```
> #Nikita John, Assignment 21
 #Okay to Post, November 15th, 2021
> ###########################################################################
## DMB.txt Save this file as  DMB.txt  to use it,                   #
# stay in the                                          #
## same directory, get into Maple (by typing: maple <Enter> )        #
## and then type:  read `DMB.txt` <Enter>                          #
## Then follow the instructions given there                       #
##                                                     #
## Written by Doron Zeilberger, Rutgers University ,                #
##  DoronZeil at gmail dot com                                  #
###########################################################################


print(`First Written: Nov. 2021 `) :
print( ) :
     print(`This is DMB.txt, A Maple package to explore Dynamical models in Biology (both
     discrete and continuous)`) :
     print(`accompanying the class Dynamical Models in Biology, Rutgers University. Taught by
     Dr. Z. (Doron Zeilbeger) `) :

print( ) :
print(`The most current version is available on WWW at:`) :
print(` http://sites.math.rutgers.edu/~zeilberg/tokhniot/DMB.txt .`) :
print(`Please report all bugs to: DoronZeil at gmail dot com .`) :
print( ) :
print(`For general help, and a list of the MAIN functions,`) :
print(` type "Help();". For specific help type "Help(procedure_name);" `) :
print(``) :

print(`------------------------------`) :
print(`For a list of the supporting functions type: Help1();`) :
print(`For help with any of them type: Help(ProcedureName);`) :
print( ) :
 print(`------------------------------`) :

     print(`For a list of the functions that give examples of Discrete-time dynamical systems (some
     famous), type: HelpDDM();`) :
print(`For help with any of them type: Help(ProcedureName);`) :
print( ) :
 print(`------------------------------`) :


     print(`For a list of the functions continuous-time dynamical systems (some famous) type:
     HelpCDM();`) :
print(`For help with any of them type: Help(ProcedureName);`) :
print( ) :
 print(`------------------------------`) :
```

```
with(LinearAlgebra) :

Help1 :=proc( )
if args = NULL then

print( `The SUPPORTING procedures are`) :
print( `IsContStable, IsDisStable, JAC,  PhaseDiag, RandNice, TimeSeriesE, ToSys`) :

else
Help(args) :

fi:


end:



HelpDDM :=proc( )
if args = NULL then

    print( `The  procedures giving discrete-time dynamical systems (some famous), by giving the
    the underlying transformations, followed by the list of variables used are:`) :
print( `AllenSIR, HW, HWg, RT`) :

else
Help(args) :

fi:


end:


HelpCDM :=proc( )
if args = NULL then

    print( `The  procedures giving the underlying transformations, followed by the list of
    variables used are:`) :
print( ` ChemoStat, GeneNet, Lotka, RandNice, SIRS , SIRSdemo, Volterra, VolterraM `) :

else
Help(args) :

fi:


end:
```

```
Help := proc( )
if args = NULL then

print(` DMB.txt: A Maple package for  exploring Dynamical models in Biology  `) :

print(`The MAIN procedures are`) :
print(` ComK, Dis, EquP, FP, Orb, Orbk, PhaseDiag, SEquP, SFP, TimeSeries`) :



elif nargs = 1 and args[ 1 ] = AllenSIR then
    print(`AllenSIR(a,b,c,x,y): The Linda Allen discrete SIR model given in https://sites.math.
    rutgers.edu/~zeilberg/Bio21/LadasSri.pdf`) :
print(`with parameters a,b,c. try:`) :
print(`AllenSIR(1,1/3,1/3,x,y);`) :



elif nargs = 1 and args[ 1 ] = ChemoStat then
    print(`ChemoStat(N,C,a1,a2): The Chemostat continuous-time dynamical system with N=
    Bacterial poplulation densitty, and C=nutient Concentration in growth chamber (see Table
    4.1 of Edelstein-Keshet, p. 122)`) :
    print(`with paramerts a1, a2, Equations (19a_, (19b) in Edelestein-Keshet p. 127 (section
    4.5, where they are called alpha1, alpha2). a1 and a2 can be symbolic or numeric. Try:`) :
print(`ChemoStat(N,C,a1,a2);`) :
print(`ChemoStat(N,C,2,3);`) :


elif nargs = 1 and args[ 1 ] = ComK then
    print(`ComK(F,x,K): inputs a transformation F in the list of variables x, outputs the
    composition of F with itself K times. Try:`) :
print(`ComK([k*x*(1-x)],[x],2);`) :
print(`ComK([x*(1-y),y*(1-x)],[x,y],4);`) :


elif nargs = 1 and args[ 1 ] = Dis then
print(`Dis(F,x,pt,h,A): Inputs a transformation F in the list of variables x`) :
    print(`The approximate orbit of the Dynamical system approximating the  the autonomous
    continuous dynamical  process  `) :
print(`dx/dt=F[1](x(t)) by a discrete time dynamical system with step-size h from t=0 to t=A`) :
print(`Try:`) :
print(`Dis([x*(1-y),y*(1-x)],[x,y],[0.5,0.5], 0.01, 10);`) :

elif nargs = 1 and args[ 1 ] = EquP then
    print(`EquP(F,x): Given a transformation F in the list of variables finds all the Equilibrium
    points of   the continuous-time dynamical system x'(t)=F(x(t))`) :
print(`EquP([5/2*x*(1-x)],[x]]);`) :
print(`EquP([y*(1-x-y),x*(3-2*x-y)],[x,y]]);`) :
```

```
elif nargs = 1 and args[1] = FP then
    print(`FP(F,x): Given a transformation F in the list of variables finds all the fixed point of
    the transformation x->F(x), i.e. the set of solutions of`) :
print(`the system {x[1]=F[1], ..., x[k]=F[k]}. Try:`) :
print(`FP([5/2*x*(1-x)],[x]);`) :
print(`evalf(FP([(1+x+y)/(2+3*x+y),  (3+x+2*y)/(5+x+3*y)],[x,y]));`) :


elif nargs = 1 and args[1] = GeneNet then
    print(`GeneNet(a0,a,b,n,m1,m2,m3,p1,p2,p3): The contiuous-time dynamical system, with
    quantities m1,m2,m3,p1,p2,p3, due to M. Elowitz and S. Leibler`) :
print(`described in the Ellner-Guckenheimer book, Eq. (4.1) (chapter 4,  p. 112)`) :
    print(`and parameers a0 (called alpha_0 there),a (called alpha there), b (called beta there)
    and n. Try:`) :
print(`GeneNet(0,0.5,0.2,2,m1,m2,m3,p1,p2,p3);`) :

elif nargs = 1 and args[1] = HW then
    print(`HW(u,v): The Hardy-Weinberg unerlying transformation witu (u,v,w), Eqs. (53a,53b,
    53c) in Edelestein-Keshet Ch. 3 using the fact that u+v+w=1. try:`) :
print(`HW(u,v);`) :

elif nargs = 1 and args[1] = HWg then
    print(`HWg(u,v,M): The Generalized Hardy-Weinberg unerlying transformation with (u,v),
    M is the survival matrix. Based on Ann Somalwar's HW3g(u,v,w) (only retain the first two
    components and replace w by 1-u-v)`) :
print(`Try:`) :
print(`HWg(u,v,[[1,2,1],[2,3,4],[1,3,2]]);`) :


elif nargs = 1 and args[1] = IsContStable then
    print(`IsContStable(M): inputs a numeric matrix M (given as a list of lists M) and decides
    whether all ite eigenvalues have real negative part. Try`) :
print(`IsContStable([[1,-1],[-1,1]]);`) :


elif nargs = 1 and args[1] = IsDisStable then
    print(`IsDisStable(M): inputs a numeric matrix M (given as a list of lists M) and decides
    whether all ite eigenvalues have absolute value less than 1.Try`) :
print(`IsDisStable([[1,-1],[-1,1]]);`) :

elif nargs = 1 and args[1] = JAC then
    print(`JAC(F,x): The Jacobian Matrix (given as a list of lists) of the transformation F in the
    list of variables x. Try:`) :
print(`JAC([x+y,x^2+y^2],[x,y]); `) :


elif nargs = 1 and args[1] = Lotka then
    print(`Lotka(r1,k1,r2,k2,b12,b21,N1,N2): The Lotka-Volterra continuous-time dynamical
    system, Eqs. (9a),(9b)  (p. 224, section 6.3) of Edelstein-Keshet`) :
```

```
print(`with popoluations N1, N2, and parameters r1,r2,k1,k2, b12, b21 (called there beta_12 and beta_21)`) :
print(`Try:`) :
print(`Lotka(r1,k1,r2,k2,b12,b21,N1,N2);`) :
print(`Lotka(1,2,2,3,1,2,N1,N2);`) :


elif nargs = 1 and args[1] = Orb then
print(`Orb(F,x,x0,K1,K2): Inputs a transformation F in the list of variables x with initial point pt, outputs the trajectory of`) :
print(`of the discrete dynamical system (i.e. solutions of the difference equation): x(n)=F(x(n-1)) with x(0)=x0 from n=K1 to n=K2. `) :
print(`For the full trajectory (from n=0 to n=K2), use K1=0. Try:`) :
print(`Orb(5/2*x*(1-x),[x], [0.5], 1000,1010);`) :
print(`Orb([(1+x+y)/(2+x+y),(6+x+y)/(2+4*x+5*y)],[x,y], [2.,3.], 1000,1010);`) :

elif nargs = 1 and args[1] = Orbk then
print(`Orbk(k,z,f,INI,K1,K2): Given a positive integer k, a letter (symbol), z, an expression f of z[1], ..., z[k] (representing a multi-variable function of the variables z[1],...,z[k]`) :
print(`a vector INI representing the initial values [x[1],..., x[k]], and (in applications) positive integres K1 and K2, outputs the`) :
print(`values of the sequence starting at n=K1 and ending at n=K2. of the sequence satisfying the difference equation`) :
print(`x[n]=f(x[n-1],x[n-2],..., x[n-k+1]):`) :
print(`This is a generalization to  higher-order difference equation of procedure Orb(f,x,x0,K1,K2). For example, try:`) :
print(`Orbk(1,z,5/2*z[1]*(1-z[1]),[0.5],1000,1010);   `) :
print(`To get the Fibonacci sequence, type:`) :
print(`Orbk(2,z,z[1]+z[2],[1,1],1000,1010);`) :
print(``) :
print(`To get the part of the orbit between n=1000 and n=1010, of the 3rd order recurrence given in Eq. (4) of the Ladas-Amleh paper`) :
print(`https://sites.math.rutgers.edu/~zeilberg/Bio21/AmlehLadas.pdf`) :
print(`with initial conditions x(0)=1, x(1)=3, x(2)=5, Type: `) :
print(`Orbk(3,z,z[2]/(z[2]+z[3]),[1.,3.,5.],1000,1010);`) :

print(``) :
print(`To get the part of the orbit between n=1000 and n=1010, of the 3rd order recurrence given in Eq. (5) of the Ladas-Amleh paper`) :
print(`with initial conditions x(0)=1, x(1)=3, x(2)=5, Type: `) :
print(`Orbk(3,z,(z[1]+z[3])/z[2],[1.,3.,5.],1000,1010);`) :


print(``) :
print(`To get the part of the orbit between n=1000 and n=1010, of the 3rd order recurrence given in Eq. (6) of the Ladas-Amleh paper`) :
print(`with initial conditions x(0)=1, x(1)=3, x(2)=5, Type: `) :
print(`Orbk(3,z,(1+z[3])/z[1],[1.,3.,5.],1000,1010);`) :
```

```
print(``):
    print(`To get the part of the orbit between n=1000 and n=1010, of the 3rd order recurrence
    given in Eq. (7) of the Ladas-Amleh paper`):
print(`with initial conditions x(0)=1, x(1)=3, x(2)=5, Type: `):
print(`Orbk(3,z,(1+z[1])/(z[2]+z[3]),[1.,3.,5.],1000,1010);`):



elif nargs = 1 and args[1] = PhaseDiag then
    print(`PhaseDiag(F,x,pt,h,A): Inputs a transformation F in the list of variables x (of length
    2), i.e. a mapping from R^2 to R^2 gives the`):
print(`The phase diagram of the solution with initial condition x(0)=pt`):
print(`dx/dt=F[1](x(t)) by a discrete time dynamical system with step-size h from t=0 to t=A`):
print(`Try:`):
print(`PhaseDiag([x*(1-y),y*(1-x)],[x,y],[0.5,0.5], 0.01, 10);`):


elif nargs = 1 and args[1] = PhaseDiagE then
    print(`PhaseDiagE(F,x,pt,h,A): Inputs a transformation F in the list of variables x (of length
    2), i.e. a mapping from R^2 to R^2 gives the`):
print(`The phase diagram of the solution with initial condition x(0)=pt`):
print(`dx/dt=F[1](x(t)) using dsolve. It should only be used for linear system`):
print(`Try:`):
print(`PhaseDiagE([y,-x],[x,y],[0,1],10);`):

elif nargs = 1 and args[1] = RandNice then
    print(`RandNice(x,K): A random transformation in the set of variables x where each
    component if a a product of two affine-linear expressions.`):
print(`To  generate examples of continuout time dynamical systems`):
print(`Try: RandNice([x,y],100); `):

elif nargs = 1 and args[1] = RT then
    print(`RT(var,K): A random rational transformation of numerator and denominator degrees
    1 from R^k to R^k (where k=nops(var), with postiive integer coefficients from 1 to K The
    inpus are a list of variables x and a posi. integer K`):
print(`is for generating examples. Try:`):
print(`RT([x,y],10);`):


elif nargs = 1 and args[1] = SEquP then
    print(`SEquP(F,x): Given a transformation F in the list of variables finds all the Stable
    Equilibrium points of   the continuous-time dynamical system x'(t)=F(x(t))`):
print(`SEquP([5/2*x*(1-x)],[x]]);`):
print(`SEquP([y*(1-x-y),x*(3-2*x-y)],[x,y]]);`):

elif nargs = 1 and args[1] = SFP then
    print(`SFP(F,x): Given a transformation F in the list of variables finds all the STABLE fixed
    point of the transformation x->F(x), i.e. the set of solutions of`):
print(`the system {x[1]=F[1], ..., x[k]=F[k]}} that are stable. Try:`):
```

```
print(`SFP([5/2*x*(1-x)],[x]]);`) :
print(`SFP([(1+x+y)/(2+3*x+y),  (3+x+2*y)/(5+x+3*y)],[x,y]]);`) :

elif nargs = 1 and args[1] = SIRS then
    print(`SIRS(s,i,beta,gamma,nu,N): The SIRS dynamical model with parameters beta,gamma, nu,N (see  section 6.6 of Edelstein-Keshet), s is the number of`) :
    print(`Susceptibles, i is the number of infected, (the number of removed is given by N-s-i). N is the total population. Try:`) :
print(`SIRS(s,i,beta,gamma,nu,N);`) :


elif nargs = 1 and args[1] = SIRSdemo then
    print(`SIRSdemo(N,IN,gamma,nu,h,A): A demonstartion of the SIRS model with NUMBERS N: The total population, IN: The number of infected individuals at the start`) :
    print(`parameters gamma, and nu and various beta changing from 0.1*(nu/N) to 4*(nu/N) . Using a discretization with mesh size h and going until t=A.`) :
print(`Try:`) :
print(`SIRSdemo(1000,200,1,1,0.01,10);`) :


elif nargs = 1 and args[1] = TimeSeries then
print(`TimeSeries(F,x,pt,h,A,i): Inputs a transformation F in the list of variables x`) :
    print(`The time-series of x[i] vs. time of the Dynamical system approximating the  the autonomous  continuous dynamical  process `) :
print(`dx/dt=F(x(t)) by a discrete time dynamical system with step-size h from t=0 to t=A`) :
print(`Try:`) :
print(`TimeSeries([x*(1-y),y*(1-x)],[x,y],[0.5,0.5], 0.01, 10,1);`) :

elif nargs = 1 and args[1] = TimeSeriesE then
print(`TimeSeriesE(F,x,pt,A,i): Inputs a transformation F in the list of variables x, outputs`) :
    print(`The time-series of x[i] vs. time of the Dynamical system using the EXACT solutions via dsolve (note that it is usuall not possible)`) :
    print(`It works for linear transformations, and is a good check with the approximate TimeSeries(F,x,pt,h,A,i) that uses discretization with`) :
print(`dx/dt=F(x(t)) by a discrete time dynamical system with step-size h from t=0 to t=A`) :
print(`Try:`) :
print(`TimeSeriesE([y,-x],[x,y],[1,0], 10,1);`) :

elif nargs = 1 and args[1] = ToSys then
    print(`ToSys(k,z,f): converts the kth order difference equation x(n)=f(x[n-1],x[n-2],...x[n-k]) to a first-order system`) :
    print(`x1(n)=F(x1(n-1),x2(n-1), ...,xk(n-1)), it gives the unerlying transormation, followed by the set of variables`) :
print(`Try:`) :
print(`ToSys(2,z,z[1]+z[2]);`) :

elif nargs = 1 and args[1] = Volterra then
    print(`Volterra(a,b,c,d,x,y): The (simple, original) Volterra predator-prey continuous-time dynamical system with parameters a,b,c,d`) :
```

```
print(` Given by Eqs. (7a) (7b) in Edelstein-Keshet p. 219 (section 6.2).`) :
print(`a,b,c,d may be symbolic or numeric`) :
print(`Try: `) :
print(`Volterra(a,b,c,d,x,y);`) :
print(`Volterra(1,2,3,4,x,y);`) :


elif nargs = 1 and args[1] = VolterraM then
      print(`VolterraM(a,b,c,d,x,K,y): The MODIFIED Volterra predator-prey continuous-time
      dynamical system with parameters a,b,c,d,K`) :
print(` Given by Eqs. (8a) (8b) in Edelstein-Keshet p. 220 (section 6.2).`) :
print(`a,b,c,d ,Kmay be symbolic or numeric`) :
print(`Try: `) :
print(`VolterraM(a,b,c,d,K,x,y);`) :
print(`VolterraM(1,2,3,4,3,x,y);`) :

else
 print(`There is no such thing as`, args) :

fi:


end:



      #Orb(F,x,x0,K1,K2): Inputs a transformation F in the list of variables x with initial point pt,
      outputs the trajectory

      #of the discrete dynamical system (i.e. solutions of the difference equation): x(n)=F(x(n-1))
      with x(0)=x0 from n=K1 to n=K2.
#For the full trajectory (from n=0 to n=K2), use K1=0. Try:
#Orb(5/2*x*(1-x),[x], [0.5], 1000,1010);
#Orb((1+x+y)/(2+x+y),[x,y], [2.,3.], 1000,1010);
Orb := proc(F, x, x0, K1, K2) local x1, i, L, i1, i2 :

if not (type(F, list) and type(x, list) and type(x0, list) and nops(F) = nops(x) and  nops(x)
       = nops(x0) and type(K1, integer) and type(K2, integer) and K1 ≥ 0 and K1 < K2) then
 print(`bad input`) :
 RETURN(FAIL) :
fi:

x1 := x0 :

for i from 0 to K1 - 1 do
x1 := [seq(subs( {seq(x[i2] = x1[i2], i2 = 1 ..nops(x))}, F[i1]), i1 = 1 ..nops(F))] :
od:

L := [x1] :
```

```
for i from K1 to K2 do
x1 := [seq(subs( {seq(x[i2]=x1[i2], i2=1..nops(x))}, F[i1]), i1=1..nops(F))]:
 L := [op(L), x1]: #we append it to the list
od:

L: #that's the output

end:


#OrbF(F,x,x0,K1,K2):  Same as Orb(F,x,x0,K1,K2) but in floating-point
#Inputs a transformation F in the list of variables x with initial point pt, outputs the trajectory

    #of the discrete dynamical system (i.e. solutions of the difference equation): x(n)=F(x(n-1))
    with x(0)=x0 from n=K1 to n=K2.
#For the full trajectory (from n=0 to n=K2), use K1=0. Try:
#OrbF(5/2*x*(1-x),[x], [0.5], 1000,1010);
#OrbF((1+x+y)/(2+x+y),[x,y], [2.,3.], 1000,1010);
OrbF := proc(F, x, x0, K1, K2) local x1, i, L, i1, i2:

if not (type(F, list) and type(x, list) and type(x0, list) and nops(F) = nops(x) and  nops(x)
     = nops(x0) and type(K1, integer) and type(K2, integer) and K1 ≥ 0 and K1 < K2) then
 print(`bad input`):
 RETURN(FAIL):
fi:

x1 := x0:

for i from 0 to K1-1 do
x1 := evalf([seq(subs( {seq(x[i2]=x1[i2], i2=1..nops(x))}, F[i1]), i1=1..nops(F))]):
od:

L := [x1]:

for i from K1 to K2 do
x1 := evalf([seq(subs( {seq(x[i2]=x1[i2], i2=1..nops(x))}, F[i1]), i1=1..nops(F))]):
 L := [op(L), x1]: #we append it to the list
od:

L: #that's the output

end:



    #FP(F,x): Given a transformation F in the list of variables finds all the fixed point of the
    transformation x->F(x), i.e. the set of solutions of
#the system {x[1]=F[1], ..., x[k]=F[k]}. Try:
#FP([5/2*x*(1-x)],[x]);
```

```
#FP([(1+x+y)/(2+3*x+y),  (3+x+2*y)/(5+x+3*y)],[x,y]]);
FP :=proc(F, x) local i, sol :
if not (type(F, list) and type(x, list) and nops(F) = nops(x)) then
 print(`bad input`) :
 RETURN(FAIL) :
fi:

sol := {solve({seq(F[i] = x[i], i = 1 ..nops(F))}, {op(x)}, allsolutions = true)} :

{seq(subs(sol[i], x), i = 1 ..nops(sol))} :

end:




    #RT(var,K): A random rational transformation of numerator and denominator degrees  1
    from R^k to R^k (where k=nops(var), with postiive integer coefficients from 1 to K The inpus
    are a list of variables x and a posi. integer K
#is for generating examples
#Try:
#RT([x,y],10);
RT :=proc(x, K) local ra, i, i1 :
if not (type(x, list) and type(K, integer) and K > 0) then
 print(`bad input`) :
 RETURM(FAIL) :
fi:

ra := rand(1 ..K) : #random integer from -K to K

[seq((ra() + add(ra() * x[i1], i1 = 1 ..nops(x))) / (ra() + add(ra() * x[i1], i1 = 1
    ..nops(x))), i = 1 ..nops(x))] :
end:



    #IsContStable(M): inputs a numeric matrix M (given as a list of lists M) and decides whether
    all ite eigenvalues have real negative part. Try
#IsContStable(Matrix([[1,-1],[-1,1]]));
IsContStable :=proc(M) local Ei1, i :
#k:=nops(M):
Ei1 := Eigenvalues(evalf(Matrix(M))) :
evalb(max(seq(coeff(Ei1[i], I, 0), i = 1 ..nops(M))) < 0):
end:




    #IsDisStable(M): inputs a numeric matrix M (given as a list of lists M) and decides whether
    all ite eigenvalues have absolute value less than 1
```

```
#IsDisStable(Matrix([[1,-1],[-1,1]]));
IsDisStable :=proc(M) local Ei1, i :
Ei1 := Eigenvalues(evalf(Matrix(M) ) ) :
evalb(max(seq(abs(Ei1[i]), i=1 ..nops(M) ) ) < 1):
end:



    #JAC(F,x): The Jacobian Matrix (given as a list of lists) of the transformation F in the list of
    variables x. Try:
#JAC([x+y,x^2+y^2],[x,y]):

JAC :=proc(F, x) local i, j :
if not (type(F, list) and type(x, list) and nops(F) =nops(x) ) then
 print( `Bad input`) :
 RETURN(FAIL) :
fi:

normal( [seq( [seq(diff(F[i], x[j]), j=1 ..nops(x) ) ], i=1 ..nops(F) ) ]) :

end:



    #SFP(F,x): Given a transformation F in the list of variables finds all the STABLE fixed point
    of the transformation x->F(x), i.e. the set of solutions of
#the system {x[1]=F[1], ..., x[k]=F[k]}} that are stable. Try:
#SFP([5/2*x*(1-x)],[x]]);
#SFP([(1+x+y)/(2+3*x+y),  (3+x+2*y)/(5+x+3*y)],[x,y]]);
SFP :=proc(F, x) local i, Fi, St, J, J0, pt :
if not (type(F, list) and type(x, list) and nops(F) =nops(x) ) then
 print( `bad input`) :
 RETURN(FAIL) :
fi:
Fi := evalf(FP(F, x) ) :    #Fi is the set of fixed points in floating-point

St := { } :     #St is the set of stable fixed points, that starts out empty

J := JAC(F, x) :   #The general Jacobian in terms of the list of variables x

for pt in Fi do       #we examine each fixed point, one at a time
J0 := subs( {seq(x[i]=pt[i], i=1 ..nops(x) ) }, J) :
    #J0 is the NUMETRICAL Jacobian matrix evaluated at the examined fixed point

if IsDisStable(J0) then
 St := St union{pt} :                  #if it is stable we include it
fi:
```

```
od:

St :        #The output is the set of all the successful fixed points that happened to be stable
end:



    #Orbk(k,z,f,INI,K1,K2): Given a positive integer k, a letter (symbol), z, an expression f of z
    [1], ..., z[k] (representing a multi-variable function of the variables z[1],...,z[k]

    #a vector INI representing the initial values [x[1],..., x[k]], and (in applications) positive
    integres K1 and K2, outputs the

    #values of the sequence starting at n=K1 and ending at n=K2. of the sequence satisfying the
    difference equation
##x[n]=f(x[n-1],x[n-2],..., x[n-k+1]):

    #This is a generalization to  higher-order difference equation of procedure Orb(f,x,x0,K1,K2)
    . For example
#Orbk(1,z,5/2*z[1]*(1-z[1]),[0.5],1000,1010);  should be the same as
#Orb(5/2*z[1]*(1-z[1]),z[1],[0,5],1000,1010);
#Try:
#Orbk(2,z,(5/4)*z[1]-(3/8)*z[2],[1,2],1000,1010);

Orbk := proc(k, z, f, INI, K1, K2) local L, i, newguy :
L := INI :  #We start out with the list of initial values

if not (type(k, integer) and type(z, symbol) and type(INI, list) and nops(INI) = k and type(K1,
    integer) and type(K2, integer) and K1 > 0 and K2 > K1) then
    #checking that the input is OK
 print(`bad input`) :
 RETURN(FAIL) :
fi:

while nops(L) < K2 do
 newguy := subs({seq(z[i] = L[-i], i = 1 ..k)}, f) :
    #Using what we know about the value yesterday, the day before yesterday, ... up to k days
    before yesterday we find the value of the sequence today
 L := [op(L), newguy] :  #we append the new value to the running list of values of our sequence
od:

[op(K1 ..K2, L)] :

end:



    #ToSys(k,z,f): converts the kth order difference equation x(n)=f(x[n-1],x[n-2],...x[n-k]) to a
    first-order system
```

```
#x1(n)=F(x1(n-1),x2(n-1), ...,xk(n-1)), it gives the unerlying transormation, followed by the set of variables
#x2(n)=x1(n-1)
#Try:
#ToSys(2,z,z[1]+z[2]);
ToSys := proc(k, z, f) local i :
[f, seq(z[i-1], i=2..k)], [seq(z[i], i=1..k)] :
end:
```

```
#HW3(u,v,w): The Hardy-Weinberg unerlying transformation witu (u,v,w), Eqs. (53a,53b, 53c) in Edelestein-Keshet Ch. 3
HW3 := proc(u, v, w) : [u^2 + u*v + (1/4)*v^2,  u*v + 2*u*w + 1/2*v^2 + v*w, 1/4*v^2 + v*w + w^2] :end:
```

```
#HW(u,v): The Hardy-Weinberg unerlying transformation witu (u,v,w), Eqs. (53a,53b,53c) in Edelestein-Keshet Ch. 3 using the fact that u+v+w=1
HW := proc(u, v) : expand([u^2 + u*v + (1/4)*v^2 , u*v + 2*u*(1-u-v) + 1/2*v^2 + v*(1-u-v)]), [u, v] :end:
```

```
#HW3g(u,v,w,M): The Hardy-Weinberg unerlying transformation with (u,v,w), GENERALIZED Eqs. with the 3 by 3 matrix M  (53a,53b,53c) in Edelestein-Keshet Ch. 3
#Based on Anne Somalwar's solution of the bonus problem from hw15, see the end of
#from https://sites.math.rutgers.edu/~zeilberg/Bio21/HW15posted/hw15AnneSomalwar.pdf
HW3g := proc(u, v, w, M) local tot, LI :
LI := [

M[1][1]*u^2 + (M[1][2] + M[2][1])/2*u*v + M[2][2]*(1/4)*v^2,

(M[1][2] + M[2][1])/2*u*v + (M[1][3] + M[3][1])*u*w + M[2][2]/2*v^2 + (M[2][3] + M[3][2])/2*v*w,

M[2][2]*1/4*v^2 + (M[2][3] + M[3][2])/2*v*w  + M[3][3]*w^2] :
tot := LI[1] + LI[2] + LI[3] :
[LI[1]/tot, LI[2]/tot, LI[3]/tot] :
end:
```

```
#HWg(u,v,M): The Generalized Hardy-Weinberg unerlying transformation with (u,v), M is the survival matrix. Based on Ann Somalwar's HW3g(u,v,w) (only retain the first two
```

```
components and replace w by 1-u-v)
HWg :=proc(u, v, M) local LI, w :
LI := HW3g(u, v, w, M) :
normal(subs(w = 1 - u - v, [LI[1], LI[2]])) :
end:



#RandNice(x,K): A random transformation in the set of variables x where each component if
a a product of two affine-linear expressions.
#To  generate examples
#Try: RandNice([x,y],100);
RandNice :=proc(x, K) local ra, i :
ra := rand(1 ..K) :
[seq((ra( ) - add(ra( ) * x[i], i = 1 ..nops(x))) * (ra( ) - add(ra( ) * x[i], i = 1 ..nops(x))), i = 1
..nops(x))] :
end:



#EquP(F,x): Given a transformation F in the list of variables finds all the Equilibrium points
of   the continuous-time dynamical system x'(t)=F(x(t))
#EquP([5/2*x*(1-x)],[x]]);
#EquP([y*(1-x-y),x*(3-2*x-y)],[x,y]]);
EquP :=proc(F, x) local i, sol :
if not (type(F, list) and type(x, list) and nops(F) = nops(x)) then
 print(`bad input`) :
 RETURN(FAIL) :
fi:

sol := {solve({op(F)}, {op(x)}, allsolutions = true)} :

{seq(subs(sol[i], x), i = 1 ..nops(sol))} :

end:



#SEquP(F,x): Given a transformation F in the list of variables x describing the
CONTINUOUS-time dynamical system x'(t)=F(x(t))
#Finds the set of Stable Equilibria. Try:
#SEquP([y*(1-x-y),x*(3-2*x-y)],[x,y]]);
SEquP :=proc(F, x) local i, Fi, St, J, J0, pt :
if not (type(F, list) and type(x, list) and nops(F) = nops(x)) then
 print(`bad input`) :
```

```
 RETURN(FAIL) :
fi:
Fi := evalf(EquP(F, x)) :    #Fi is the set of equilibrium points in floating-point

St := { } :    #St is the set of stable fixed points, that starts out empty

J := JAC(F, x) :   #The general Jacobian in terms of the list of variables x

for pt in Fi do      #we examine each fixed point, one at a time
J0 := subs({seq(x[i]=pt[i], i=1 ..nops(x))}, J) :
     #J0 is the NUMETRICAL Jacobian matrix evaluated at the examined fixed point

if IsContStable(J0) then
 St := St union{pt} :                 #if it is stable we include it
fi:

od:

St :      #The output is the set of all the successful fixed points that happened to be stable
end:


#Dis(F,x,pt,h,A): Inputs a transformation F in the list of variables x

     #The approximate orbit of the Dynamical system approximating the  the autonomous
     continuous dynamical  process
#dx/dt=F[1](x(t)) by a discrete time dynamical system with step-size h from t=0 to t=A
#Try:
#Dis([x*(1-y),y*(1-x)],[x,y],[0.5,0.5], 0.01, 10);
Dis :=proc(F, x, pt, h, A) local L, i :


if not (type(F, list) and type(x, list) and type(pt, list) and nops(F) = nops(x) and nops(F)
      = nops(pt) and type(h, numeric) and h ≤ 0.1 and type(A, numeric) and A > 0) then
 print(`bad input`) :
 RETURN(FAIL) :
fi:

L := Orb([seq(x[i] + h * F[i], i=1 ..nops(F))], x, pt, 0, trunc(A/h)) :

L := [seq([i * h, L[i]], i=1 ..nops(L))] :
end:
```

#SIRS(s,i,beta,gamma,nu,N): The SIRS dynamical model with parameters beta,gamma, nu,N (see section 6.6 of Edelstein-Keshet), s is the number of

```
#Susceptibles, i is the number of infected, (the number of removed is given by N-s-i). N is the total population
SIRS :=proc(s, i, beta, gamma, nu, N) : [-beta*s*i + gamma*(N-s-i), beta*s*i-nu*i] :
end:



#SIRSdemo(N,IN,gamma,nu,h,A): A demonstartion of the SIRS model with NUMBERS N: The total population, IN: The number of infected individuals at the start

#parameters gamma, and nu and various beta changing from 0.1*(nu/N) to 4*(nu/N) . Using a discretization with mesh size h and going until t=A.
#Try:
#SIRSdemo(1000,200,1,1,0.01,10);

SIRSdemo :=proc(N, IN, gamma, nu, h, A) local s, i, L, beta, i1 :
print(`This is a numerical demonstration of the R0 phenomenon in the SIRS model using discretization with mesh size=`, h, `and letting it run until time t=`, A) :
print(`with population size`, N, `and fixed parameters nu=`, nu, `and gamma=`, gamma) :
print(`where we change beta from 0.2*nu/N to 4*nu/N `) :
print(`Recall that the epidemic will persist if beta exceeds nu/N, that in this case is`, nu/N) :
print(`We start with `, IN, `infected individuals, 0 removed and hence`, N-IN, ` susceptible `) :
print(`We will show what happens once time is close to`, A) :
for i1 from 1 by 2 to 40 do
beta := i1/10*(nu/N) :
print(`beta is `, i1/10, `times the threshold value`) :
L := Dis(SIRS(s, i, beta, gamma, nu, N), [s, i], [N-IN, IN], h, A) :
print(`the long-term behavior is`) :
print([op(nops(L)-3 ..nops(L), L)]) :
od:

end:




#TimeSeries(F,x,pt,h,A,i): Inputs a transformation F in the list of variables x

#The time-series of x[i] vs. time of the Dynamical system approximating the  the autonomous continuous dynamical  process
#dx/dt=F[1](x(t)) by a discrete time dynamical system with step-size h from t=0 to t=A
#Try:
#TimeSeries([x*(1-y),y*(1-x)],[x,y],[0.5,0.5], 0.01, 10,1);
TimeSeries :=proc(F, x, pt, h, A, i) local L, i1 :

if not (type(F, list) and type(x, list) and type(pt, list) and nops(F) = nops(x) and nops(F) = nops(pt) and type(h, numeric) and h ≤ 0.1 and type(A, numeric) and A > 0 and 1 ≤ i and i ≤ nops(x) ) then
 print(`bad input`) :
```

```
 RETURN(FAIL) :
fi:

L := Dis(F, x, pt, h, A) :
plot( [seq( [L[i1][1], L[i1][2][i]], i1 = 1 ..nops(L) ) ]) :
end:
```

```
    #PhaseDiag(F,x,pt,h,A): Inputs a transformation F in the list of variables x (of length 2), i.e.
    a mapping from R^2 to R^2 gives the
#The phase diagram of the solution with initial condition x(0)=pt
#dx/dt=F[1](x(t)) by a discrete time dynamical system with step-size h from t=0 to t=A
#Try:
#PhaseDiag([x*(1-y),y*(1-x)],[x,y],[0.5,0.5], 0.01, 10);
PhaseDiag :=proc(F, x, pt, h, A) local L, i1 :

if not (type(F, list) and type(x, list) and type(pt, list) and nops(F) = nops(x) and nops(F)
     = nops(pt) and nops(x) = 2 and type(h, numeric) and h ≤ 0.1 and type(A, numeric) and A
     > 0) then
 print(`bad input`) :
 RETURN(FAIL) :
fi:

L := Dis(F, x, pt, h, A) :
plot( [seq(L[i1][2], i1 = 1 ..nops(L) ) ], style = point) :
end:
```

```
    #ComK(F,x,K): inputs a transformation F in the list of variables x, outputs the composition of
    F with itself K times. Try:
#ComK([k*x*(1-x)],[x],2);
#ComK([x*(1-y),y*(1-x)],[x,y],4);

ComK :=proc(F, x, K) local F1, i :
option remember :
if K = 0 then
 RETURN(x) :
elif K = 1 then
 RETURN(F) :
else
F1 := ComK(F, x, K - 1) :
RETURN(normal(subs( {seq(x[i] = F[i], i = 1 ..nops(x) ) }, F1) ) ) :
fi:

end:
```

```
#AllenSIR(a,b,c,x,y): The Linda Allen discrete SIR model given in https://sites.math.rutgers.edu/~zeilberg/Bio21/LadasSri.pdf
#with parameters a,b,c. try:
#AllenSIR(1,1/3,1/3,x,y);
AllenSIR :=proc(a, b, c, x, y)
[x* (1-b-c) + y* (1-exp(-a*x)), (1-y) * b + y* exp(-a*x)] :
end:


#TimeSeriesE(F,x,x0,A,i): Inputs a transformation F in the list of variables x, outputs

#The time-series of x[i] vs. time of the Dynamical system using the exact solutions via dsolve (note that it is usuall not possible)

#It works for linear transformations, and is a good check with the approximate TimeSeries(F,x,x0,h,A,i) that uses discretization with
#dx/dt=F[1](x(t)) by a discrete time dynamical system with step-size h from t=0 to t=A
#Try:
#TimeSeriesE([y,-x],[x,y],[0,1],  10,1);
TimeSeriesE :=proc(F, x, x0, A, i) local sol, t, i1, F1 :
if not (type(F, list) and type(x, list) and type(x0, list) and nops(F) = nops(x) and nops(F)
   = nops(x0) and type(A, numeric) and A > 0 and 1 <= i and i <= nops(x) ) then
 print(`bad input`) :
 RETURN(FAIL) :
fi:

F1 := subs({seq(x[i1] = X[i1](t), i1 = 1 ..nops(x))}, F) :
sol := dsolve({seq(diff(X[i1](t), t) = F1[i1], i1 = 1 ..nops(x)), seq(X[i1](0) = x0[i1], i1 = 1
   ..nops(x0))}) :

plot(subs(sol, X[i](t)), t = 0 ..A) :

end:



#PhaseDiagE(F,x,x0,A): Inputs a transformation F in the PAIR of variables x, outputs

#The Phase diagram  [x[1],x[2]]  (forgetting about time, that becomes a parameter) of the Dynamical system using the exact solutions via dsolve (note that it is usuall not possible)

#It works for linear transformations, and is a good check with the approximate TimeSeries(F,x,x0,h,A,i)
#Try:
#TimeSeriesE([y,-x],[x,y],[0,1],  10);
PhaseDiagE :=proc(F, x, x0, A) local sol, t, i1, X, F1 :

if not (type(F, list) and type(x, list) and nops(x) = 2 and type(x0, list) and nops(F) = nops(x)
   and nops(F) = nops(x0) and type(A, numeric) and A > 0 ) then
```

```
print(`bad input`) :
RETURN(FAIL) :
fi:

F1 := subs({seq(x[i1] = X[i1](t), i1 = 1 ..nops(x))}, F) :
sol := dsolve({seq(diff(X[i1](t), t) = F1[i1], i1 = 1 ..nops(x)), seq(X[i1](0) = x0[i1], i1 = 1
      ..nops(x0))}) :

plot([subs(sol, X[1](t)), subs(sol, X[2](t)), t = 0 ..A]) :

end:
```

#ChemoStat(N,C,a1,a2): The Chemostat continuous-time dynamical system with N=Bacterial poplulation densitty, and C=nutient Concentration in growth chamber (see Table 4.1 of Edelstein-Keshet, p. 122)

#with paramerts a1, a2, Equations (19a_, (19b) in Edelestein-Keshet p. 127 (section 4.5, where they are called alpha1, alpha2). a1 and a2 can be symbolic or numeric. Try:
#ChemoStat(N,C,a1,a2);
#ChemoStat(N,C,2,3);

```
ChemoStat := proc(N, C, a1, a2) :
[a1 * C / (1 + C) * N - N, -C / (1 + C) * N - C + a2] :
end:
```

#Volterra(a,b,c,d,x,y): The (simple, original) Volterra predator-prey continuous-time dynamical system with parameters a,b,c,d
#Eqs. (7a) (7b) in Edelstein-Keshet p. 219 (section 6.2)
#a,b,c,d may be symbolic or numeric
#Try:
#Volterra(a,b,c,d,x,y);
#Volterra(1,2,3,4,x,y);

```
Volterra := proc(a, b, c, d, x, y)
[a * x - b * x * y, -c * y + d * x * y] :
end:
```

#VolterraM(a,b,c,d,K,x,y): The modified Volterra predator-prey continuous-time dynamical system with parameters a,b,c,d,K
#Eqs. (8a) (8b) in Edelstein-Keshet p. 220 (section 6.2)
#a,b,c,d,K may be symbolic or numeric
#Try:
#VolterraM(a,b,c,d,K,x,y);

```
#VolterraM(1,2,3,4,2,x,y);
VolterraM :=proc(a, b, c, K, d, x, y)
[a * x * (1-x/K) - b * x * y, -c * y + d * x * y] :
end:



    #Lotka(r1,k1,r2,k2,b12,b21,N1,N2): The Lotka-Volterra continuous-time dynamical system,
    Eqs. (9a),(9b)  (p. 224, section 6.3) of Edelstein-Keshet

    #with popoluations N1, N2, and parameters r1,r2,k1,k2, b12, b21 (called there beta_12 and
    beta_21)
#Try:
#Lotka(r1,k1,r2,k2,b12,b21,N1,N2);
#Lotka(1,2,2,3,1,2,N1,N2);

Lotka :=proc(r1, k1, r2, k2, b12, b21, N1, N2) :
[r1 * N1 * (k1-N1-b12 * N2) / k1, r2 * N2 * (k2-N2-b21 * N1) / k2] :
end:



    #GeneNet(a0,a,b,n,m1,m2,m3,p1,p2,p3): The contiuous-time dynamical system, with
    quantities m1,m2,m3,p1,p2,p3, due to M. Elowitz and S. Leibler
#described in the Ellner-Guckenheimer book, Eq. (4.1) (chapter 4,  p. 112)
#and parameers a0 (called alpha_0 there),a (called alpha there), b (called beta there) and n. Try:
#GeneNet(0,0.5,0.2,2,m1,m2,m3,p1,p2,p3);
GeneNet :=proc(a0, a, b, n, m1, m2, m3, p1, p2, p3) :
[-m1 + a / (1 + p3^n) + a0, -m2 + a / (1 + p1^n) + a0, -m3 + a / (1 + p2^n) + a0, -b
    * (p1-m1), -b * (p2-m2), -b * (p3-m3)] :
end:
```

First Written: Nov. 2021

This is DMB.txt, A Maple package to explore Dynamical models in Biology (both discrete and continuous)

accompanying the class Dynamical Models in Biology, Rutgers University. Taught by Dr. Z. (Doron Zeilbeger)

The most current version is available on WWW at:

http://sites.math.rutgers.edu/~zeilberg/tokhniot/DMB.txt .

Please report all bugs to: DoronZeil at gmail dot com .

For general help, and a list of the MAIN functions,

type "Help();". For specific help type "Help(procedure_name);"

For a list of the supporting functions type: Help1();

For help with any of them type: Help(ProcedureName);

For a list of the functions that give examples of Discrete-time dynamical systems (some famous), type: HelpDDM();

For help with any of them type: Help(ProcedureName);

For a list of the functions continuous-time dynamical systems (some famous) type: HelpCDM();

For help with any of them type: Help(ProcedureName);

------------------------------ **(1)**

```
> #1: ChemoStat
  Help(ChemoStat);
```

ChemoStat(N,C,a1,a2): The Chemostat continuous-time dynamical system with N=Bacterial poplulation densitty, and C=nutient Concentration in growth chamber (see Table 4.1 of Edelstein-Keshet, p. 122)

with paramerts a1, a2, Equations (19a_, (19b) in Edelestein-Keshet p. 127 (section 4.5, where they are called alpha1, alpha2). a1 and a2 can be symbolic or numeric. Try:

ChemoStat(N,C,a1,a2);

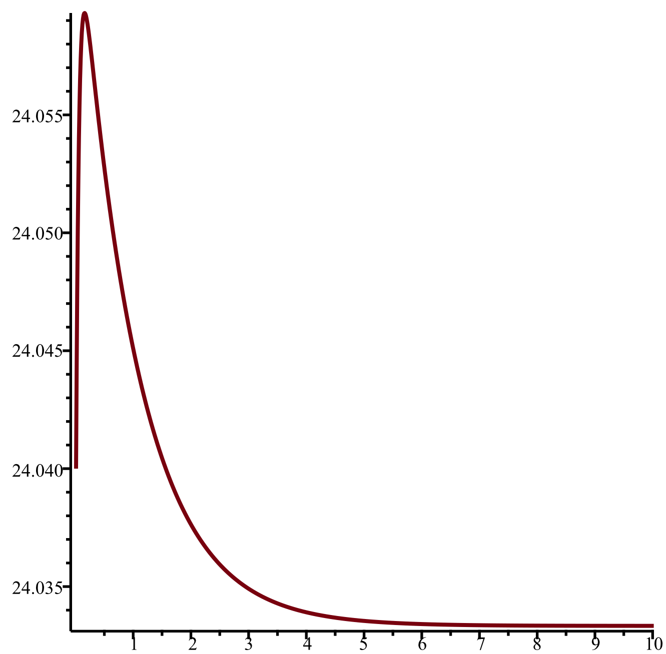
ChemoStat(N,C,2,3); **(2)**

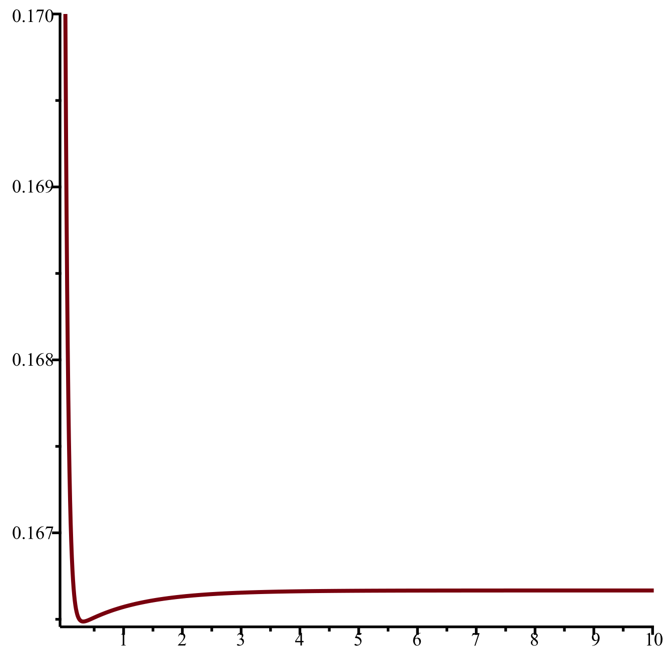
```
> f1 := ChemoStat(N, C, 7, 3.6);
  SEquP(f1, [N, C]);
```

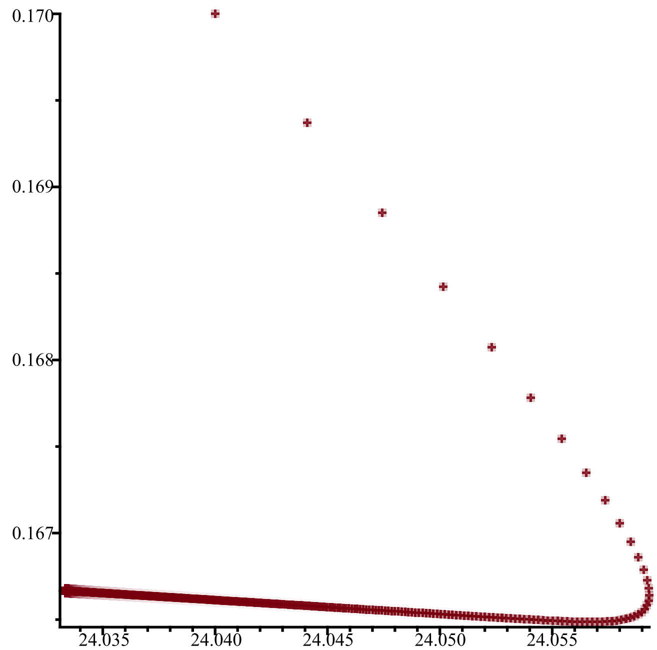
$$f1 := \left[\frac{7\,C\,N}{C+1} - N,\ -\frac{C\,N}{C+1} - C + 3.6\right]$$

$$\{[24.03333333, 0.1666666667]\}$$ **(3)**

```
> TimeSeries(f1, [N, C], [24.04, 0.17], 0.01, 10, 1);
  TimeSeries(f1, [N, C], [24.04, 0.17], 0.01, 10, 2);
  PhaseDiag(f1, [N, C], [24.04, 0.17], 0.01, 10);
```


24.055
24.050
24.045
24.040
24.035
1 2 3 4 5 6 7 8 9 10


0.170
0.169
0.168
0.167
1 2 3 4 5 6 7 8 9 10

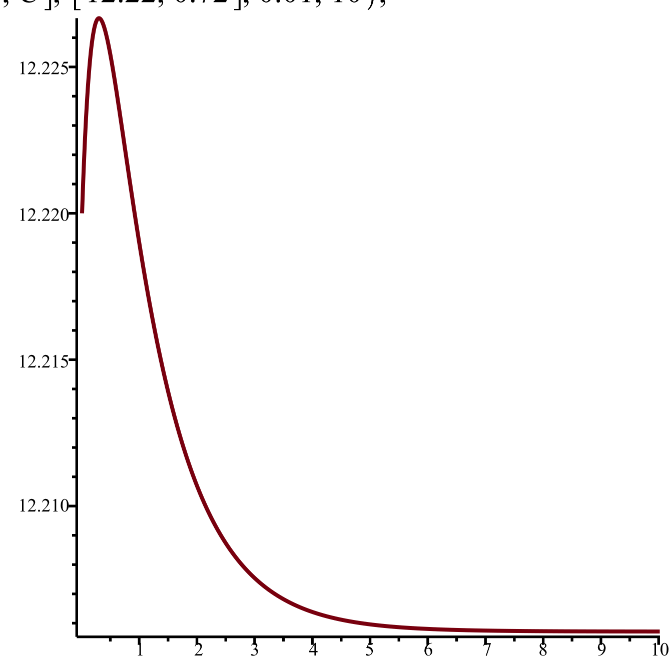


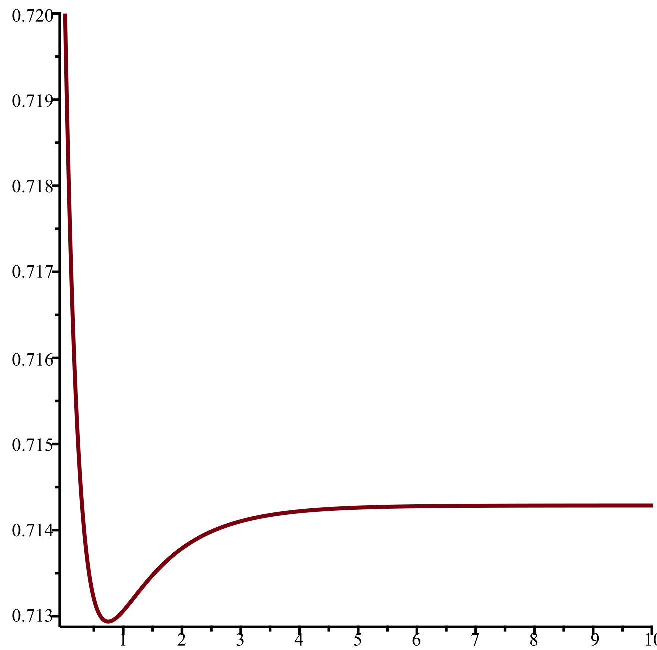

```
> f1 := ChemoStat(N, C, 2.4, 5.8);
  SEquP(f1, [N, C]);
```

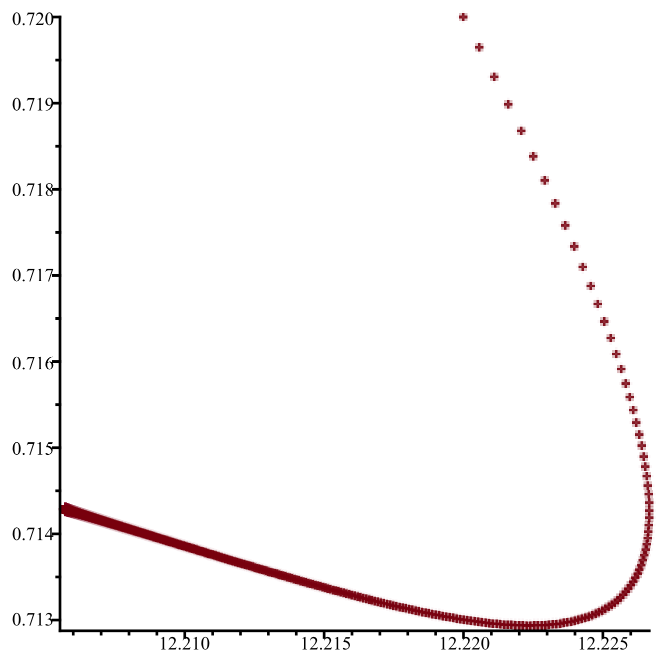
$$f1 := \left[\frac{2.4\, C\, N}{C+1} - N, -\frac{C\, N}{C+1} - C + 5.8\right]$$

$$\{[12.20571429, 0.7142857143]\} \quad \textbf{(4)}$$

```
> TimeSeries(f1, [N, C], [12.22, 0.72], 0.01, 10, 1);
  TimeSeries(f1, [N, C], [12.22, 0.72], 0.01, 10, 2);
  PhaseDiag(f1, [N, C], [12.22, 0.72], 0.01, 10);
```





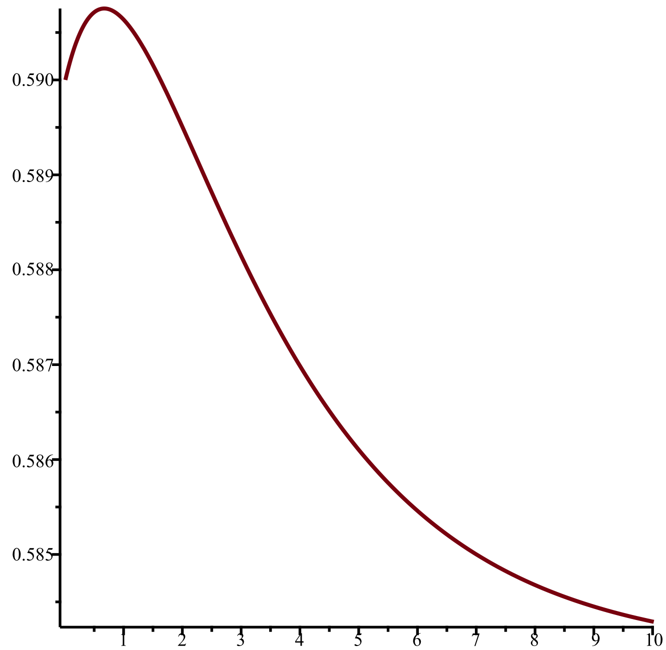


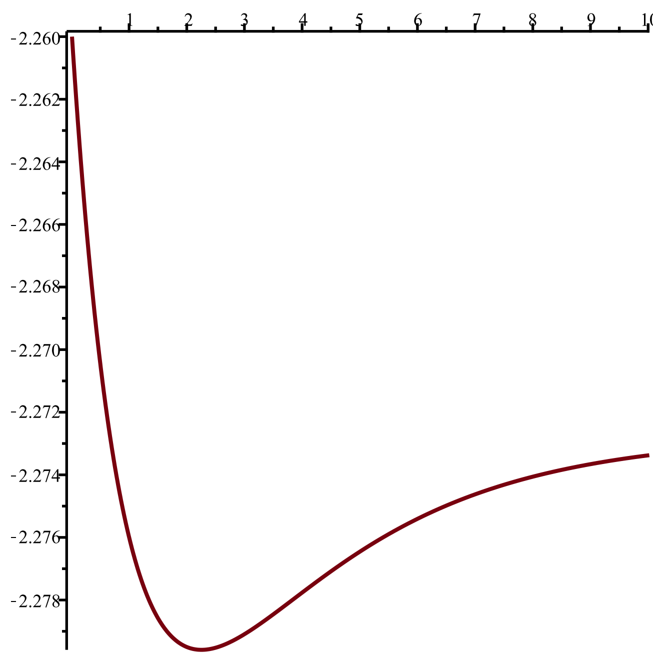
> *f1* := *ChemoStat*(*N*, *C*, 0.56, -1.23);
SEquP(*f1*, [*N*, *C*]);

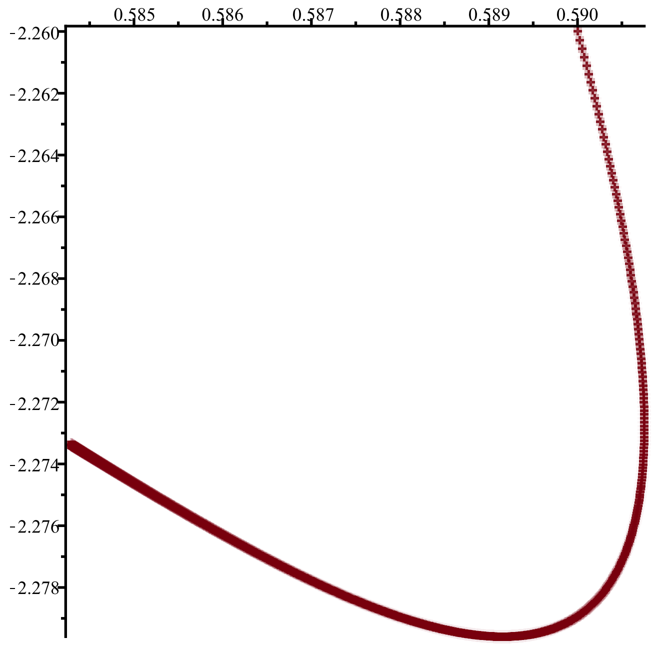
$$f1 := \left[\frac{0.56\, C\, N}{C+1} - N, -\frac{C\, N}{C+1} - C - 1.23\right]$$

$$\{[0.5839272727, -2.272727273]\} \quad \textbf{(5)}$$

> *TimeSeries*(*f1*, [*N*, *C*], [0.59, -2.26], 0.01, 10, 1);
TimeSeries(*f1*, [*N*, *C*], [0.59, -2.26], 0.01, 10, 2);
PhaseDiag(*f1*, [*N*, *C*], [0.59, -2.26], 0.01, 10);


0.590
0.589
0.588
0.587
0.586
0.585
1
2
3
4
5
6
7
8
9
10


1
2
3
4
5
6
7
8
9
10
-2.260
-2.262
-2.264
-2.266
-2.268
-2.270
-2.272
-2.274
-2.276
-2.278



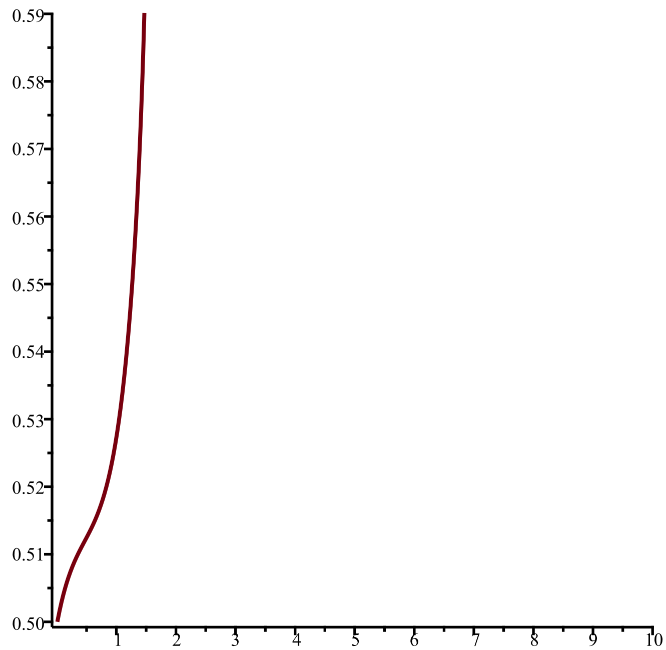
> *#2: GeneNet*
> *Help*(*GeneNet*);

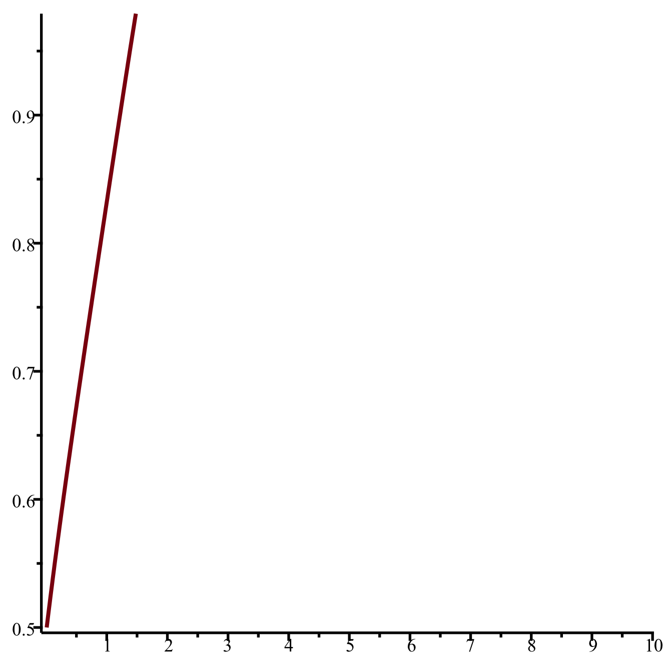
GeneNet(a0,a,b,n,m1,m2,m3,p1,p2,p3): The contiuous-time dynamical system, with quantities m1,m2,m3,p1,p2,p3, due to M. Elowitz and S. Leibler
described in the Ellner-Guckenheimer book, Eq. (4.1) (chapter 4, p. 112)
and parameers a0 (called alpha_0 there),a (called alpha there), b (called beta there) and n.
Try:
GeneNet(0,0.5,0.2,2,m1,m2,m3,p1,p2,p3); **(6)**

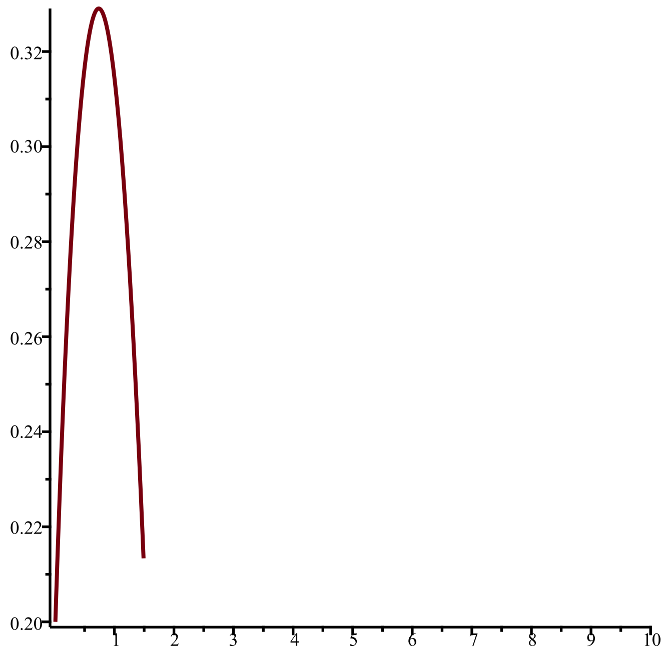
> $g1 := GeneNet(0.2, 1, 1.3, 0.5, 0.8, m1, m2, m3, p1, p2, p3);$

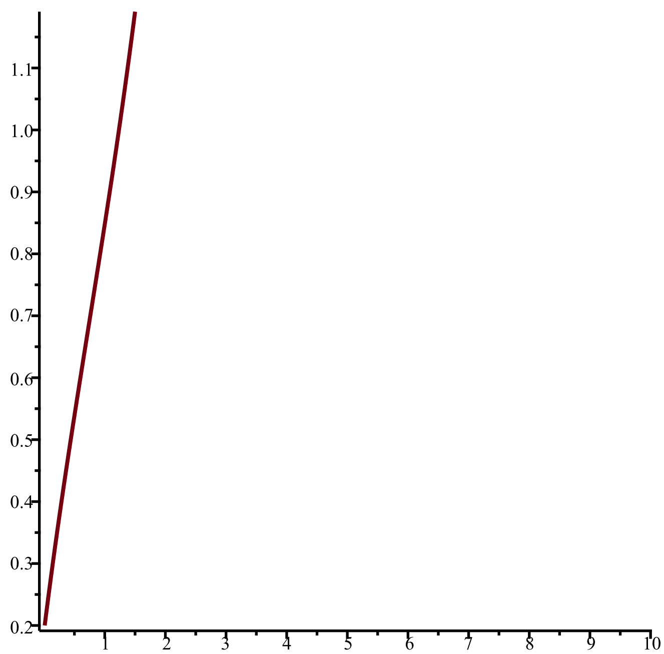
$$g1 := \left[-0.6 + \frac{1}{1+\sqrt{p2}}, -m1 + \frac{1}{1+\sqrt{m3}} + 0.2, -m2 + \frac{1}{1+\sqrt{p1}} + 0.2, -1.3\, m3 + 1.04, -1.3\, p1 + 1.3\, m1, -1.3\, p2 + 1.3\, m2\right] \quad (7)$$

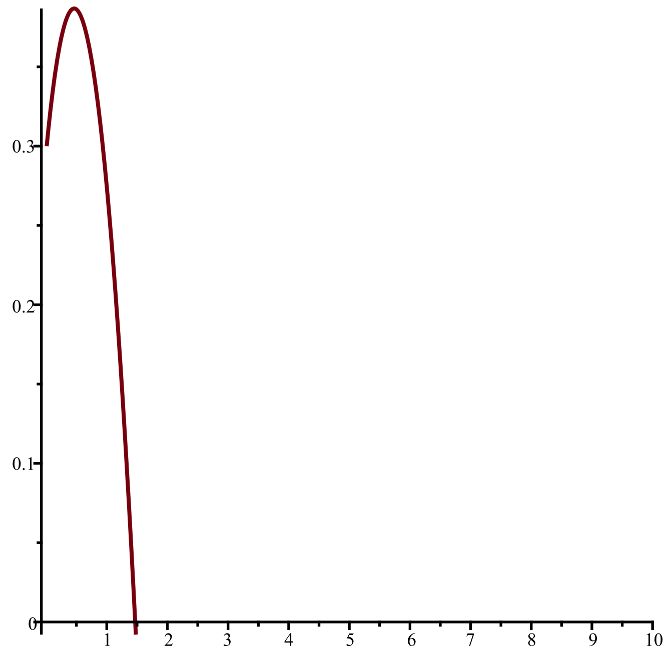
> *TimeSeries*(*g1*, [*m1*, *m2*, *m3*, *p1*, *p2*, *p3*], [0.5, 0.5, 0.2, 0.2, 0.3, 0.3], 0.01, 10, 1);
> *TimeSeries*(*g1*, [*m1*, *m2*, *m3*, *p1*, *p2*, *p3*], [0.5, 0.5, 0.2, 0.2, 0.3, 0.3], 0.01, 10, 2);
> *TimeSeries*(*g1*, [*m1*, *m2*, *m3*, *p1*, *p2*, *p3*], [0.5, 0.5, 0.2, 0.2, 0.3, 0.3], 0.01, 10, 3);
> *TimeSeries*(*g1*, [*m1*, *m2*, *m3*, *p1*, *p2*, *p3*], [0.5, 0.5, 0.2, 0.2, 0.3, 0.3], 0.01, 10, 4);
> *TimeSeries*(*g1*, [*m1*, *m2*, *m3*, *p1*, *p2*, *p3*], [0.5, 0.5, 0.2, 0.2, 0.3, 0.3], 0.01, 10, 5);
> *TimeSeries*(*g1*, [*m1*, *m2*, *m3*, *p1*, *p2*, *p3*], [0.5, 0.5, 0.2, 0.2, 0.3, 0.3], 0.01, 10, 6);

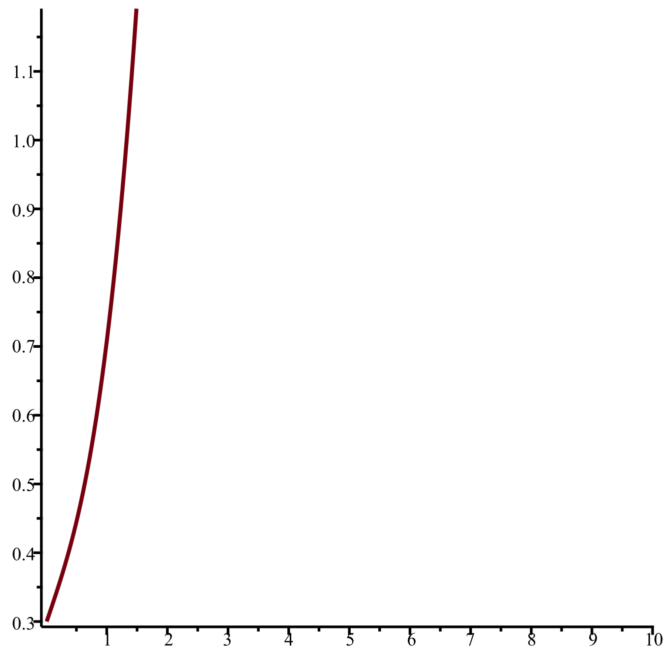

0.59
0.58
0.57
0.56
0.55
0.54
0.53
0.52
0.51
0.50
1
2
3
4
5
6
7
8
9
10


0.9
0.8
0.7
0.6
0.5
1
2
3
4
5
6
7
8
9
10


0.32
0.30
0.28
0.26
0.24
0.22
0.20
1
2
3
4
5
6
7
8
9
10


1.1
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
1
2
3
4
5
6
7
8
9
10

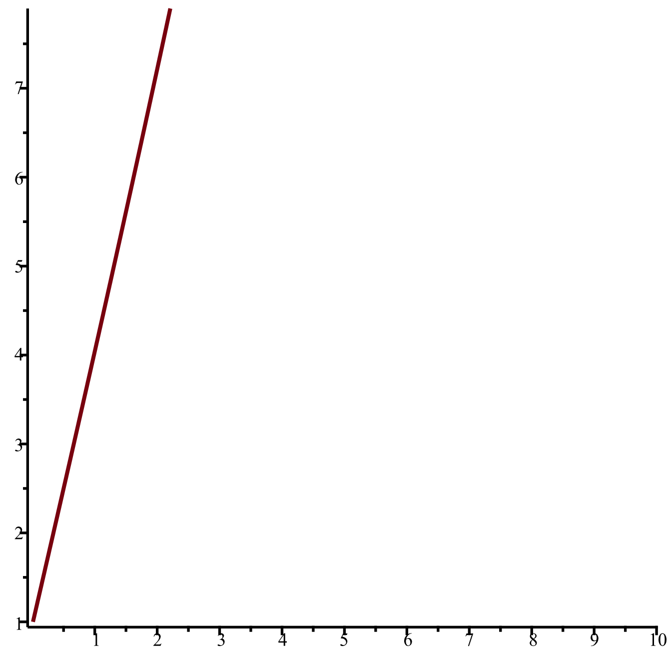




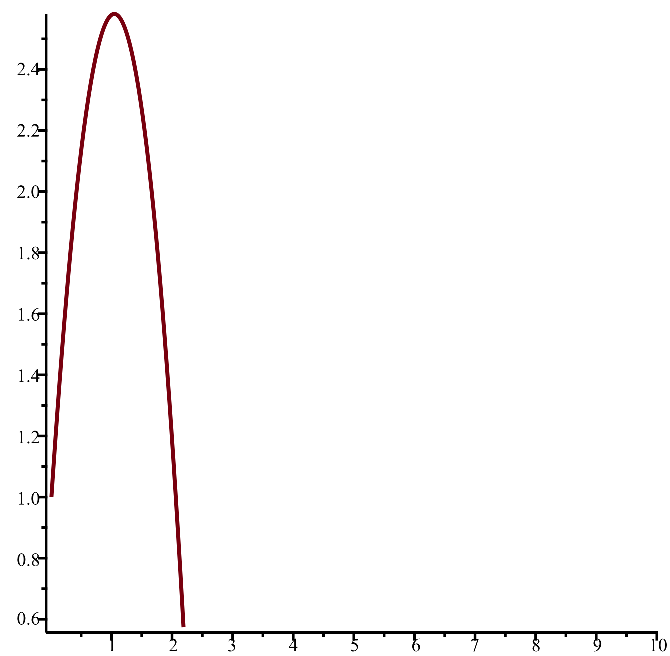
> $g1 := GeneNet(5, -2.1, 0.22, 0.4, 0.9, m1, m2, m3, p1, p2, p3);$

$$g1 := \left[4.1 - \frac{2.1}{1 + p2^{0.4}}, -m1 - \frac{2.1}{1 + m3^{0.4}} + 5, -m2 - \frac{2.1}{1 + p1^{0.4}} + 5, -0.22\, m3 + 0.198, -0.22\, p1 + 0.22\, m1, -0.22\, p2 + 0.22\, m2\right] \quad (8)$$

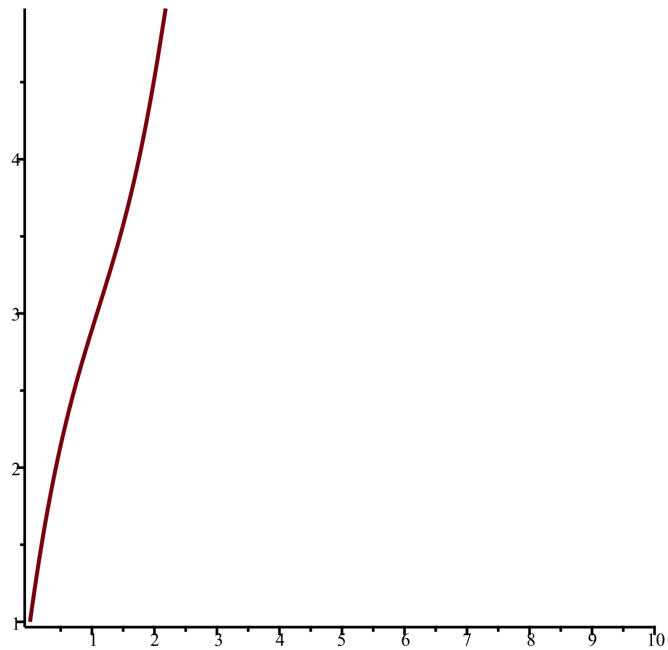
> TimeSeries(g1, [m1, m2, m3, p1, p2, p3], [1, 1, 1, 1, 1, 1], 0.01, 10, 1);
TimeSeries(g1, [m1, m2, m3, p1, p2, p3], [1, 1, 1, 1, 1, 1], 0.01, 10, 2);
TimeSeries(g1, [m1, m2, m3, p1, p2, p3], [1, 1, 1, 1, 1, 1], 0.01, 10, 3);
TimeSeries(g1, [m1, m2, m3, p1, p2, p3], [1, 1, 1, 1, 1, 1], 0.01, 10, 4);
TimeSeries(g1, [m1, m2, m3, p1, p2, p3], [1, 1, 1, 1, 1, 1], 0.01, 10, 5);
TimeSeries(g1, [m1, m2, m3, p1, p2, p3], [1, 1, 1, 1, 1, 1], 0.01, 10, 6);



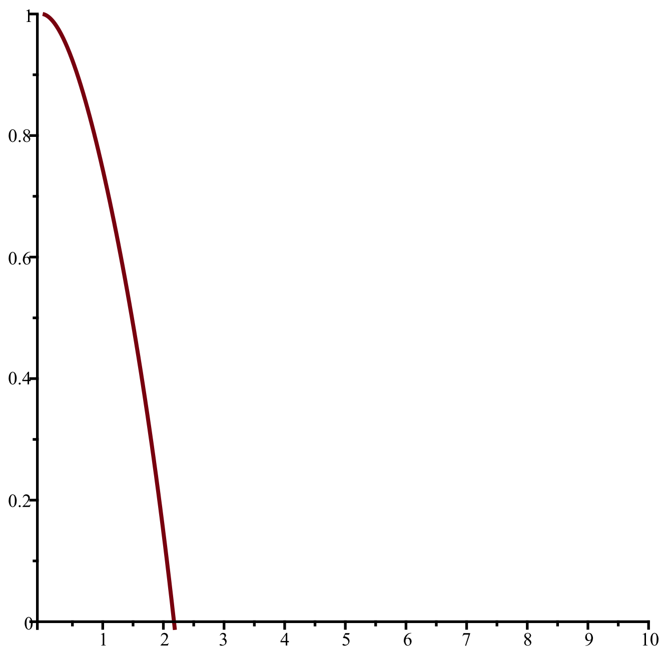
7
6
5
4
3
2
1
1
2
3
4
5
6
7
8
9
10



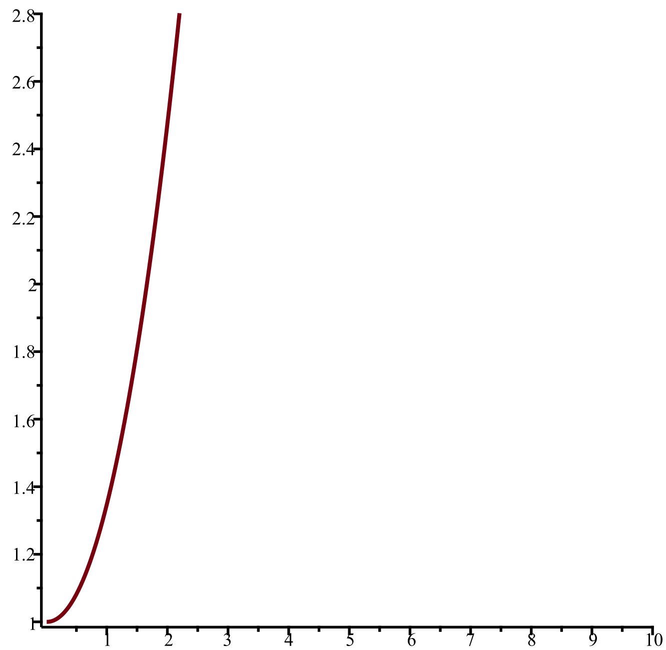
2.4
2.2
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
1
2
3
4
5
6
7
8
9
10

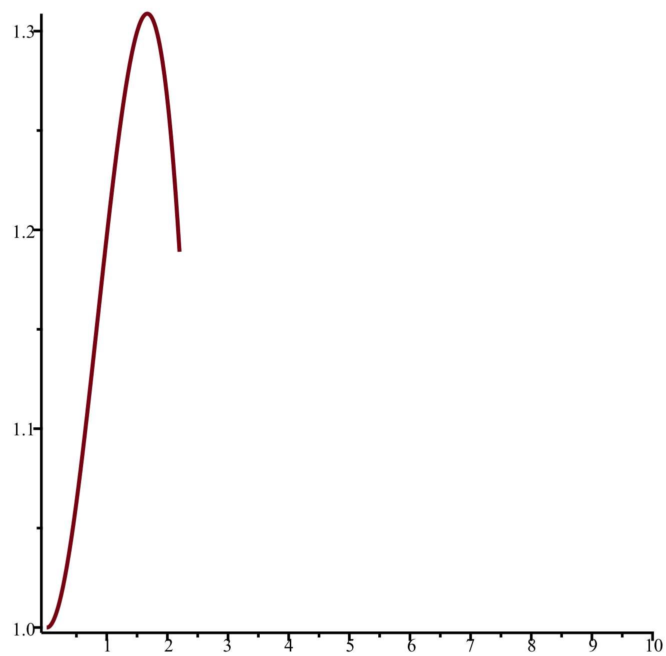


1
2
3
4
1
2
3
4
5
6
7
8
9
10



0
0.2
0.4
0.6
0.8
1
1
2
3
4
5
6
7
8
9
10

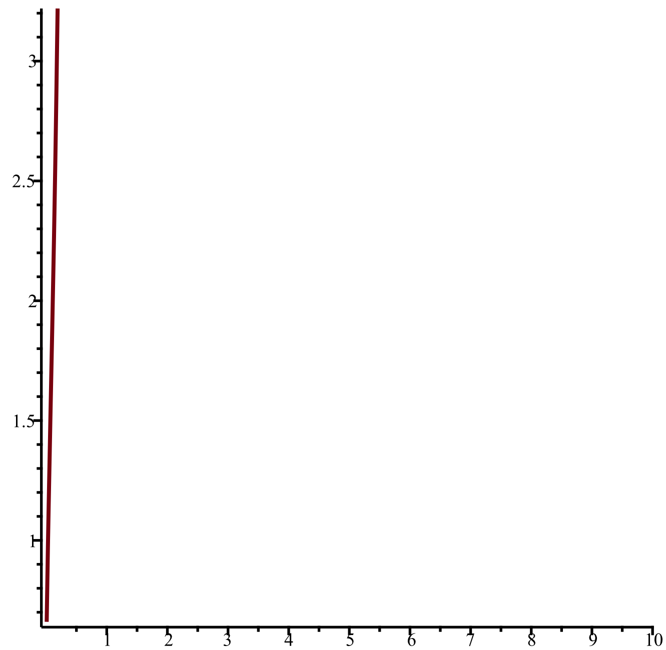


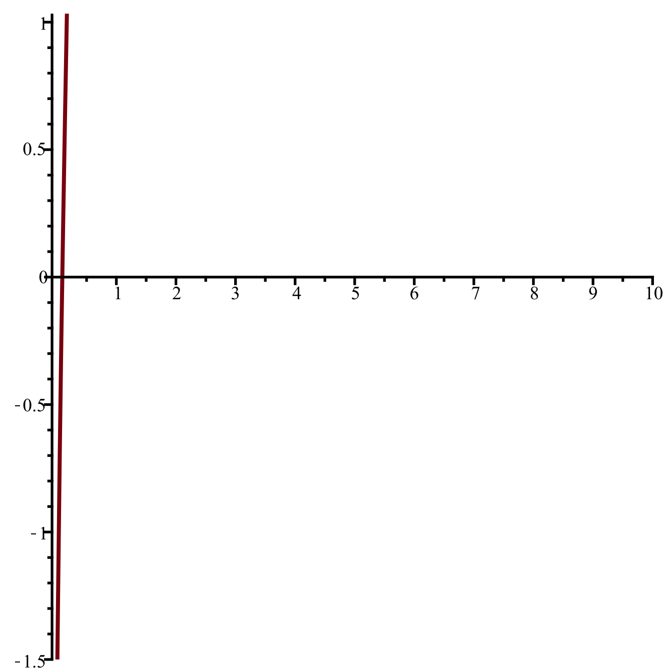


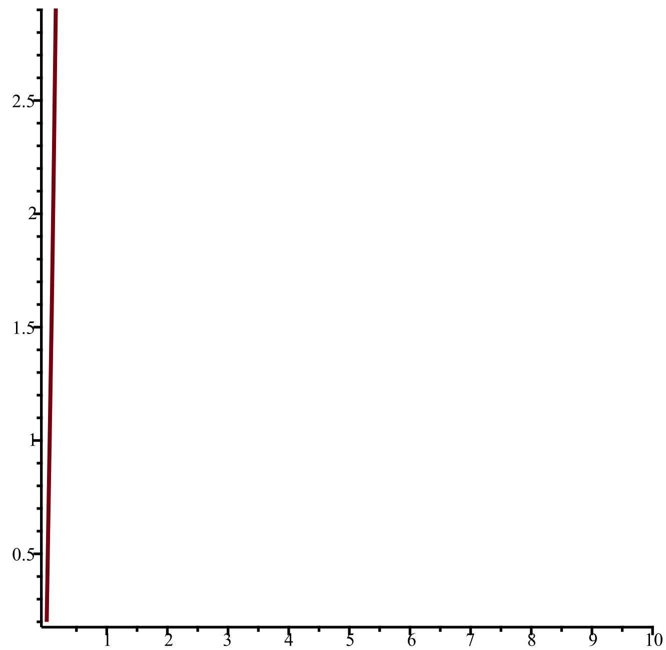
> $g1 := GeneNet(15, 6.6, -8, 3.25, 4.2, m1, m2, m3, p1, p2, p3);$

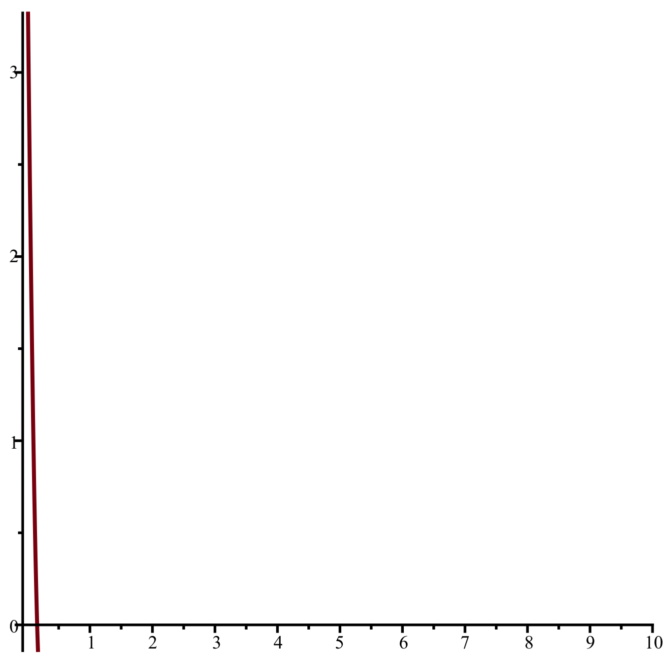
$$g1 := \left[10.8 + \frac{6.6}{1 + p2^{3.25}}, -m1 + \frac{6.6}{1 + m3^{3.25}} + 15, -m2 + \frac{6.6}{1 + p1^{3.25}} + 15, 8\,m3 - 33.6, 8\,p1 - 8\,m1, 8\,p2 - 8\,m2\right] \quad (9)$$

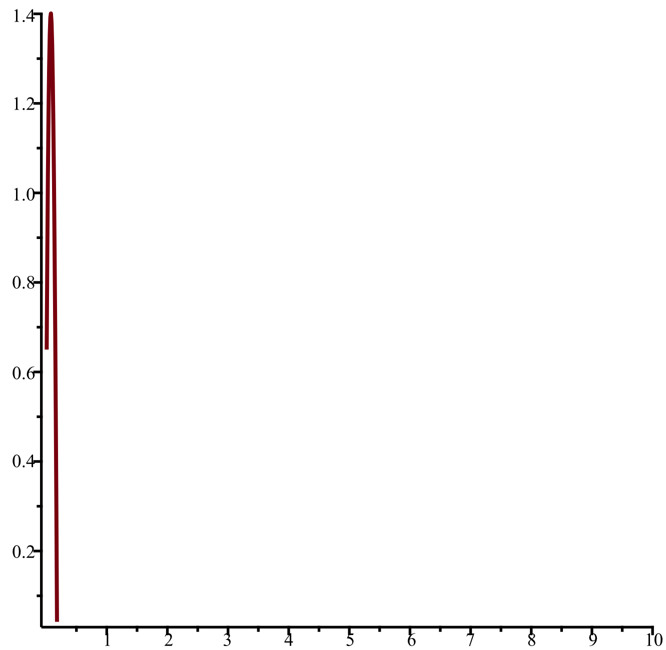
> $TimeSeries(g1, [m1, m2, m3, p1, p2, p3], [0.66, -1.5, 0.2, 3.33, 0.65, 9.8], 0.01, 10, 1);$
$TimeSeries(g1, [m1, m2, m3, p1, p2, p3], [0.66, -1.5, 0.2, 3.33, 0.65, 9.8], 0.01, 10, 2);$
$TimeSeries(g1, [m1, m2, m3, p1, p2, p3], [0.66, -1.5, 0.2, 3.33, 0.65, 9.8], 0.01, 10, 3);$
$TimeSeries(g1, [m1, m2, m3, p1, p2, p3], [0.66, -1.5, 0.2, 3.33, 0.65, 9.8], 0.01, 10, 4);$
$TimeSeries(g1, [m1, m2, m3, p1, p2, p3], [0.66, -1.5, 0.2, 3.33, 0.65, 9.8], 0.01, 10, 5);$
$TimeSeries(g1, [m1, m2, m3, p1, p2, p3], [0.66, -1.5, 0.2, 3.33, 0.65, 9.8], 0.01, 10, 6);$

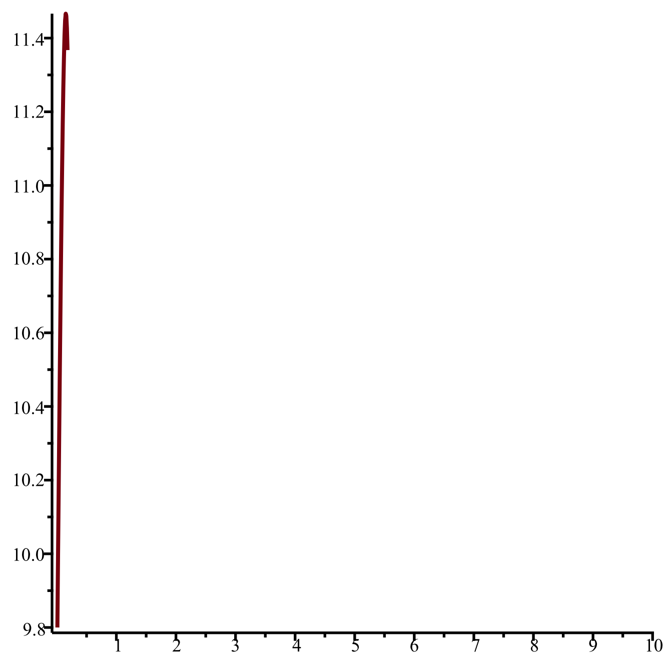

3
2.5
2
1.5
1
1 2 3 4 5 6 7 8 9 10


1
0.5
0
-0.5
-1
-1.5
1 2 3 4 5 6 7 8 9 10


2.5
2
1.5
1
0.5
1
2
3
4
5
6
7
8
9
10


3
2
1
0
1
2
3
4
5
6
7
8
9
10





> #3: Lotka
> Help(Lotka);

Lotka(r1,k1,r2,k2,b12,b21,N1,N2): The Lotka-Volterra continuous-time dynamical system, Eqs. (9a),(9b) (p. 224, section 6.3) of Edelstein-Keshet

with popoluations N1, N2, and parameters r1,r2,k1,k2, b12, b21 (called there beta_12 and beta_21)

Try:

Lotka(r1,k1,r2,k2,b12,b21,N1,N2);

Lotka(1,2,2,3,1,2,N1,N2); **(10)**

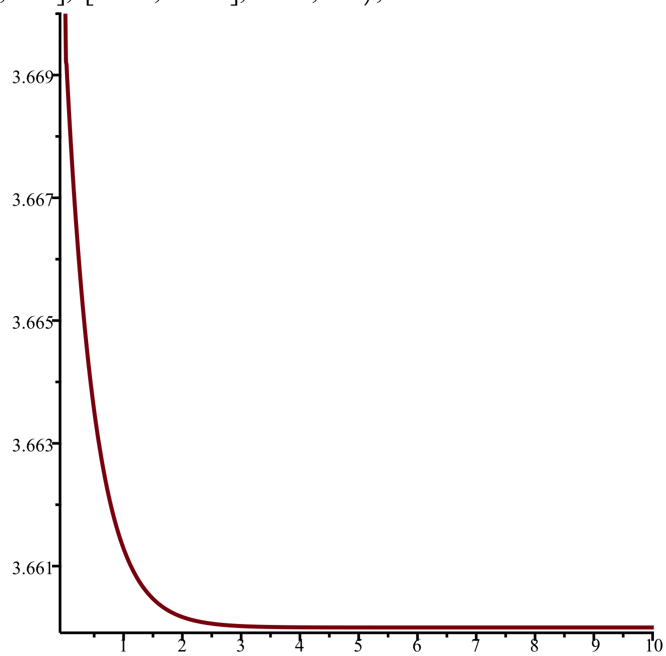
> $h := Lotka(2, 3.66, 5, 1, 2.9, 7, N1, N2);$

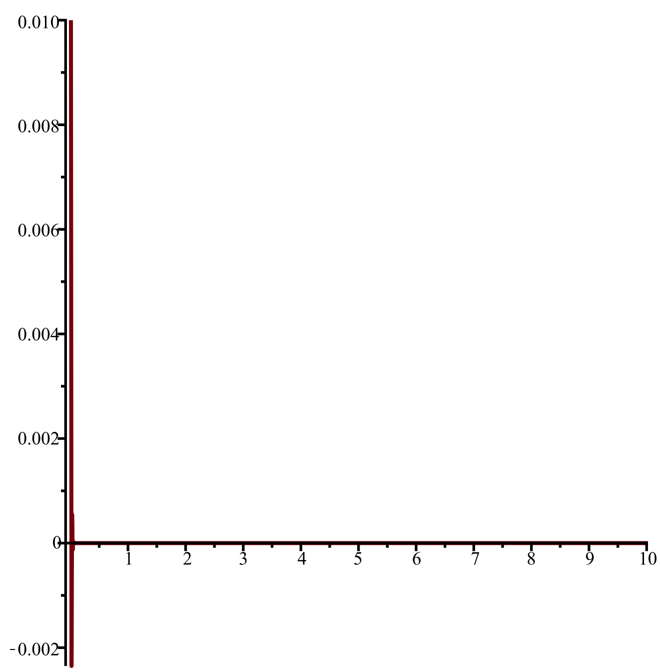
$SEquP(h, [N1, N2]);$

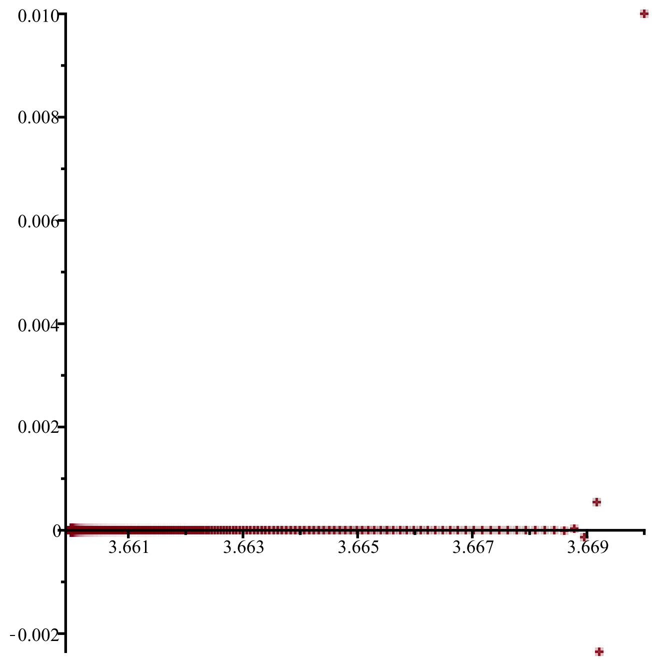
$$h := [0.5464480874\, N1\, (3.66 - N1 - 2.9\, N2),\, 5\, N2\, (1 - N2 - 7\, N1)]$$

$$\{[-0.03937823834, 1.275647668], [3.660000000, 0.]\} \quad \textbf{(11)}$$

> $TimeSeries(h, [N1, N2], [3.67, 0.01], 0.01, 10, 1);$
> $TimeSeries(h, [N1, N2], [3.67, 0.01], 0.01, 10, 2);$
> $PhaseDiag(h, [N1, N2], [3.67, 0.01], 0.01, 10);$





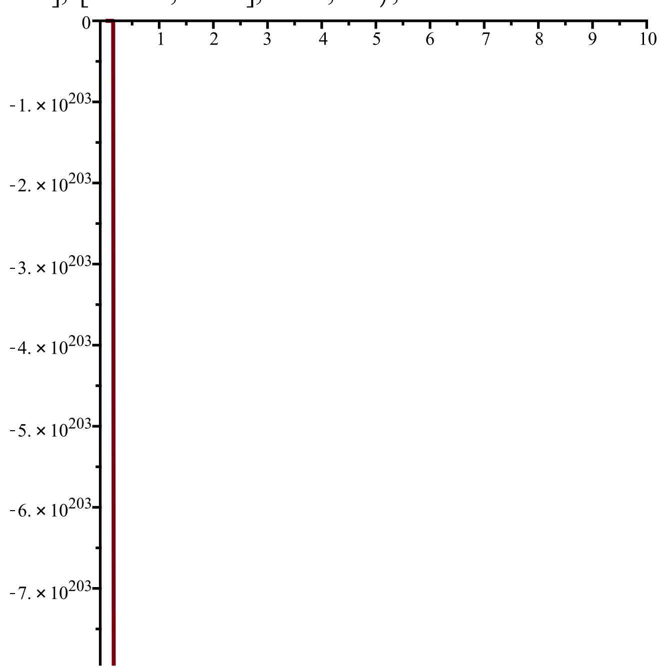


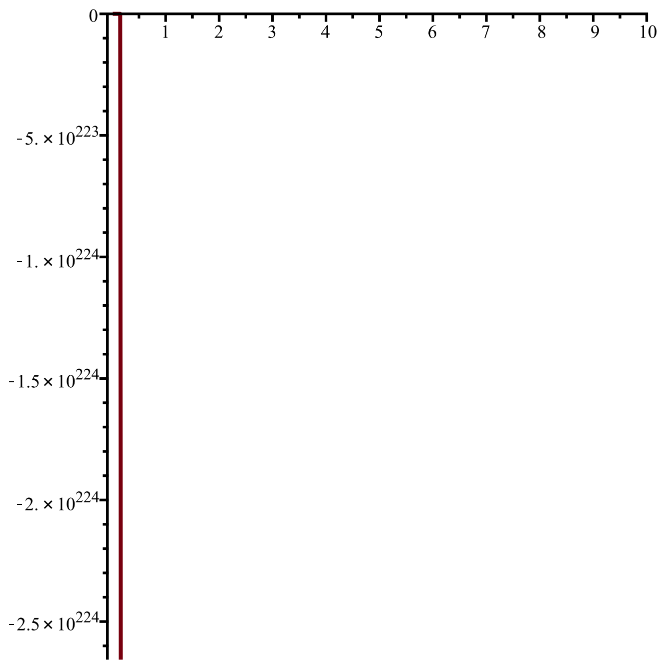
> $h := Lotka(0.5, 2.14, 45, 0.3, 0.98, 5.8, N1, N2);$
> $SEquP(h, [N1, N2]);$

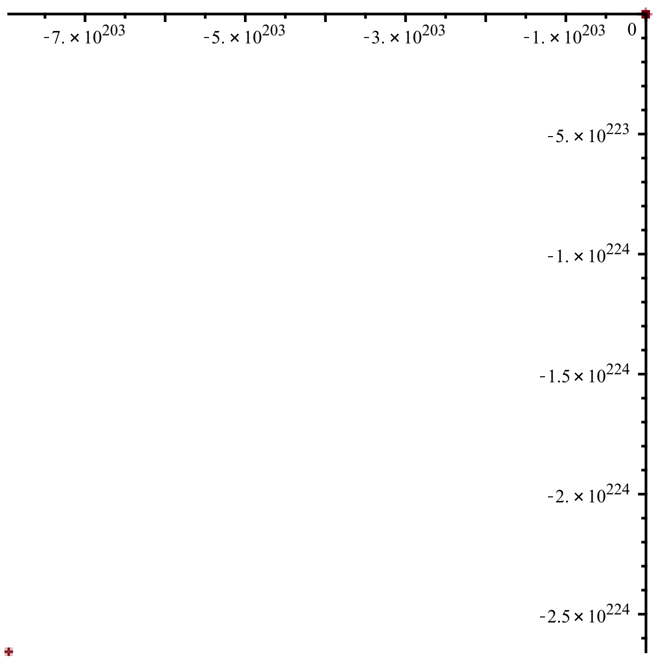
$$h := [0.2336448598\, N1\, (2.14 - N1 - 0.98\, N2), 150.0000000\, N2\, (0.3 - N2 - 5.8\, N1)]$$

$$\{[-0.3941076003, 2.585824082], [2.140000000, 0.]\} \quad \textbf{(12)}$$

> $TimeSeries(h, [N1, N2], [-0.38, 2.60], 0.01, 10, 1);$
> $TimeSeries(h, [N1, N2], [-0.38, 2.60], 0.01, 10, 2);$
> $PhaseDiag(h, [N1, N2], [-0.38, 2.60], 0.01, 10);$





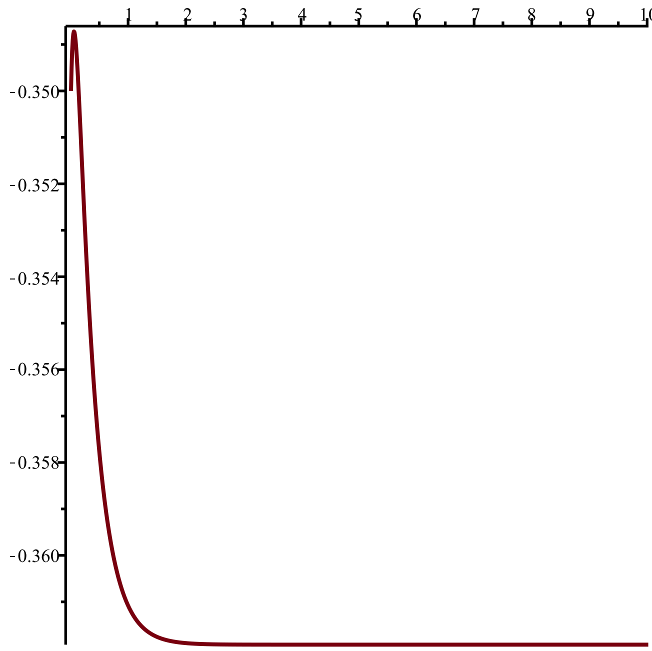


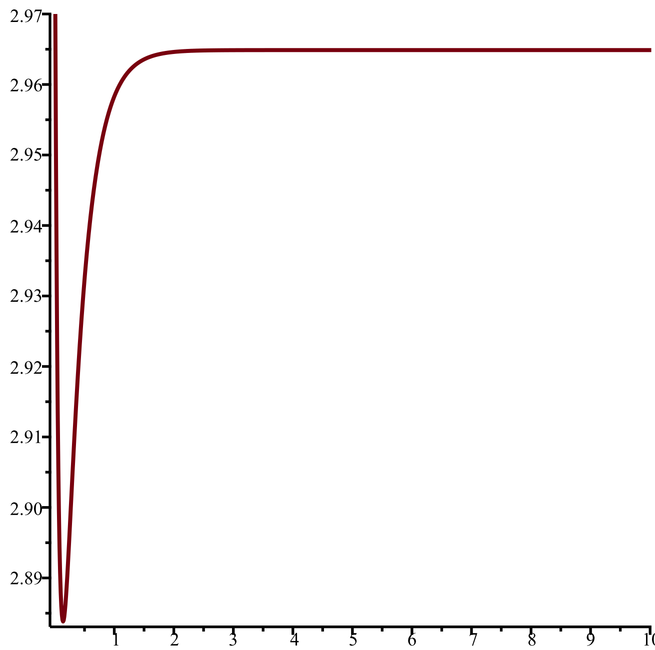

```
> h := Lotka(3.5, 0.32, 4, 0.54, 0.23, 6.7, N1, N2);
  SEquP(h, [N1, N2]);
```

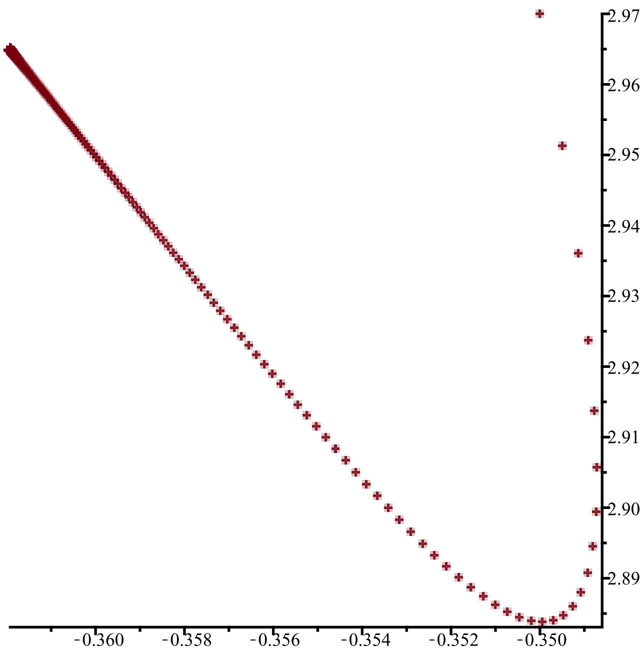
$$h := [10.93750000\, N1\, (0.32 - N1 - 0.23\, N2),\ 7.407407407\, N2\, (0.54 - N2 - 6.7\, N1)]$$

$$\{[-0.3619223660, 2.964879852], [0.3200000000, 0.]\} \quad \textbf{(13)}$$

```
> TimeSeries(h, [N1, N2], [-0.35, 2.97], 0.01, 10, 1);
  TimeSeries(h, [N1, N2], [-0.35, 2.97], 0.01, 10, 2);
  PhaseDiag(h, [N1, N2], [-0.35, 2.97], 0.01, 10);
```


1
2
3
4
5
6
7
8
9
10
-0.350
-0.352
-0.354
-0.356
-0.358
-0.360


2.97
2.96
2.95
2.94
2.93
2.92
2.91
2.90
2.89
1
2
3
4
5
6
7
8
9
10



> *#4: Volterra*
Help(*Volterra*);

Volterra(a,b,c,d,x,y): The (simple, original) Volterra predator-prey continuous-time dynamical system with parameters a,b,c,d

Given by Eqs. (7a) (7b) in Edelstein-Keshet p. 219 (section 6.2).

a,b,c,d may be symbolic or numeric

Try:

Volterra(a,b,c,d,x,y);

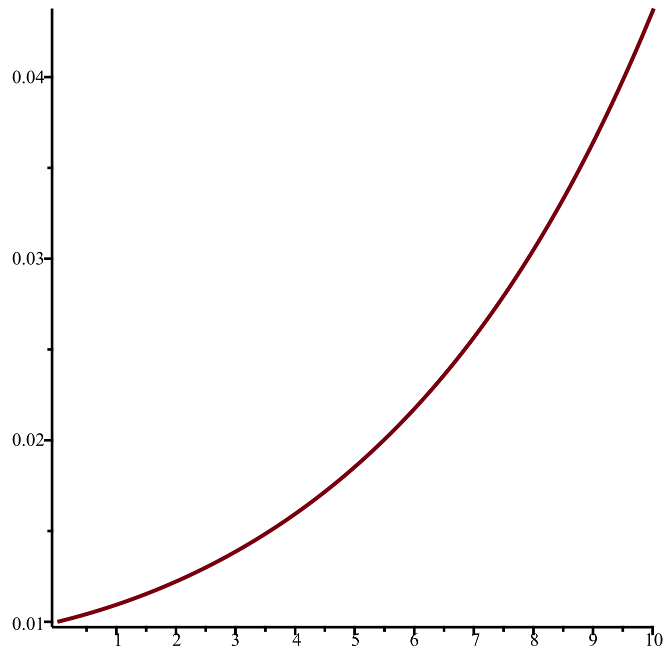
Volterra(1,2,3,4,x,y); **(14)**

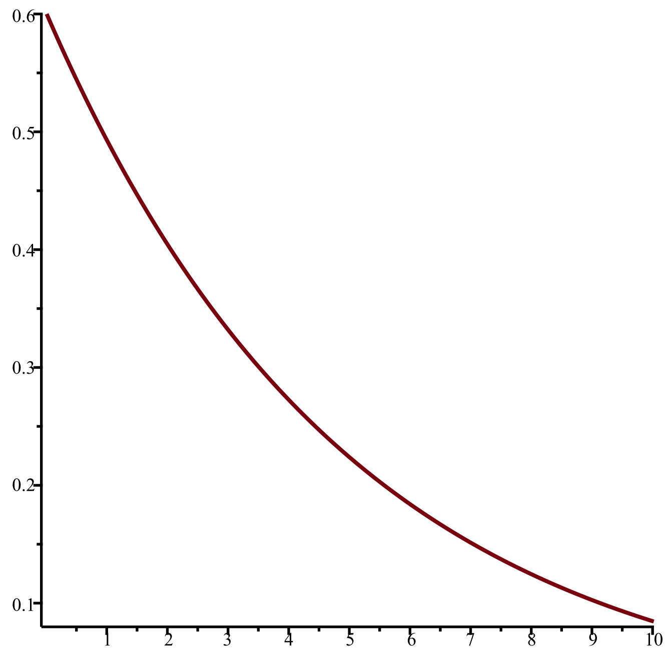
> $q := Volterra(0.2, 3, 1.5, 0.56, x, y);$
$SEquP(q, [x, y]);$

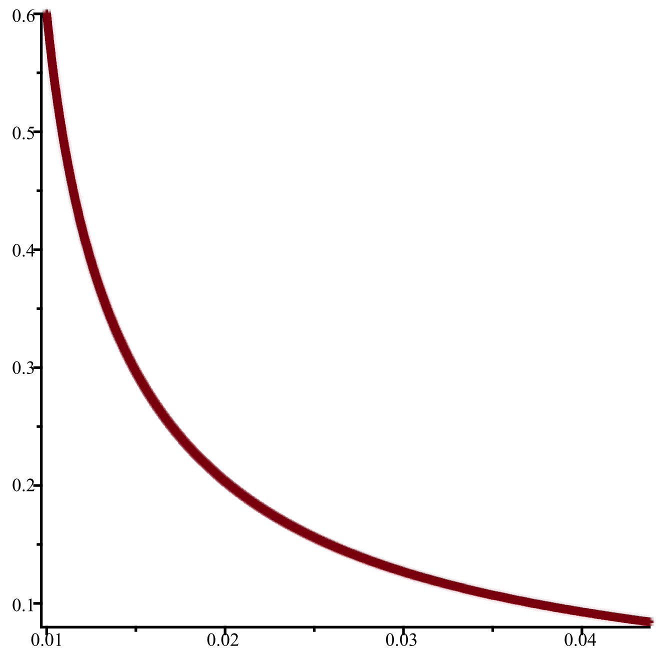
$$q := [0.2\,x - 3\,x\,y, -1.5\,y + 0.56\,x\,y]$$

$$\varnothing$$ **(15)**

> $TimeSeries(q, [x, y], [0.01, 0.6], 0.01, 10, 1);$
$TimeSeries(q, [x, y], [0.01, 0.6], 0.01, 10, 2);$
$PhaseDiag(q, [x, y], [0.01, 0.6], 0.01, 10);$


0.04
0.03
0.02
0.01
1
2
3
4
5
6
7
8
9
10


0.6
0.5
0.4
0.3
0.2
0.1
1
2
3
4
5
6
7
8
9
10

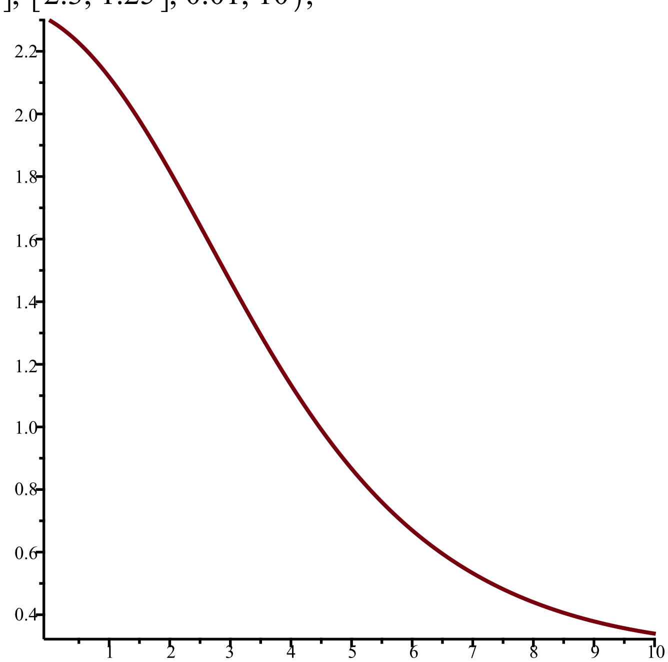


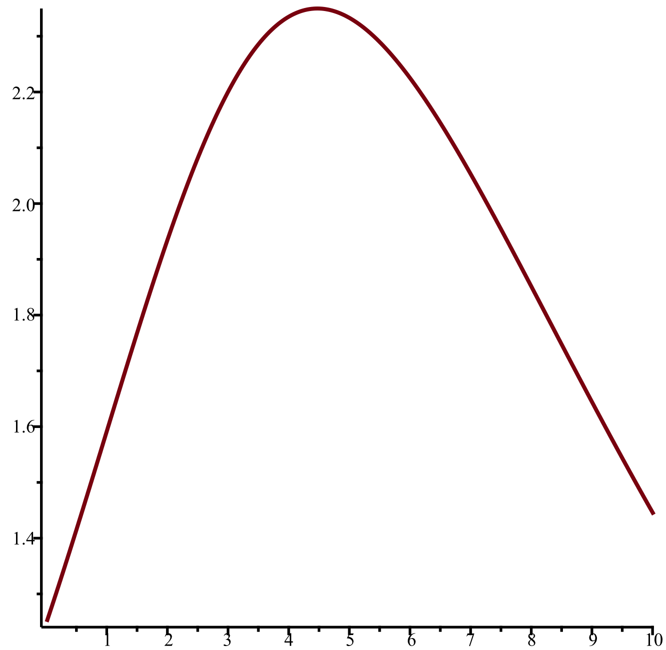
> $q := Volterra(0.2, 0.2, 0.2, 0.2, x, y);$
> $SEquP(q, [x, y]);$

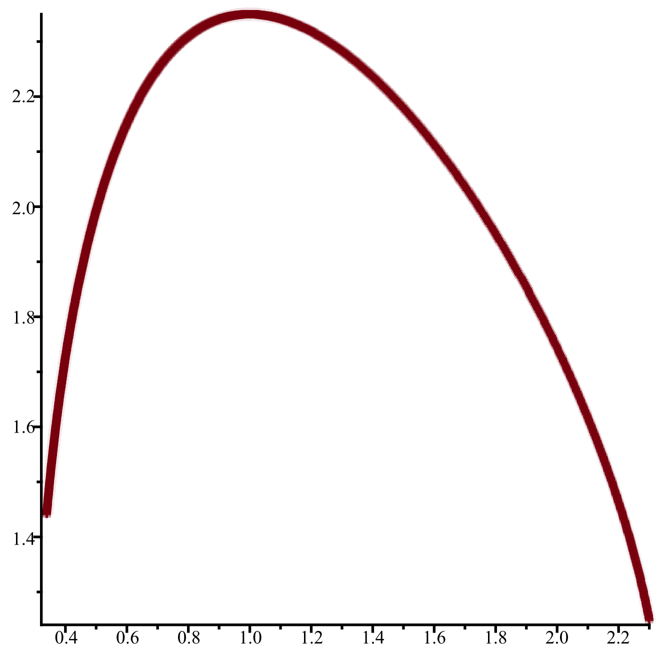
$$q := [0.2\,x - 0.2\,x\,y, -0.2\,y + 0.2\,x\,y]$$

$$\varnothing \quad (16)$$

> $TimeSeries(q, [x, y], [2.3, 1.25], 0.01, 10, 1);$
> $TimeSeries(q, [x, y], [2.3, 1.25], 0.01, 10, 2);$
> $PhaseDiag(q, [x, y], [2.3, 1.25], 0.01, 10);$





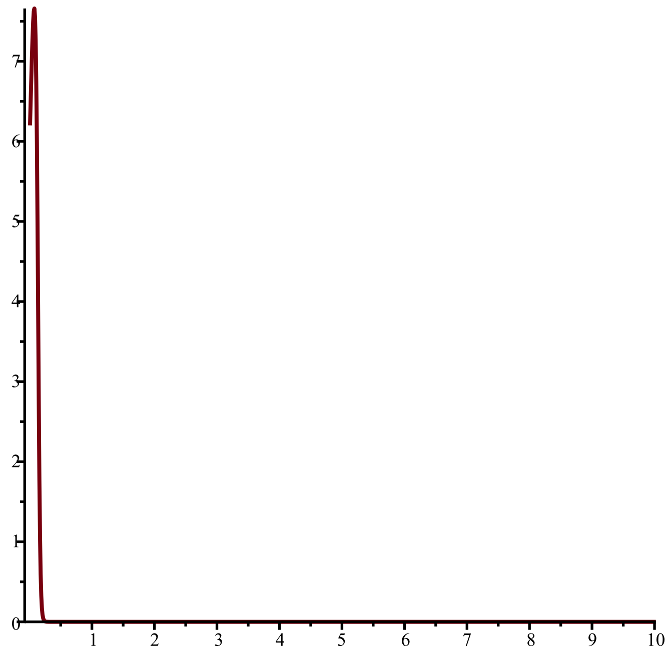



```
> q := Volterra(5.7, 4.1, 0.35, 4.6, x , y);
  SEquP(q, [x, y]);
```

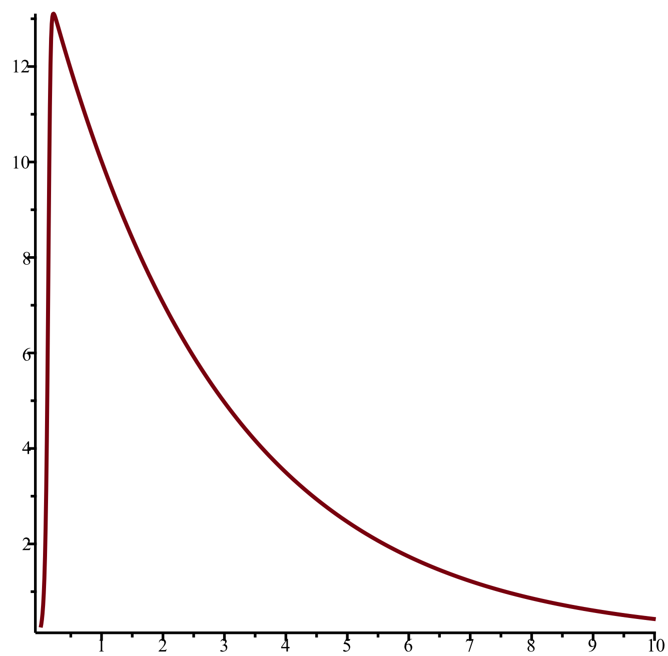
$$q := [5.7\,x - 4.1\,x\,y,\ -0.35\,y + 4.6\,x\,y]$$

$$\varnothing \qquad (17)$$

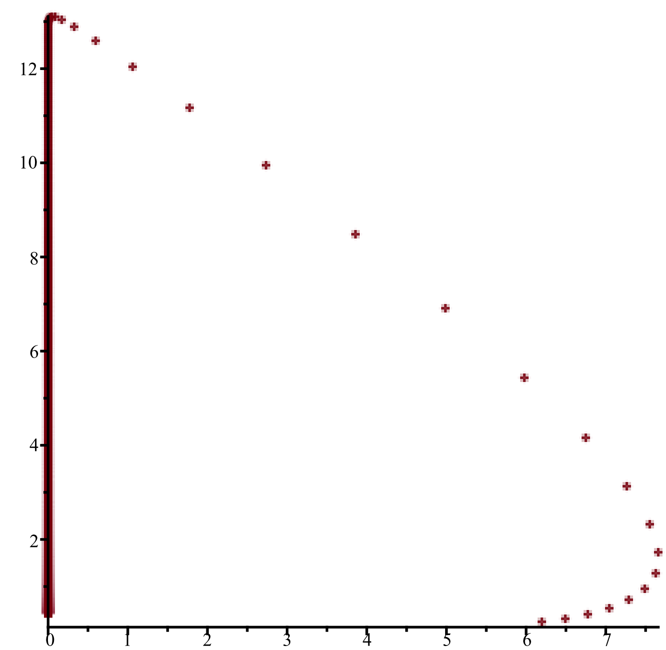
```
> TimeSeries(q, [x, y], [6.2, 0.25], 0.01, 10, 1);
  TimeSeries(q, [x, y], [6.2, 0.25], 0.01, 10, 2);
  PhaseDiag(q, [x, y], [6.2, 0.25], 0.01, 10);
```



0
1
2
3
4
5
6
7
8
9
10



2
4
6
8
10
12
1
2
3
4
5
6
7
8
9
10



> #5: VolterraM
Help(VolterraM);

VolterraM(a,b,c,d,x,K,y): The MODIFIED Volterra predator-prey continuous-time dynamical system with parameters a,b,c,d,K

Given by Eqs. (8a) (8b) in Edelstein-Keshet p. 220 (section 6.2).

a,b,c,d ,Kmay be symbolic or numeric

Try:

VolterraM(a,b,c,d,K,x,y);

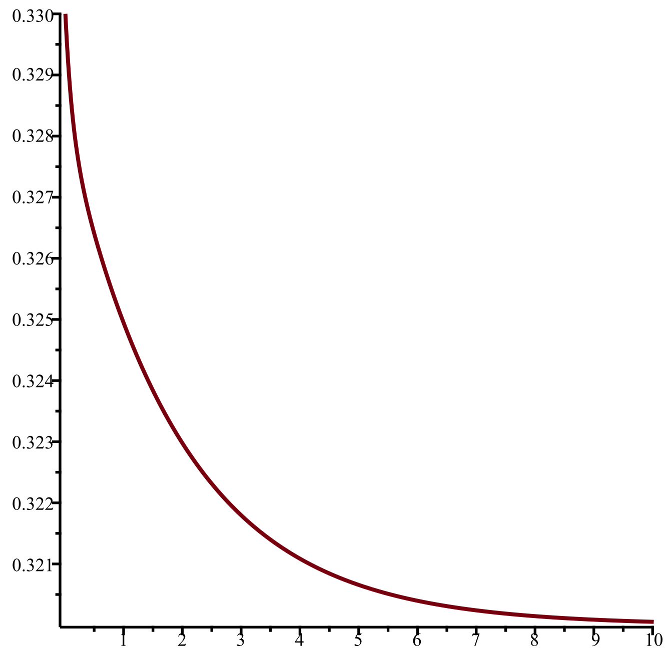
VolterraM(1,2,3,4,3,x,y); **(18)**

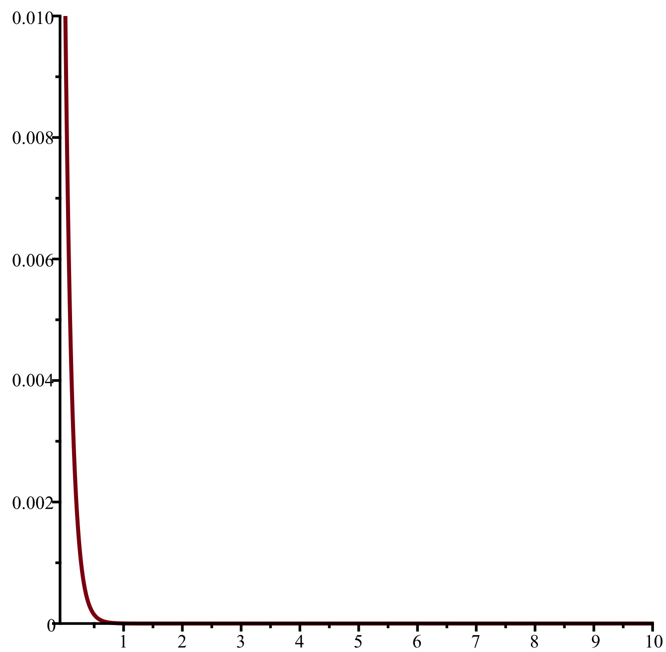
> w := VolterraM(0.5, 4.23, 8.6, 0.32, 1, x, y);
SEquP(w, [x, y]);

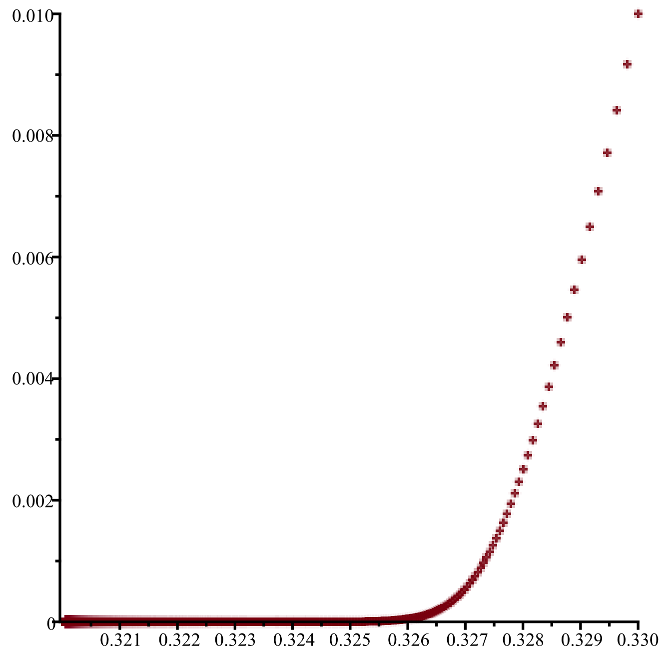
$$w := [0.5\,x\,(1 - 3.125000000\,x) - 4.23\,x\,y,\ -8.6\,y + x\,y]$$

$$\{[0.3200000000, 0.]\} \quad \textbf{(19)}$$

> TimeSeries(w, [x, y], [0.33, 0.01], 0.01, 10, 1);
TimeSeries(w, [x, y], [0.33, 0.01], 0.01, 10, 2);
PhaseDiag(w, [x, y], [0.33, 0.01], 0.01, 10);


0.330
0.329
0.328
0.327
0.326
0.325
0.324
0.323
0.322
0.321
1 2 3 4 5 6 7 8 9 10


0.010
0.008
0.006
0.004
0.002
0
1 2 3 4 5 6 7 8 9 10

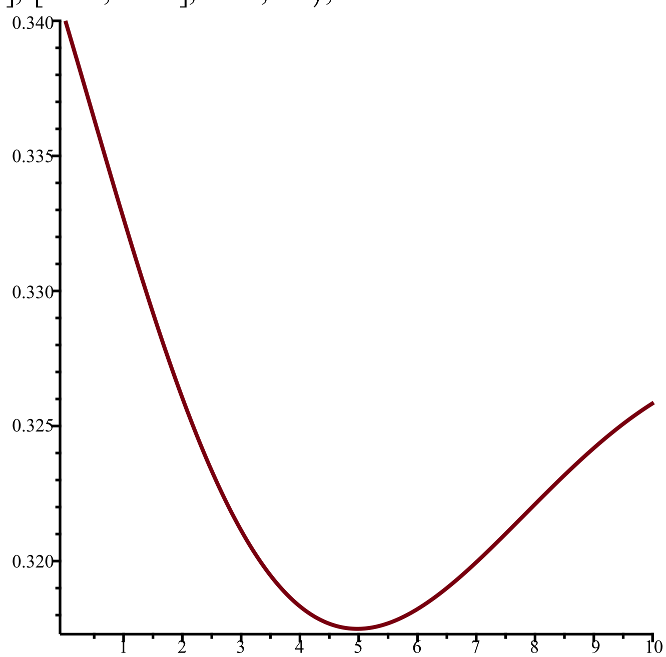


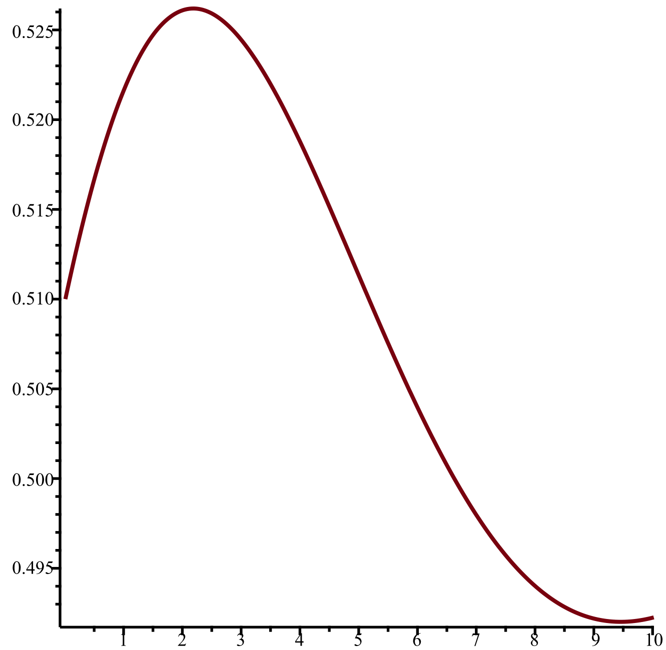
> *w* := *VolterraM*(0.65, 0.65, 0.65, 0.65, 2, *x*, *y*);
SEquP(*w*, [*x*, *y*]);

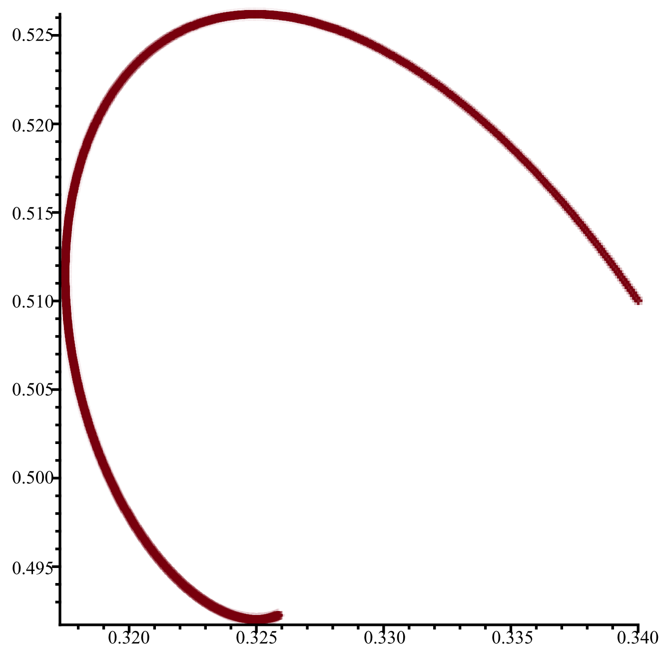
$$w := [0.65\,x\,(1 - 1.538461538\,x) - 0.65\,x\,y,\ -0.65\,y + 2\,x\,y]$$

$$\{[0.3250000000, 0.5000000002]\} \quad \textbf{(20)}$$

> *TimeSeries*(*w*, [*x*, *y*], [0.34, 0.51], 0.01, 10, 1);
TimeSeries(*w*, [*x*, *y*], [0.34, 0.51], 0.01, 10, 2);
PhaseDiag(*w*, [*x*, *y*], [0.34, 0.51], 0.01, 10);





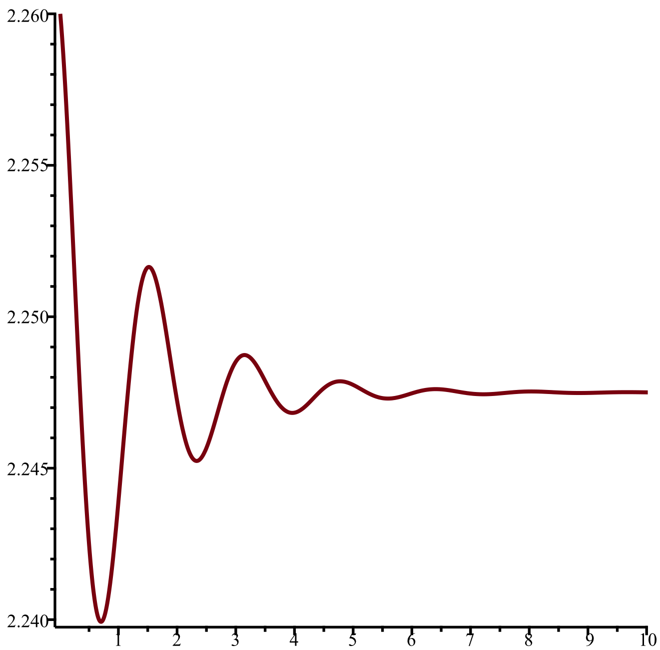


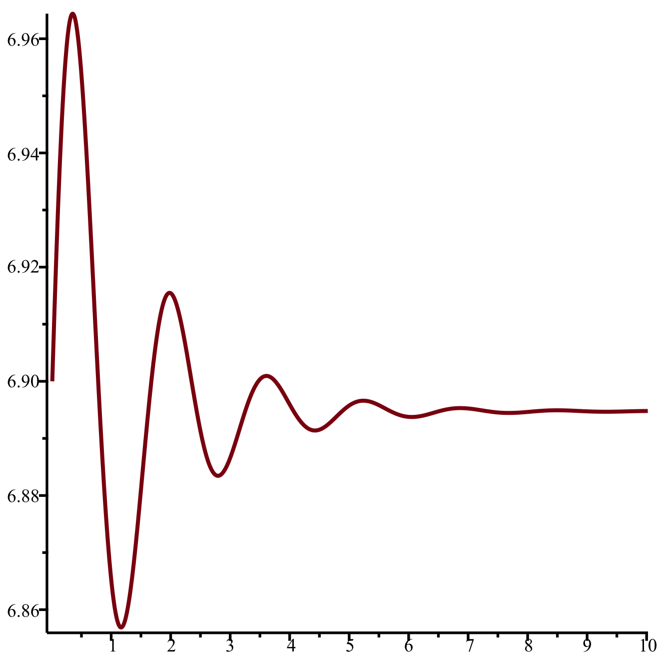
> $w := VolterraM(3.33, 0.247, 8.99, 4.6, 4, x, y);$
$SEquP(w, [x, y]);$

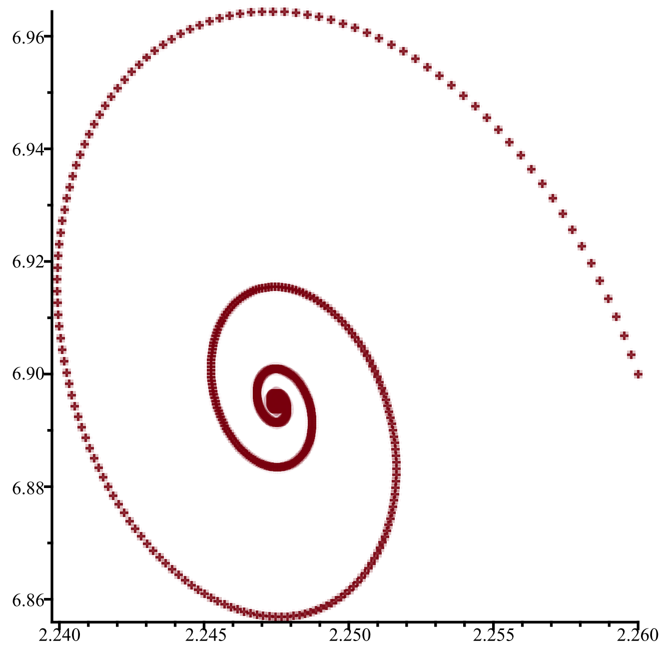
$$w := [3.33\, x\, (1 - 0.2173913043\, x) - 0.247\, x\, y,\ -8.99\, y + 4\, x\, y]$$

$$\{[2.247500000, 6.894758847]\} \quad \textbf{(21)}$$

> $TimeSeries(w, [x, y], [2.26, 6.90], 0.01, 10, 1);$
$TimeSeries(w, [x, y], [2.26, 6.90], 0.01, 10, 2);$
$PhaseDiag(w, [x, y], [2.26, 6.90], 0.01, 10);$


2.260
2.255
2.250
2.245
2.240
1 2 3 4 5 6 7 8 9 10


6.96
6.94
6.92
6.90
6.88
6.86
1 2 3 4 5 6 7 8 9 10



6.96
6.94
6.92
6.90
6.88
6.86
2.240
2.245
2.250
2.255
2.260

>