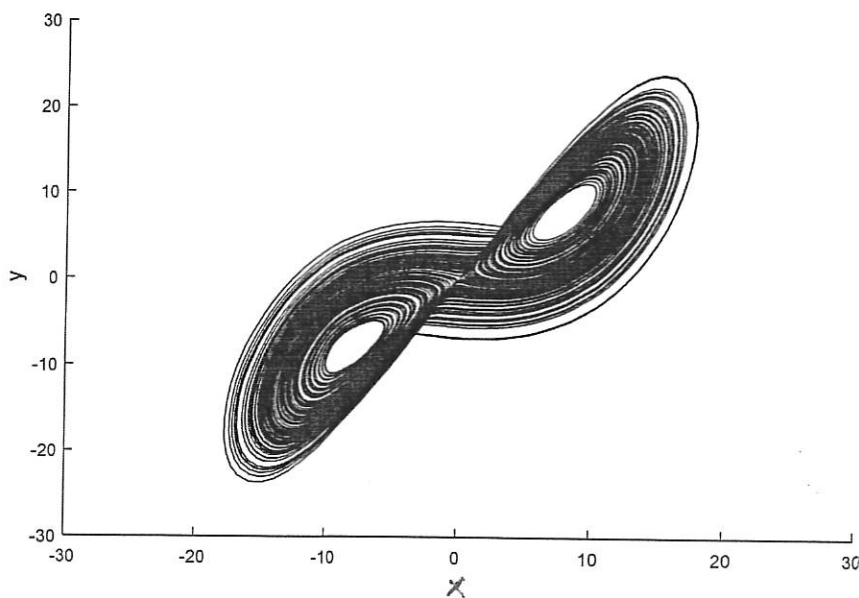
Attrattore di Lorenz

Con Select → Curve → Actions → Rename rinominiamo la curva P\_0(1) in "attrattore"

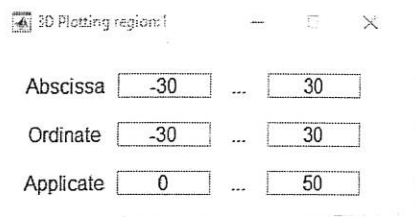
↳ l'ultima curva calcolata  
Possiamo visualizzare l'attrattore anche in altri piani o nello spazio 3D  
Con Layout → Variables on axes possiamo scegliere  $x$  e  $y$

Con Layout → Plotting region finiamo gli estremi degli assi:  
a -30 30 Abscissa  
-30 30 Ordinate

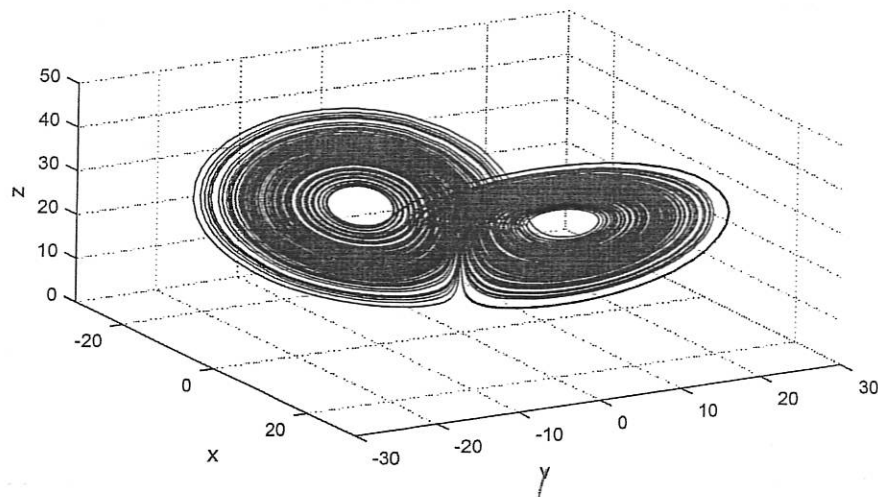
Plot → Redraw curve disegna l'attrattore nello spazio  $(x, y)$



Per visualizzare l'attrattore nello spazio di stato  $(x_1, x_2, x_3)$ , apriamo una nuova finestra grafica con  
 Window  $\rightarrow$  Graph  $\rightarrow$  3D plot  
 con Layout  $\rightarrow$  Plotting region fissiamo gli estremi come in figura



I comandi  $\gg$  grid digitati in Matlab permettono la  
 $\gg$  view(60,35)  
 visualizzazione dell'attrattore nello spazio di stato



Chiusiamo la finestra 3D.

Visualizziamo la serie temporale dell'attrattore

Layout  $\rightarrow$  variables on axes  $\rightarrow$  selezioniamo  $t \times x$

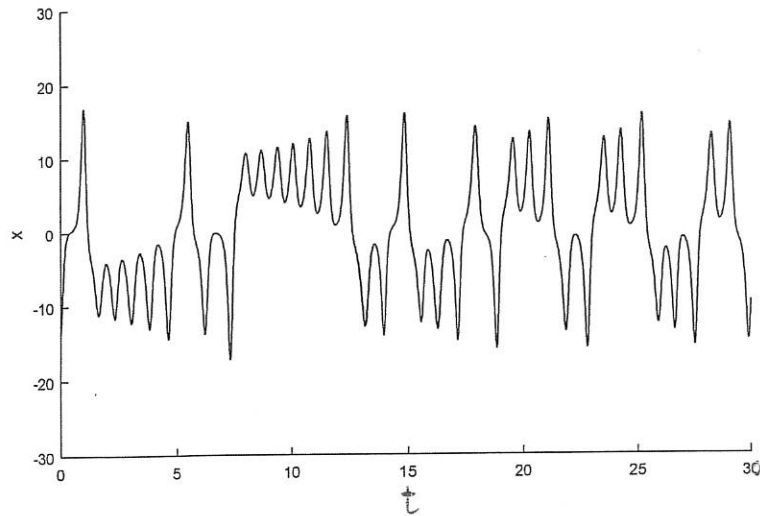
Layout  $\rightarrow$  Plotting region  $\rightarrow$  fissiamo gli estremi degli assi

Absciss  Time  
 Ordinate  Coordinates

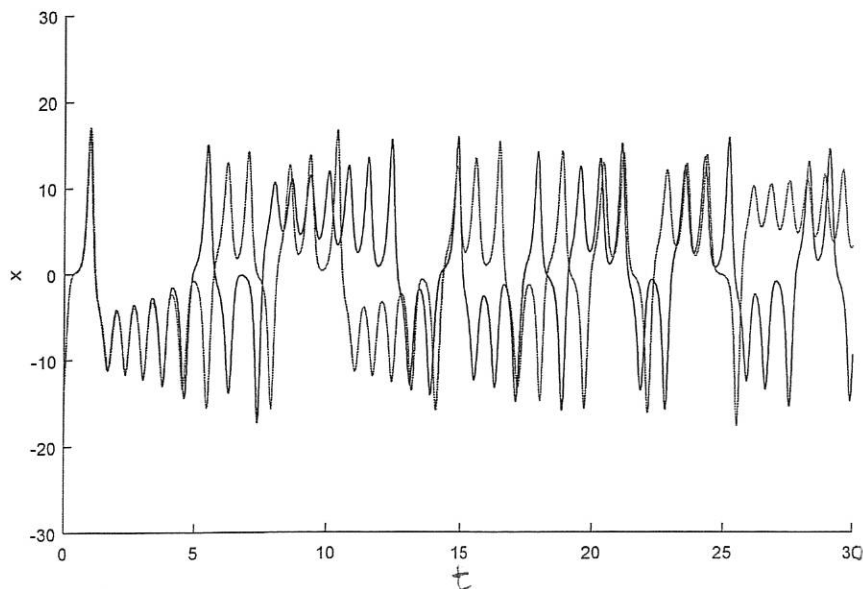
2D Plotting region:1

Abscissa  ...

Ordinate  ...



Per verificare la dipendenza sensibile dalle condizioni iniziali, nella  
 finestra di Starter, modifichiamo  $x$  in  $-13,6$  (piccola variazione  
 nella condizione iniziale), compute  $\rightarrow$  Forward simuliamo.



Le due traiettorie (sono qui visualizzati i momenti  $x(t)$ ) inizialmente vicine ( $t < 5$ ) si allontanano e si riavvicinano continuamente

↓  
dipendenza sensibile  
dalle condizioni  
iniziali

↓  
stretching and  
folding

NOTA

Per visualizzare la classica forma a "farfalla" dell'attrattore di Lorenz, a linea di comando Matlab, posizionati in "systems\lorenz\_model\  
diagram"

carichiamo il file attrattore.mat

`>> load attrattore.mat`

la matrice  $x$  contiene il vettore di stato del sistema

`>> x = x';`

`>> x1 = x(:,1);` (variabile di stato  $x$ )

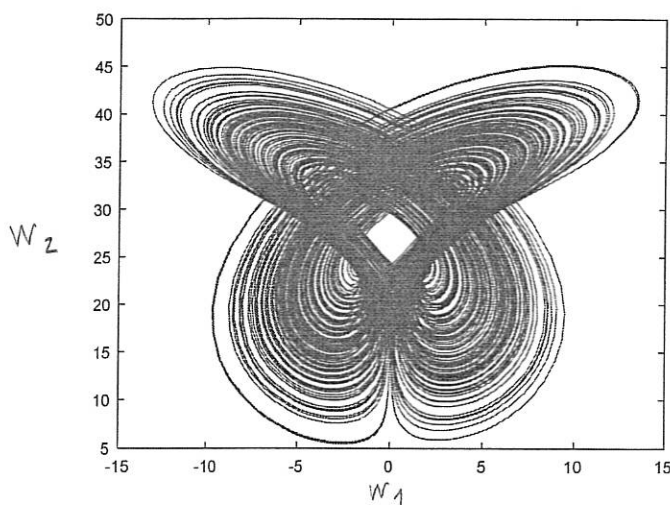
`>> x2 = x(:,2);` ( "  $y$ )

`>> x3 = x(:,3);` ( "  $z$ )

`>> w1 = x1 - x2;`

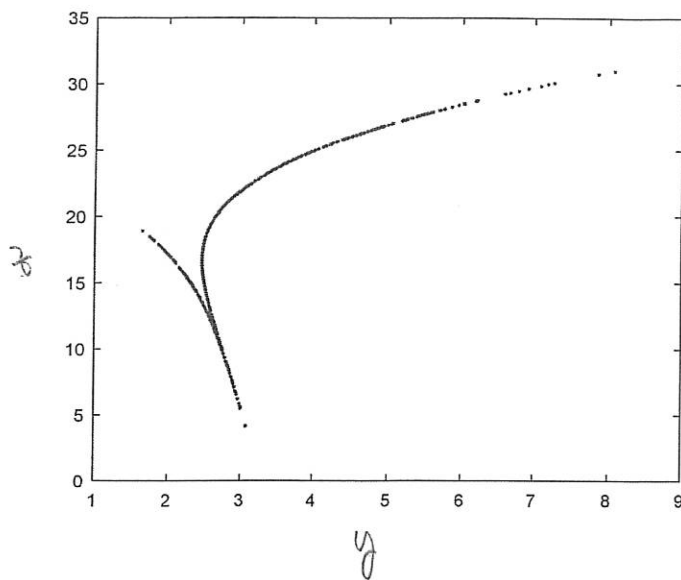
`>> w2 = z;`

`>> figure; plot(w1, w2)` → genera la "farfalla"



Vediamo ora l'attrattore sulle sezioni di Poincaré  $x_1 = 1,5$  ottenuta con `poincare-lorenz.m` (vedi file allegati)

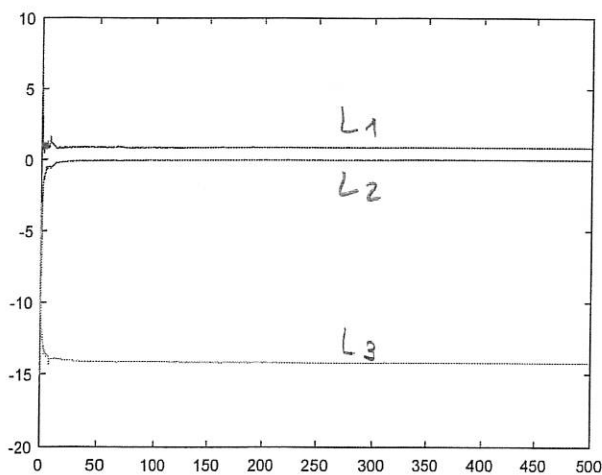




Sezione di Poincaré

$$X = 1,5$$

Calcoliamo ora gli esponenti di Liapunov associati all'attrattore con il file Liapunov-primo-esponente.m (vedi file allegati)



$$L_1 = 0,8643 > 0$$

$$L_2 = -0,0067 \approx 0$$

$$L_3 = -14,1909$$

Il primo esponente di Liapunov vale 0,8643; essendo positivo caratterizza la caoticità dell'attrattore.

### poincare\_lorenz.m

```
y0=[-0.6;-0.6;17]; % Stato iniziale
odefun = @(t,x) fun_eval(t,x,10,28,2.33333); % puntatori a funzioni che devono essere definite
event = @(t,x) poincare_section(t,x);
tspan = [0, 4000]; % intervallo di simulazione

%%
[T,Y,TE,YE,IE] = poincare_map(odefun,event,tspan,y0); % lungo la simulazione estrae i punti che si trovano sulla sezione di Poincaré
%%
figure;
plot(YE(:,2),YE(:,3),'.'); % plot della sezione
```

### poincare\_map.m

Calcola la mappa e determina i punti sulla sezione

```
function [T,Y,TE,YE,IE] = poincare_map(odefun,event,tspan,y0)
t0=tspan(1);
t1=tspan(2);
options=[];
[T,Y] = ode45(odefun,[t0,t1/10],y0,options); % Leave the transient
=> options=odeset('Events',event);
[T,Y,TE,YE,IE] = ode45(odefun,[t1/10,t1],Y(end,:),options);
end
```

### fun\_eval.m

eqz. di stato → dal file Mitcont  
lorenz\_model.m

```
function dydt = fun_eval(t,kmrgd,sigma,rho,beta)
dydt=[sigma*(kmrgd(2)-kmrgd(1));
rho*kmrgd(1)-kmrgd(1)*kmrgd(3)-kmrgd(2);
kmrgd(1)*kmrgd(2)-beta*kmrgd(3)];
end
```

### poincare\_section.m

```
function [value,isterminal,direction]=poincare_section(t,x)
value=x(1)-1.5;
isterminal=0;
direction=1;
end
```

→ "Estrae" i punti lungo una simulazione che ritrovano sulla sezione di Poincaré

$x = 1,5$

↓

$x(1)$

### Liapunov\_primo\_esponente.m

```
y0=[-0.6;-0.6;17]; → stato iniziale  
odefun = @(t,x) fun_eval(t,x,10,28,2.33333); → puntatori a funzioni che  
jac = @(t,x) jacobian(t,x,10,28,2.33333) devono essere definite  
tspan = [0, 1000];  
  
%% calcolo gli esponenti  
[Texp,Lexp]=lexp(odefun,jac,tspan,y0); espr di stato  
Lexp(:,length(Lexp)) jacobiano
```

---

jacobian.m → del file *Matcont*  
*new\_model.m*

```
function jac = jacobian(t,kmrgd,sigma,rho,beta)  
jac=[ -sigma , sigma , 0 ; rho - kmrgd(3) , -1 , -kmrgd(1) ; kmrgd(2) , kmrgd(1)  
 , -beta ];  
end
```

---

### lexp.m

```
function [Texp,Lexp]=lexp(odefun,jacobian,tspan,y0)  
stept=0.2;  
ioutp=100;  
n1=length(y0); n2=n1*(n1+1);  
nit = round(diff(tspan)/stept); % Number of steps  
% Memory allocation  
y=zeros(n2,1); cum=zeros(n1,1);  
Lexp=zeros(n1,nit); Texp=zeros(1,nit);  
% Initial values  
rhs_ext=@(t,x)  
[odefun(t,x);reshape(jacobian(t,x)*reshape(x(n1+1:n2),n1,n1),n2-n1,1)];  
y=[y0(:);reshape(eye(n1),n1^2,1)];  
t=tspan(1);  
% Main loop  
for ITERLYAP=1:nit  
[T,Y] = ode45(rhs_ext,[t t+stept],y); % Solution of extended ODE system  
t=t+stept; y=Y(size(Y,1),:); % Take the last computed point  
[Q,R]=qr(reshape(y(n1+1:n2),n1,n1)); % Construct new orthonormal basis  
y(n1+1:n2)=Q(:);  
cum=cum+log(abs(diag(R))); % Compute lyapunov coefficient  
lp=cum/(t-tspan(1)); % normalize exponent  
Lexp(:,ITERLYAP)=lp; Texp(ITERLYAP)=t;  
if (mod(ITERLYAP,ioutp)==0)  
fprintf('t=%6.4f ',t); fprintf('%10.6f ',lp); fprintf('\n');  
end  
end  
figure, plot(Texp,Lexp)  
end
```

---

## 2. Modello di Lorenz (84)

Select → System → New inseriamo il modello

System

Name:

Coordinates:

Parameters:

Time:

Derivatives

	1st ord	2nd ord	3rd ord	4th ord	5th ord
- numerically	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>
- from window	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
- symbolically	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

$x' = y^2 + z^2 + a*x - a*F$   
 $y' = -x*y + b*x*z + y - G$   
 $z' = -b*x*y - x*z + z$

OK Cancel

Type → Initial point → Point definiamo condizioni iniziali e valori dei parametri per la simulazione (il tipo di curva è Orbit)

Integrator

Integration data

Method:

Interval:

InitStepsize:

MaxStepsize:

Rel. Tolerance:

Abs. Tolerance:

Refine:

Normcontrol:

Starter

Initial Point

t	<input type="text" value="0"/>
x	<input type="text" value="1"/>
y	<input type="text" value="0"/>
z	<input type="text" value="0.5"/>
a	<input type="text" value="0.25"/>
b	<input type="text" value="4"/>
F	<input type="text" value="2"/>
G	<input type="text" value="1.67"/>

Select Cycle

Window → Graphic → 3D plot apriamo una finestra grafica in cui visualizzare la traiettoria (orbita) nello spazio di stato (x,y,z)

Layout → Plotting region  
fissiamo gli estremi degli assi

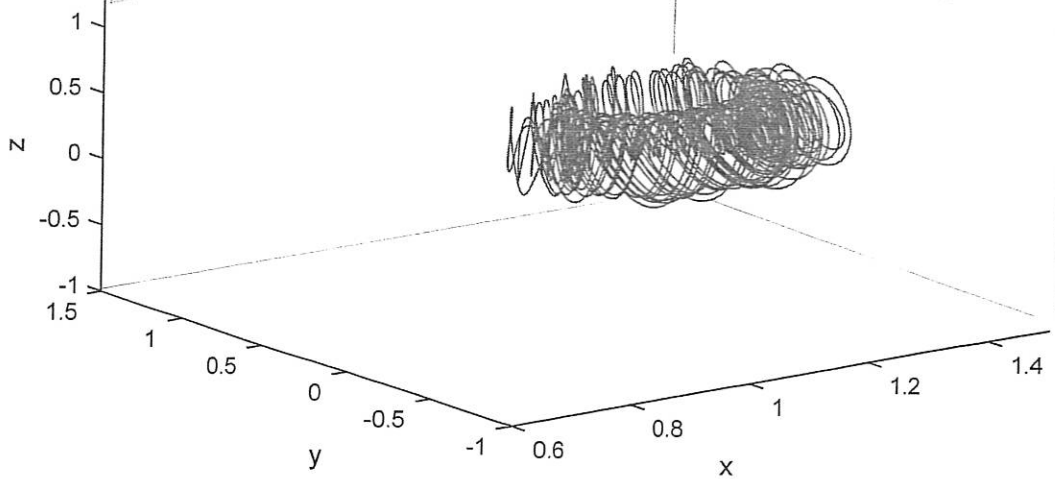
3D Plotting region:1

Abscissa:  ...

Ordinate:  ...

Applicate:  ...

Compute  $\rightarrow$  Forward simuliamo il sistema partendo dalle condizioni iniziali assegnate



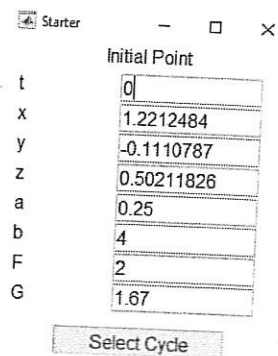
Select  $\rightarrow$  Curve  $\rightarrow$  Actions  $\rightarrow$  Rename rinominiamo la curva  $P_0(t)$  in "transitorio".

Per determinare l'attrattore, fissiamo come nuova condizione iniziale l'ultimo punto della curva "transitorio"

Select  $\rightarrow$  Initial point  $\rightarrow$  ... This is the last ...  $\rightarrow$  select



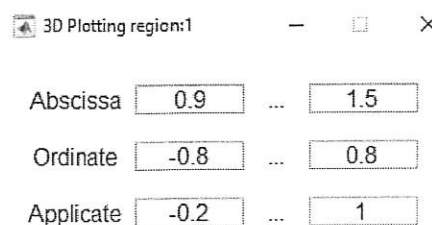
Nella finestra Starter (aggiornata ai nuovi valori) fissare  $t=0$



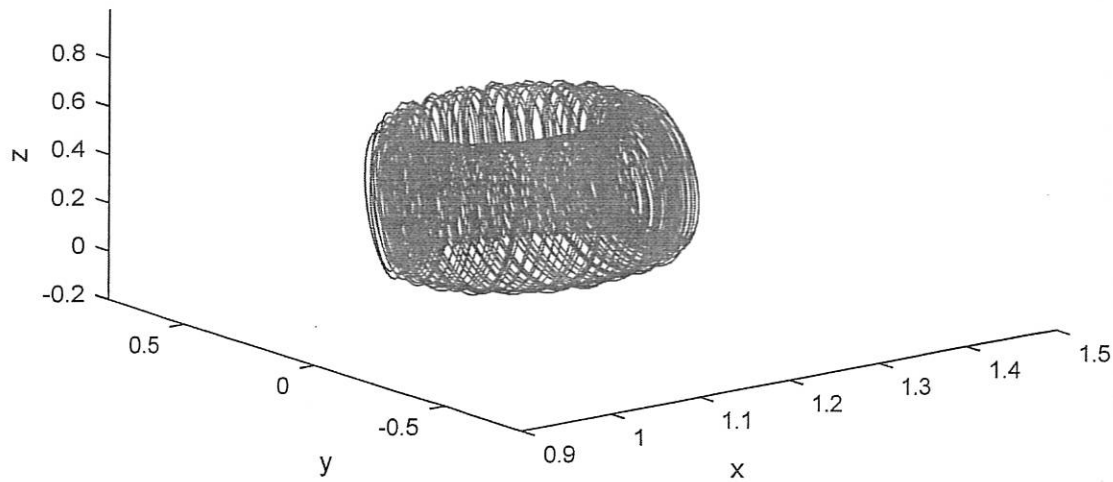
Plot  $\rightarrow$  Clear puliamo la finestra grafica

Layout  $\rightarrow$  Plotting region

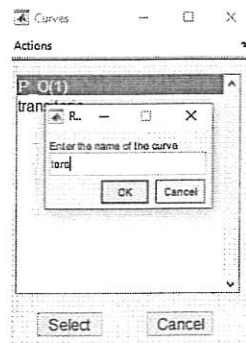
fissiamo dei nuovi estremi per gli assi



compute → Forward simuliamo l'attrattore: è un toro



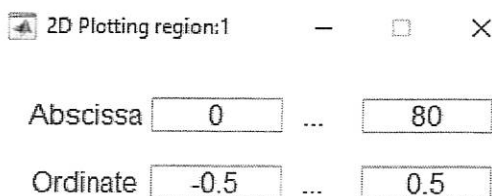
Select → Curve → Actions → Rename rinominiamo la curva in "toro"



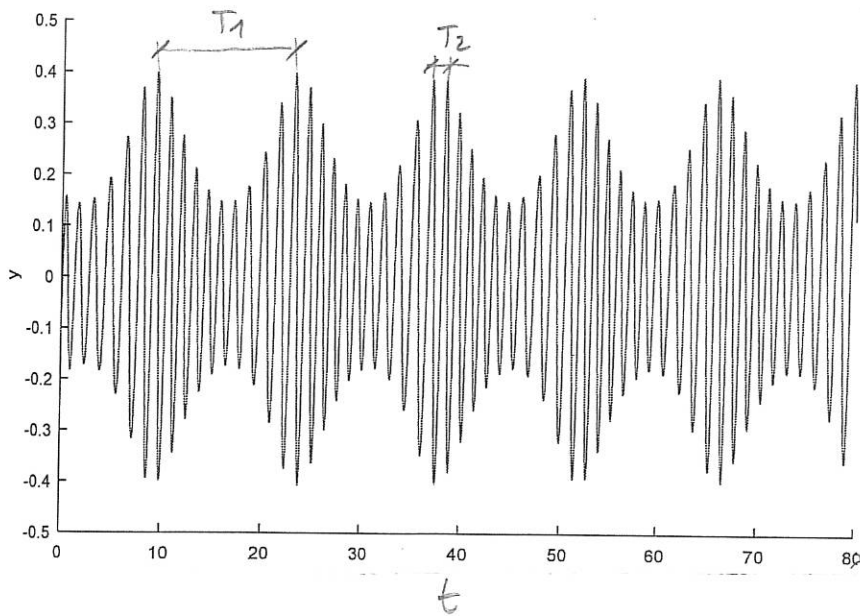
Possiamo anche visualizzare la serie temporale, per esempio della variabile  $y$ , su toro

Window → Graphic → 2Dplot apriamo una finestra grafica bidimensionale in cui in ascissa visualizziamo time →  $t$  e in ordinata coordinate →  $y$

Layout → Plotting region fissiamo gli estremi sugli assi



Plot  $\rightarrow$  Redraw curve mostra il movimento  $y(t)$



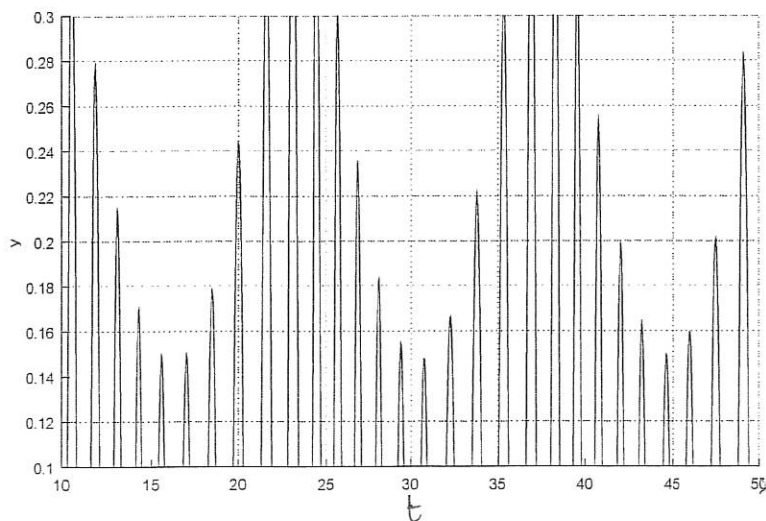
L'andamento suggerisce la presenza di due pulsazioni (responsabili delle oscillazioni del segnale in alta e bassa periodicità  $T_1 \approx 15$  e  $T_2 \approx 2$ ) che non essendo in rapporto razionale non generano un comportamento periodico ma quasi periodico (toro).

Infatti, digitando i comandi Matlab

`>> grid`

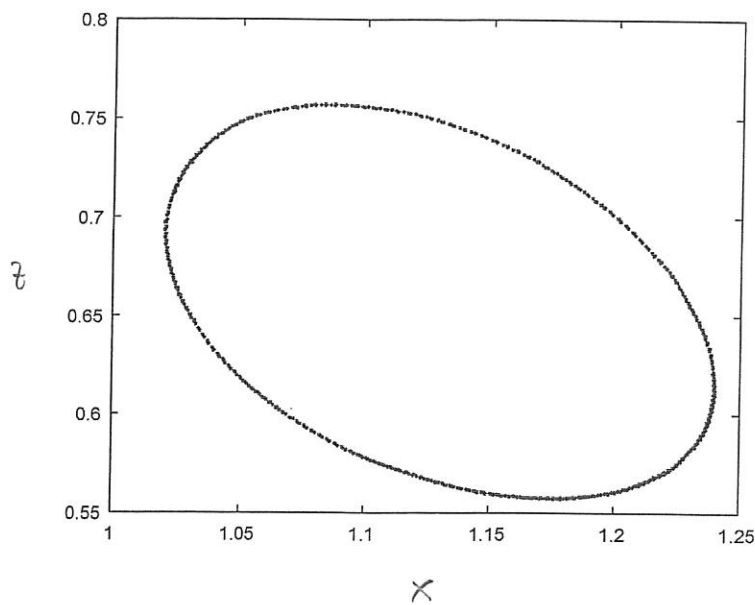
`>> axis([10 50 0.1 0.3])`

si ottiene



Da cui si nota meglio l'assenza di periodicità e la quasi-periodicità del comportamento

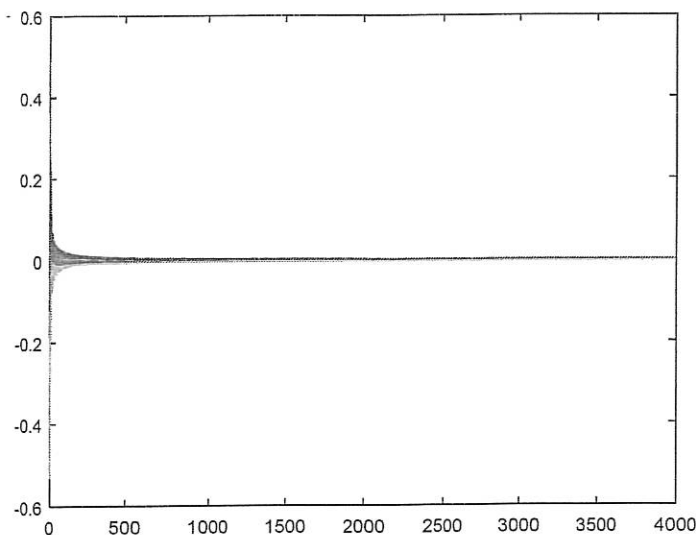
Vediamo ora l'attrattore sulla sezione di Poincaré  $y=0$   
 ottenuta con `poincare_toro.m` (vedi file allegati)



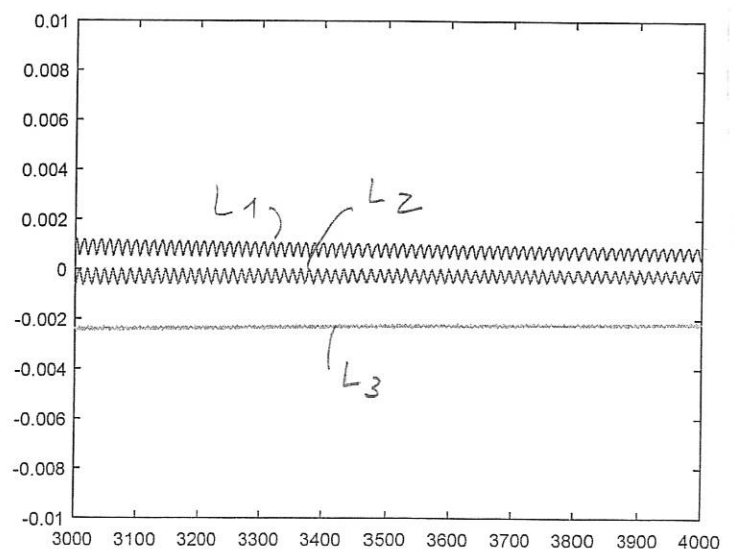
sezione  $P$   $y=0$

TORO

Calcoliamo gli esponenti di Liapunov associati all'attrattore con  
 il file `Liapunov-primo-esponente.m` (vedi file allegati)



$\gg \text{axis}([3000 \ 4000 \ -0.01 \ 0.01])$



$$L_1 = -0.0005$$

$$L_2 = -0.0001$$

$$L_3 = -0.0022$$

$$L_1, L_2 \approx 0 \Rightarrow \text{TORO}$$

$$L_3 < 0$$



### poincare\_toro.m

```
clear all; close all;

y0=[1.2;-0.14;0.17];
odefun = @(t,x) fun_eval(t,x,0.25,4,2,1.67);
event = @(t,x) poincare_section(t,x);
tspan = [0,1000];

%%
[T,Y,TE,YE,IE] = poincare_map(odefun,event,tspan,y0);

%%
figure;
plot(YE(:,1),YE(:,3),'.');
```

---

### poincare\_map.m

```
function [T,Y,TE,YE,IE] = poincare_map(odefun,event,tspan,y0)
t0=tspan(1);
t1=tspan(2);
options=[];
[T,Y] = ode45(odefun,[t0,t1],y0,options); % Leave the transient
options=odeset('Events',event,'RelTol',1e-3,'AbsTol',1e-6);
[T,Y,TE,YE,IE] = ode45(odefun,[t0,t1],Y(end,:),options);
end
```

---

### fun\_eval.m

```
function dydt = fun_eval(t,kmrgd,a,b,F,G)
dydt=[ kmrgd(2)^2 + kmrgd(3)^2 + a*kmrgd(1) - a*F;
- kmrgd(1)*kmrgd(2) + b*kmrgd(1)*kmrgd(3) + kmrgd(2) - G;
- b*kmrgd(1)*kmrgd(2) - kmrgd(1)*kmrgd(3) + kmrgd(3)];
end
```

---

### poincare\_section.m

```
function [value,isterminal,direction]=poincare_section(t,x)
value=x(2)-0;
isterminal=0;
direction=1;
end
```

---

### Liapunov\_primo\_esponente.m

```
clear all; close all;

y0=[1.2;-0.14;0.17];
odefun = @(t,x) fun_eval(t,x,0.25,4,2,1.67);
jac = @(t,x) jacobian(t,x,0.25,4,2,1.67)
tspan = [0,4000];

%%
[Texp,Lexp]=lexp(odefun,jac,tspan,y0);
Lexp(:,length(Lexp))
```

---

### jacobian.m

```
function jac = jacobian(t,kmrgd,a,b,F,G)
jac=[ a , 2*kmrgd(2) , 2*kmrgd(3) ; b*kmrgd(3) - kmrgd(2) , 1 - kmrgd(1) ,
b*kmrgd(1) ; - kmrgd(3) - b*kmrgd(2) , -b*kmrgd(1) , 1 - kmrgd(1) ];
end
```

---

### lexp.m

```
function [Texp,Lexp]=lexp(odefun,jacobian,tspan,y0)
stept=0.2;
ioutp=100;
n1=length(y0); n2=n1*(n1+1);
nit = round(diff(tspan)/stept); % Number of steps
% Memory allocation
y=zeros(n2,1); cum=zeros(n1,1);
Lexp=zeros(n1,nit); Texp=zeros(1,nit);
% Initial values
rhs_ext=@(t,x)
[odefun(t,x);reshape(jacobian(t,x)*reshape(x(n1+1:n2),n1,n1),n2-n1,1)];
y=[y0(:);reshape(eye(n1),n1^2,1)];
t=tspan(1);
% Main loop
for ITERLYAP=1:nit
[T,Y] = ode45(rhs_ext,[t t+stept],y); % Solutuion of extended ODE system
t=t+stept; y=Y(size(Y,1),:); % Take the last computed point
[Q,R]=qr(reshape(y(n1+1:n2),n1,n1)); % Construct new orthonormal basis
y(n1+1:n2)=Q(:);
cum=cum+log(abs(diag(R))); % Compute lyapunov coefficient
lp=cum/(t-tspan(1)); % normalize exponent
Lexp(:,ITERLYAP)=lp; Texp(ITERLYAP)=t;
if (mod(ITERLYAP,ioutp)==0)
fprintf('t=%6.4f ',t); fprintf('%10.6f ',lp); fprintf('\n');
end
end
figure, plot(Texp,Lexp)
end
```

---

### 3 Modello preda - predatore - superpredatore

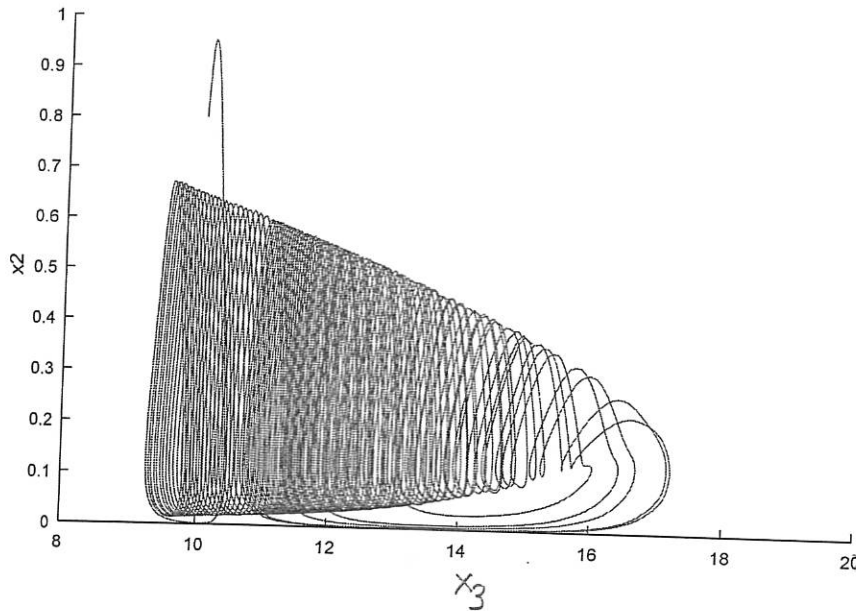
Select → System → New inseriamo il modello

Type → Initial point → Point definiamo condizioni iniziali e valori dei parametri per la simulazione

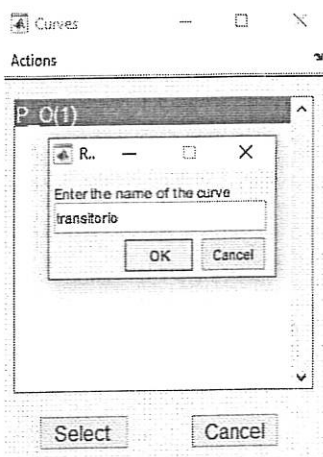
Window → Graphic → 2D plot apriamo una finestra grafica in cui visualizzare la traiettoria nello spazio ( $x_3, x_2$ )

Layout → Plotting region fissiamo gli estremi sugli assi

Compute  $\rightarrow$  Forward simuliamo il sistema partendo dalla condizione iniziale assegnata

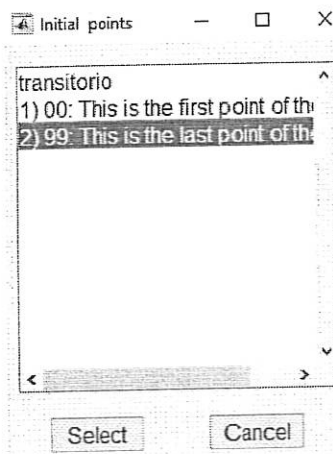


Select  $\rightarrow$  Curve  $\rightarrow$  Actions  $\rightarrow$  Rename rinominiamo la curva in "transitorio"



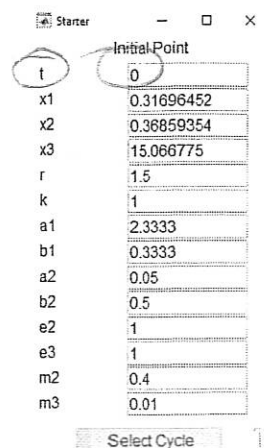
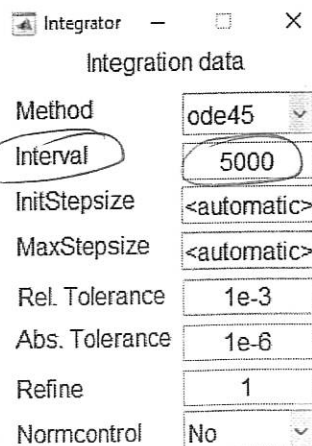
Per determinare l'attrattore, fissiamo come nuova condizione iniziale l'ultimo punto della curva "transitorio"

Select  $\rightarrow$  Initial point  $\rightarrow$  ... This is the last...  $\rightarrow$  Select



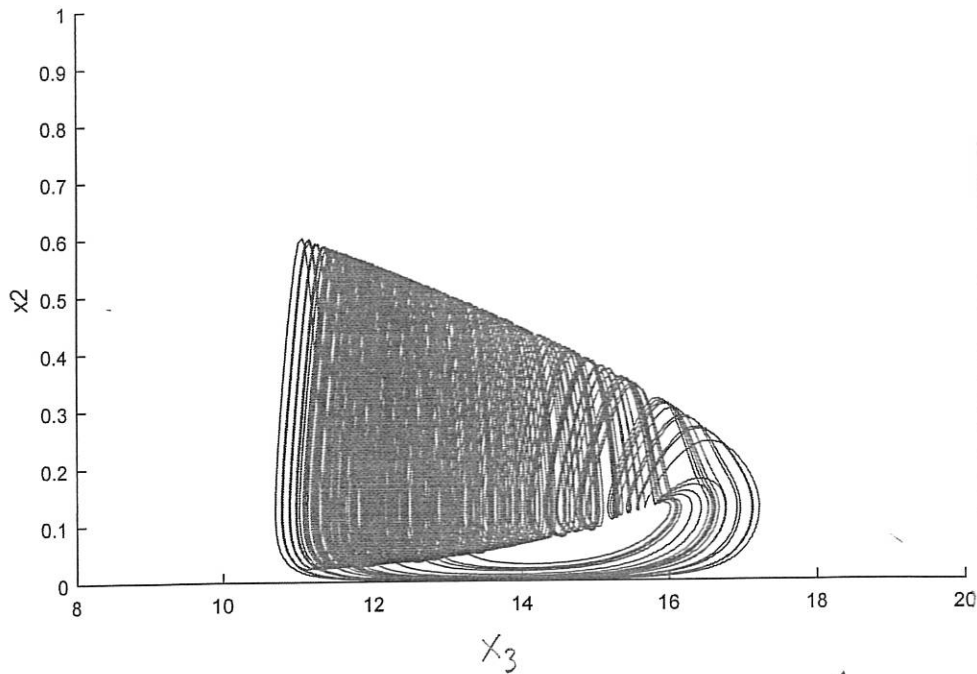
Aggiorniamo nella finestra Integrator il tempo di simulazione Interval a 5000 e nella finestra Starter l'istante iniziale di simulazione  $t$  a 0

Plot  $\rightarrow$  Clear puliamo le finestre grafiche

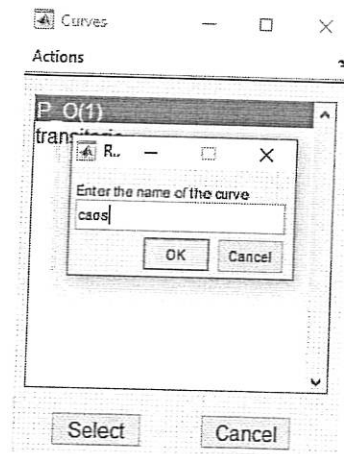


Compute → Forward

simuliamo l'attrattore: è un attrattore caotico.



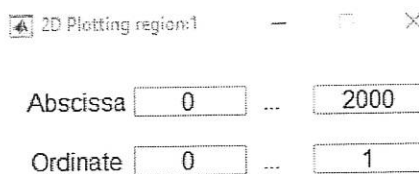
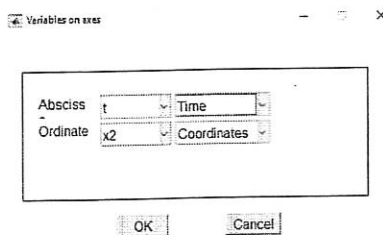
Select → Curve → Actions → Rename rinominiamo la curva in "aos"

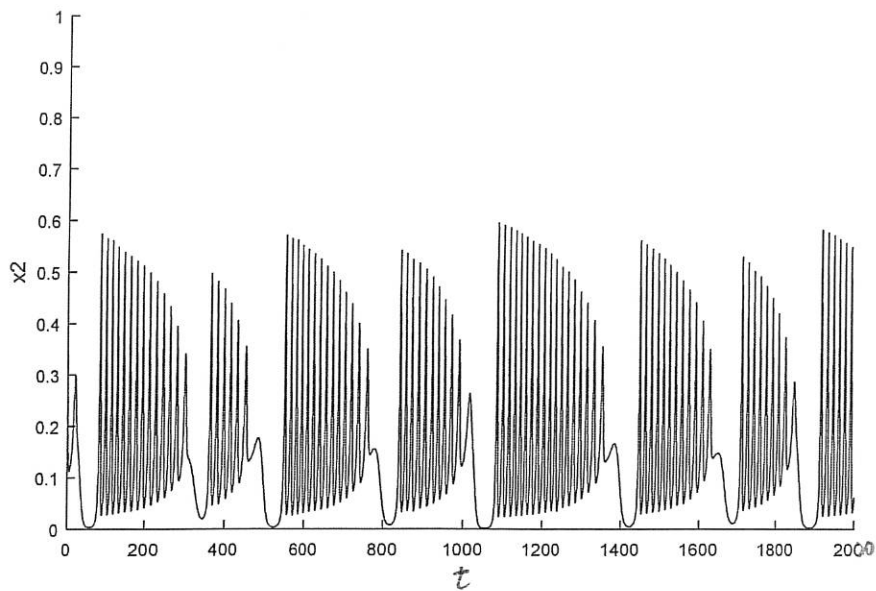


Volendo visualizzare la serie temporale, ad esempio, della variabile  $x_2$  ...

Layout → Variables on axes

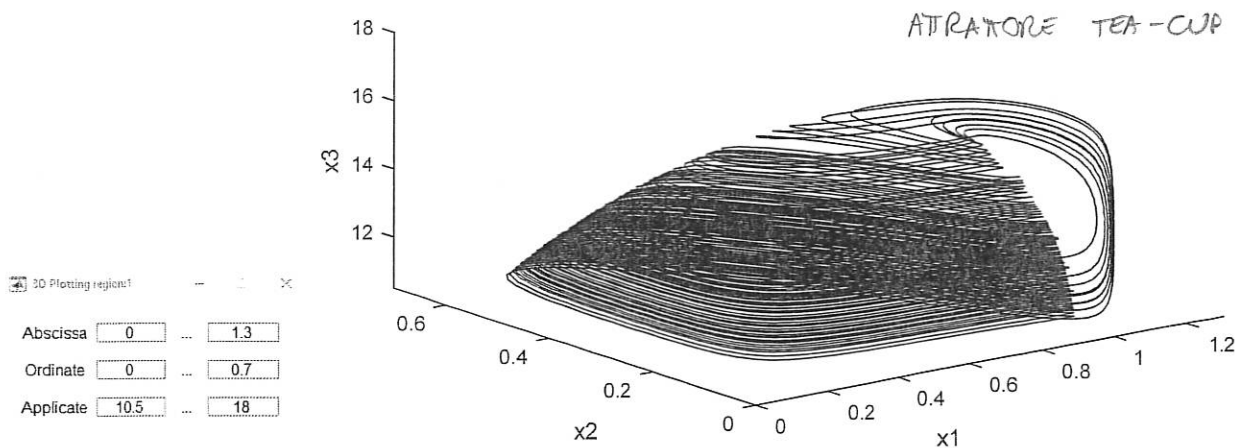
Layout → Plotting region





Per visualizzare l'attrattore costico nello spazio di stato  $(x_1, x_2, x_3)$  apriamo una nuova finestra grafica con

Window → Graphic → 3Dplot



Fissiamo ora  $b_2$  al valore 0,46 e simuliamo il sistema visualizzando la transizione e l'attrattore nello spazio  $(x_3, x_2)$  e nello spazio di stato  $(x_1, x_2, x_3)$

Starter

Initial Point	
t	0
x1	0.7
x2	0.24
x3	13
r	1.5
k	1
a1	2.3333
b1	0.3333
a2	0.05
b2	0.46
e2	1
e3	1
m2	0.4
m3	0.01

Select Cycle

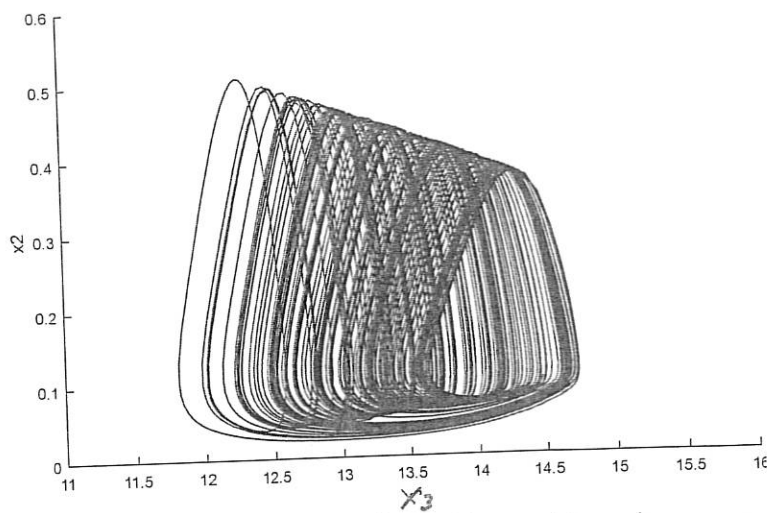
→ Aggiornare anche questi

← 0,46!

2D Plotting region:1

Abscissa 11 ... 16

Ordinate 0 ... 0.6



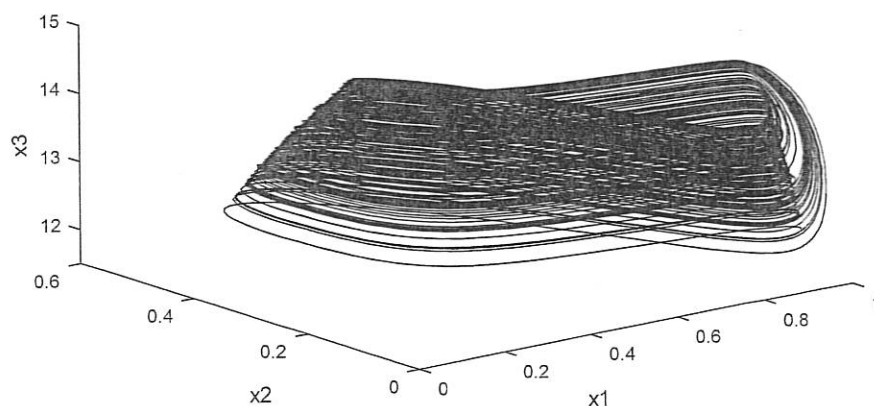
ATTRATTORE COT-TEA-CUP

3D Plotting region:1

Abscissa 0 ... 1

Ordinate 0 ... 0.6

Applicate 11.5 ... 15



ALTRI COMPORTAMENTI

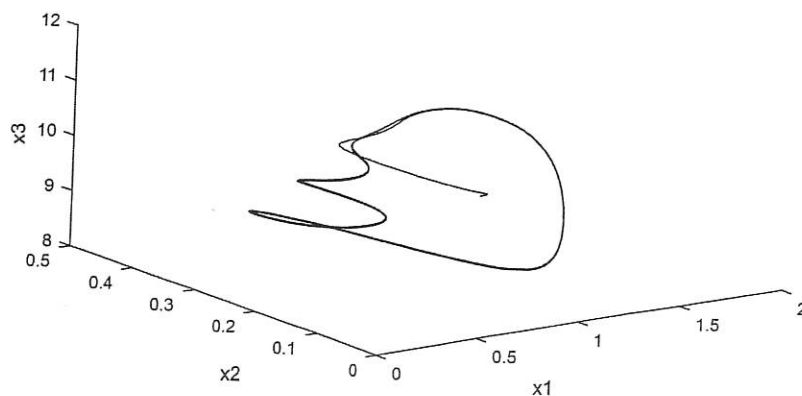
• coesistenza ciclica  $x_1, x_2, x_3$   $m_2 = 98$   $x(0) = (0.8, 0.1, 10)$

3D Plotting region:1

Abscissa 0 ... 2

Ordinate 0 ... 0.5

Applicate 8 ... 12



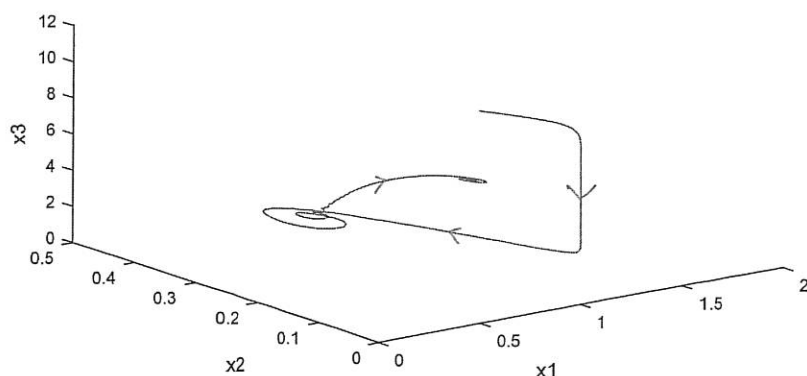
• Corrente estacionária  $x_1, x_2, x_3$   $m_2 = 1, 2$

3D Plotting region:1

Abscissa 0 ... 2

Ordinate 0 ... 0.5

Applicate 0 ... 12



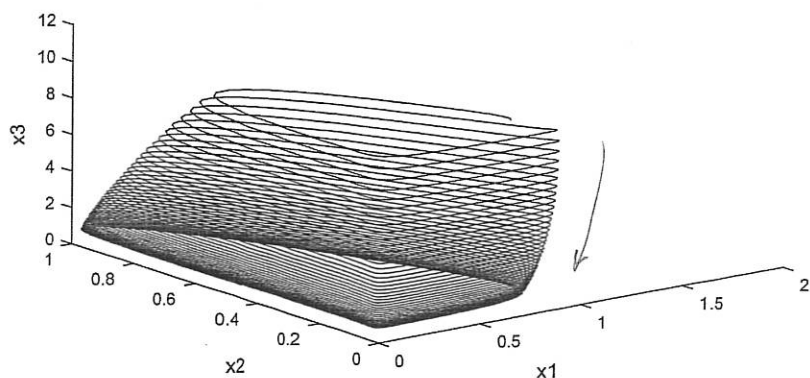
• Corrente ciclica  $x_1, x_2$  com  $x_3 = 0$   $m_2 = 0,4$   $e_3 = 0,5$

3D Plotting region:1

Abscissa 0 ... 2

Ordinate 0 ... 1

Applicate 0 ... 12



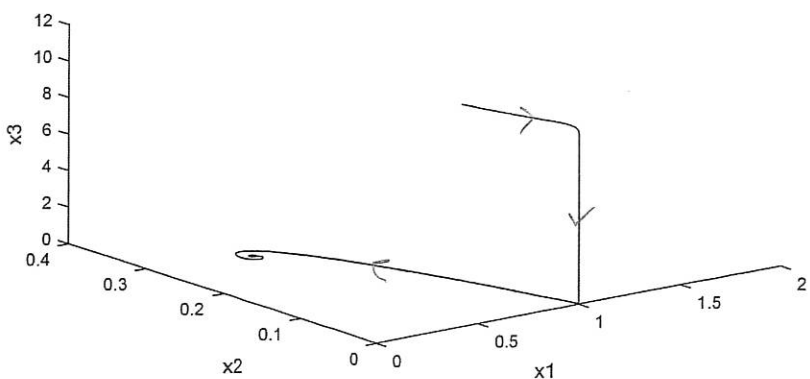
• Corrente estacionária  $x_1, x_2$  com  $x_3 = 0$   $m_2 = 0,4$   $e_3 = 0,5$   $e_2 = 0,3$

3D Plotting region:1

Abscissa 0 ... 2

Ordinate 0 ... 0.4

Applicate 0 ... 12

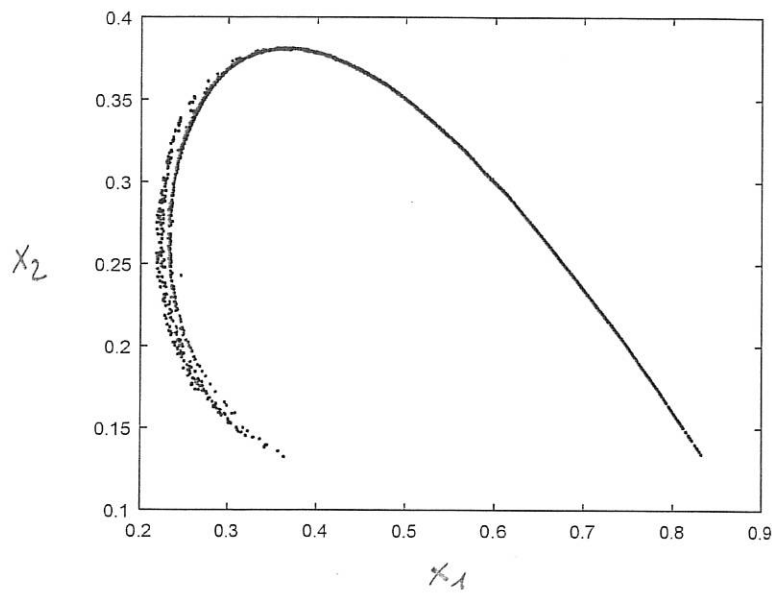


• Solo  $x_1 = k$  com  $x_2 = x_3 = 0$

$m_2 = 0,4$   $e_3 = 0,5$   $e_2 = 0,2$



Vediamo ora l'attrattore caotico ottenuto per  $b_2 = 0,5$  sulla sezione di Poincaré  $x_3 = 15$  (file poincare-rucca3.m allegato)

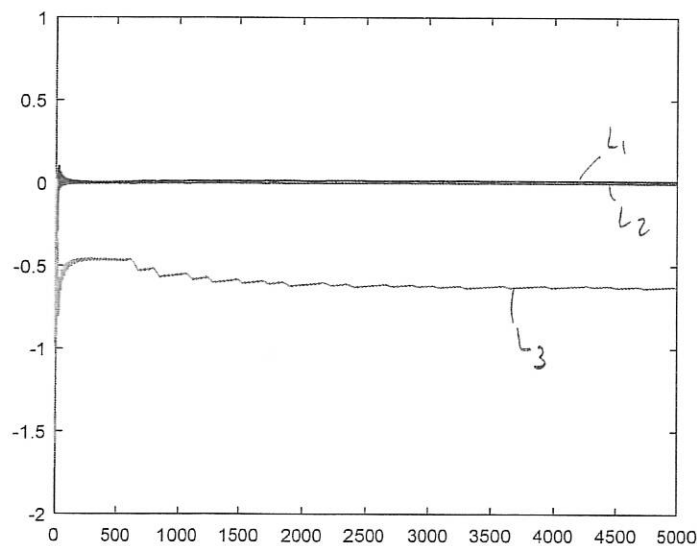


Sezione P

$x_3 = 15$

CAOS

Calcoliamo gli esponenti di Liapunov con liapunov-primo-esponente.m



$$L_1 \approx 0,013 > 0 \Rightarrow \text{CAOS}$$

$$L_2 \approx -0,001 \approx 0$$

$$L_3 \approx -0,63 < 0$$

### **poincare\_rmca3.m**

```
clear all; close all;
y0=[0.5;0.8;10];
odefun = @(t,x) fun_eval(t,x,1.5,1,2.3333,0.3333,0.05,0.5,1,1,0.4,0.01);
event = @(t,x) poincare_section(t,x);
tspan = [0, 200000];

%%
[T,Y,TE,YE,IE] = poincare_map(odefun,event,tspan,y0);

%%
figure;
plot(Y(:,3),Y(:,2));
figure;
plot(YE(:,1),YE(:,2),'.');
```

---

### **poincare\_map.m**

```
function [T,Y,TE,YE,IE] = poincare_map(odefun,event,tspan,y0)
t0=tspan(1);
t1=tspan(2);
options=[];
[T,Y] = ode45(odefun,[t0,t1/10],y0,options); % Leave the transient
options=odeset('Events',event);
[T,Y,TE,YE,IE] = ode45(odefun,[t0,t1],Y(end,:),options);
end
```

---

### **fun\_eval.m**

```
function dydt = fun_eval(t,kmrgd,r,k,a1,b1,a2,b2,e2,e3,m2,m3)
dydt=[ r*kmrgd(1)*(1-kmrgd(1)/k) - a1*kmrgd(1)*kmrgd(2)/(b1+kmrgd(1));
      e2*a1*kmrgd(1)*kmrgd(2)/(b1+kmrgd(1)) - a2*kmrgd(2)*kmrgd(3)/(b2+kmrgd(2)) -
      m2*kmrgd(2);
      e3*a2*kmrgd(2)*kmrgd(3)/(b2+kmrgd(2)) - m3*kmrgd(3)];
end
```

---

### **poincare\_section.m**

```
function [value,isterminal,direction]=poincare_section(t,x)
value=x(3)-15;
isterminal=0;
direction=1;
end
```

---

### Liapunov\_primo\_esponente.m

```
clear all; close all;
y0=[0.5;0.8;10];
odefun = @(t,x) fun_eval(t,x,1.5,1,2.3333,0.3333,0.05,0.5,1,1,0.4,0.01);
jac = @(t,x) jacobian(t,x,1.5,1,2.3333,0.3333,0.05,0.5,1,1,0.4,0.01)
tspan = [0, 5000];
%%
[Texp,Lexp]=lexp(odefun,jac,tspan,y0);
Lexp(:,length(Lexp))
```

---

### jacobian.m

```
function jac = jacobian(t,kmrgd,r,k,a1,b1,a2,b2,e2,e3,m2,m3)
jac=[ (a1*kmrgd(1)*kmrgd(2))/(b1 + kmrgd(1))^2 - (r*kmrgd(1))/k -
(a1*kmrgd(2))/(b1 + kmrgd(1)) - r*(kmrgd(1)/k - 1) , -(a1*kmrgd(1))/(b1 +
kmrgd(1)) , 0 ; (a1*e2*kmrgd(2))/(b1 + kmrgd(1)) - (a1*e2*kmrgd(1)*kmrgd(2))/(b1
+ kmrgd(1))^2 , (a1*e2*kmrgd(1))/(b1 + kmrgd(1)) - (a2*kmrgd(3))/(b2 + kmrgd(2))
- m2 + (a2*kmrgd(2)*kmrgd(3))/(b2 + kmrgd(2))^2 , -(a2*kmrgd(2))/(b2 + kmrgd(2))
; 0 , (a2*e3*kmrgd(3))/(b2 + kmrgd(2)) - (a2*e3*kmrgd(2)*kmrgd(3))/(b2 +
kmrgd(2))^2 , (a2*e3*kmrgd(2))/(b2 + kmrgd(2)) - m3 ];
end
```

---

### lexp.m

```
function [Texp,Lexp]=lexp(odefun,jacobian,tspan,y0)
stept=0.2;
ioutp=100;
n1=length(y0); n2=n1*(n1+1);
nit = round(diff(tspan)/stept); % Number of steps
% Memory allocation
y=zeros(n2,1); cum=zeros(n1,1);
Lexp=zeros(n1,nit); Texp=zeros(1,nit);
% Initial values
rhs_ext= @(t,x)
[odefun(t,x);reshape(jacobian(t,x)*reshape(x(n1+1:n2),n1,n1),n2-n1,1)];
y=[y0(:);reshape(eye(n1),n1^2,1)];
t=tspan(1);
% Main loop
for ITERLYAP=1:nit
[T,Y] = ode45(rhs_ext,[t t+stept],y); % Solution of extended ODE system
t=t+stept; y=Y(size(Y,1),:); % Take the last computed point
[Q,R]=qr(reshape(y(n1+1:n2),n1,n1)); % Construct new orthonormal basis
y(n1+1:n2)=Q(:);
cum=cum+log(abs(diag(R))); % Compute lyapunov coefficient
lp=cum/(t-tspan(1)); % normalize exponent
Lexp(:,ITERLYAP)=lp; Texp(ITERLYAP)=t;
if (mod(ITERLYAP,ioutp)==0)
fprintf('t=%6.4f ',t); fprintf('%10.6f ',lp); fprintf('\n');
end
end
figure, plot(Texp,Lexp)
end
```

---