```
> #Nikita John, Assignment 20
  #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, RandNice, ToSys` ) :

else
ezra(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( `HW2, HW2g, RT` ) :

else
ezra(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( ` RandNice, SIRS , SIRSdemo` ) :

else
ezra(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( ` Dis, EquP, FP, Orb, Orbk, PhaseDiag, SEquP, SFP, TimeSeries`) :



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 ] = 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 ] = HW2 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 ] = HW2g 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] = 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] = 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] = 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[1](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] = 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]);`):

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:



    #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:


    #HW2g(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]])), [u, v] :
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:
```

*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 (i)
```

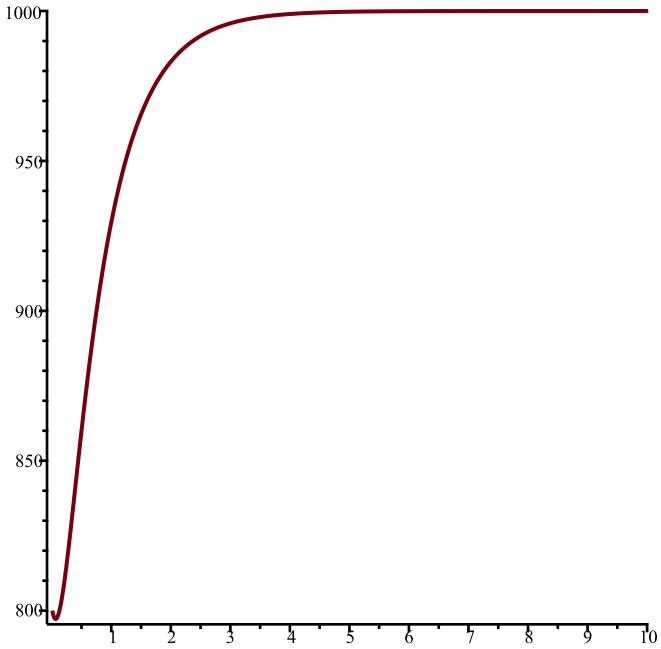
$$F1 := SIRS\left(s, i, 0.3 \cdot \left(\frac{2}{1000}\right), 5, 2, 1000\right);$$

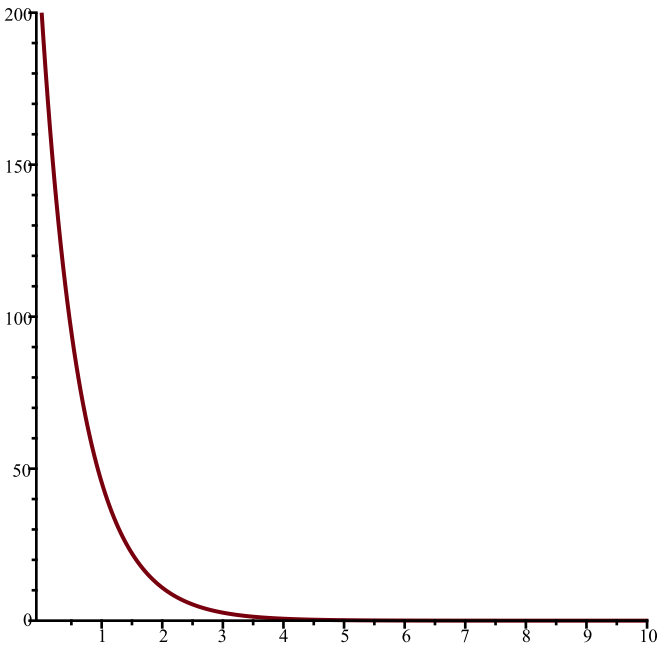
```
EquP(F1, [s, i]);
SEquP(F1, [s, i]);
TimeSeries(F1, [s, i], [800, 200], 0.01, 10, 1);
TimeSeries(F1, [s, i], [800, 200], 0.01, 10, 2);
PhaseDiag(F1, [s, i], [800, 200], 0.01, 10);
```

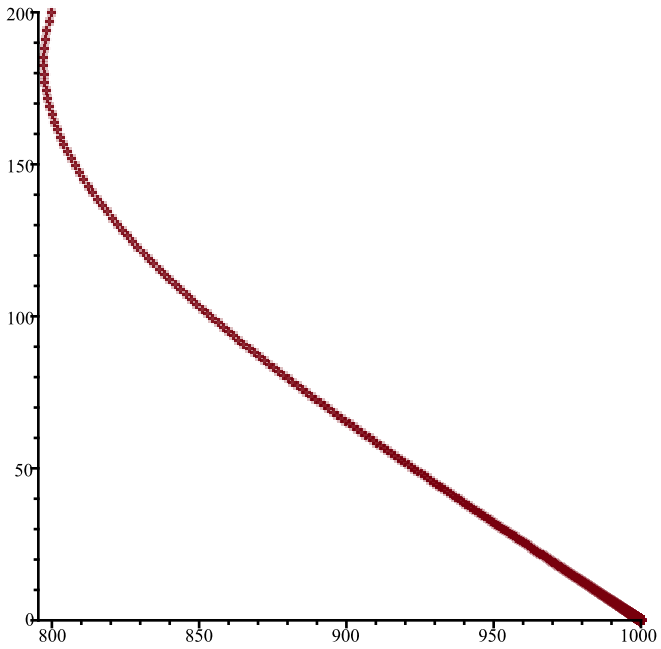
$$F1 := [-0.0006000000000\, s\, i + 5000 - 5\, s - 5\, i,\ 0.0006000000000\, s\, i - 2\, i]$$

$$\{[1000., 0.], [3333.333333, -1666.666667]\}$$

$$\{[1000., 0.]\}$$





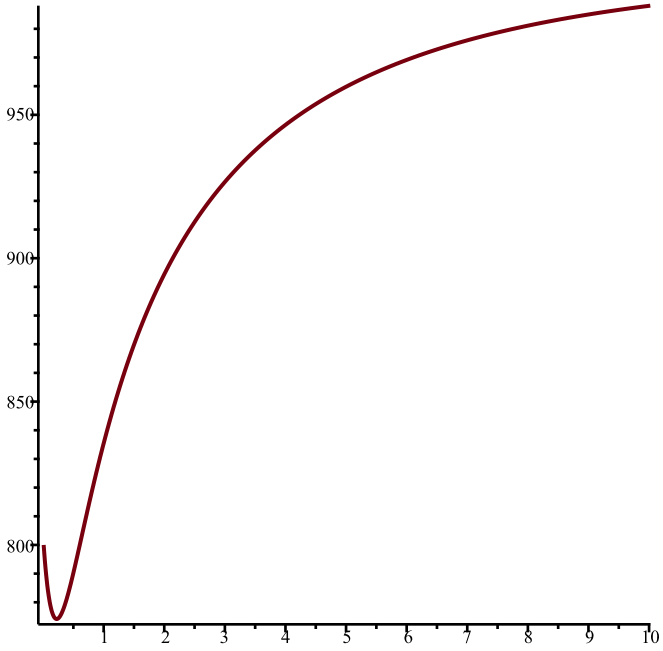

> $F2 := SIRS\left(s, i, 0.9 \cdot \left(\frac{2}{1000}\right), 5, 2, 1000\right);$

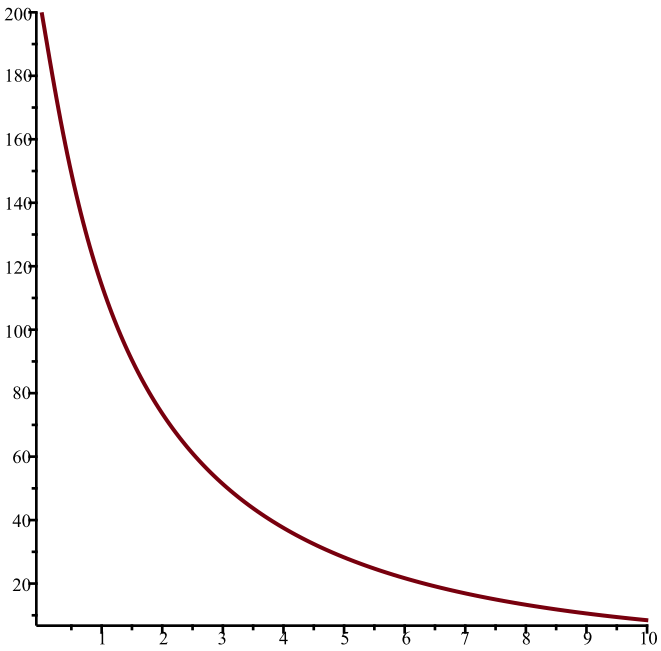
$EquP(F2, [s, i]);$
$SEquP(F2, [s, i]);$
$TimeSeries(F2, [s, i], [800, 200], 0.01, 10, 1);$
$TimeSeries(F2, [s, i], [800, 200], 0.01, 10, 2);$
$PhaseDiag(F2, [s, i], [800, 200], 0.01, 10);$

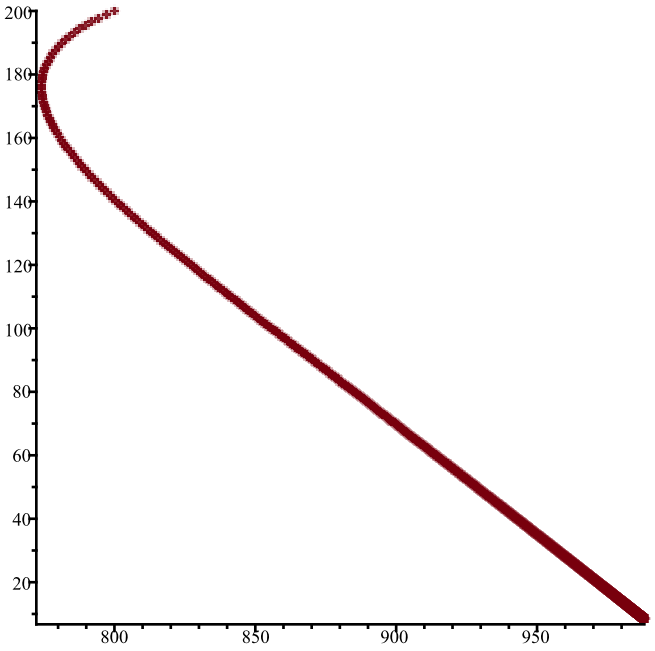
$$F2 := [-0.001800000000\, s\, i + 5000 - 5\, s - 5\, i,\ 0.001800000000\, s\, i - 2\, i]$$

$$\{[1000., 0.], [1111.111111, -79.36507937]\}$$

$$\{[1000., 0.]\}$$





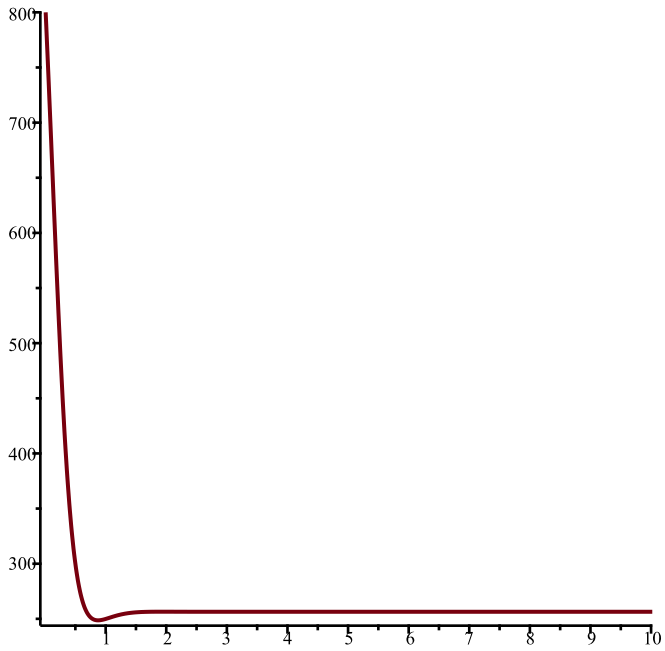

> $F3 := SIRS\left(s, i, 3.9 \cdot \left(\frac{2}{1000}\right), 5, 2, 1000\right);$

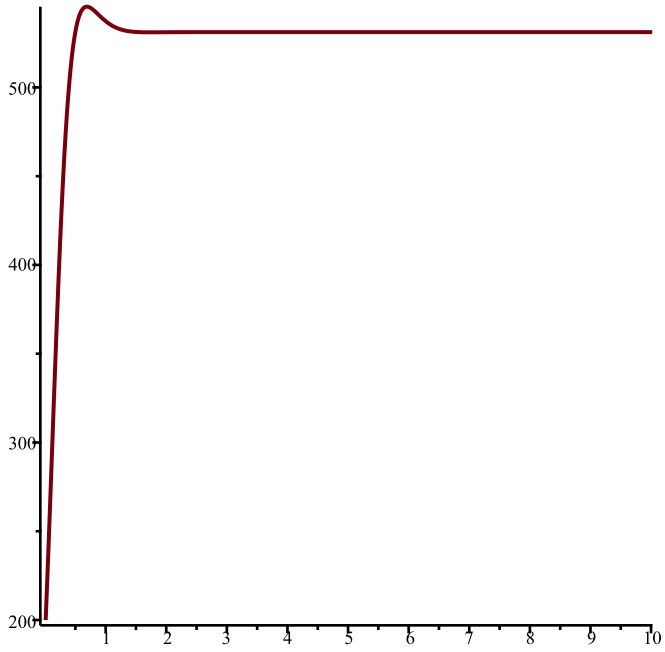
$EquP(F3, [s, i]);$
$SEquP(F3, [s, i]);$
$TimeSeries(F3, [s, i], [800, 200], 0.01, 10, 1);$
$TimeSeries(F3, [s, i], [800, 200], 0.01, 10, 2);$
$PhaseDiag(F3, [s, i], [800, 200], 0.01, 10);$

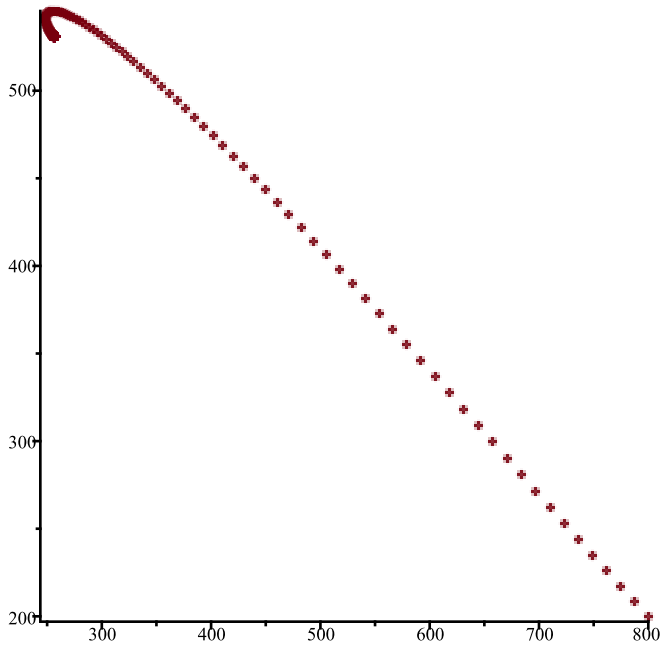
$$F3 := [-0.007800000000\, s\, i + 5000 - 5\, s - 5\, i,\ 0.007800000000\, s\, i - 2\, i]$$

$$\{[256.4102564, 531.1355311], [1000., 0.]\}$$

$$\{[256.4102564, 531.1355311]\}$$


800
700
600
500
400
300
1
2
3
4
5
6
7
8
9
10


500
400
300
200
1
2
3
4
5
6
7
8
9
10




> *#1 (ii)*

$$G1 := SIRS\left(s, i, 0.3 \cdot \left(\frac{3}{1000}\right), 6, 3, 1000\right);$$

$$EquP(G1, [s, i]);$$

$$SEquP(G1, [s, i]);$$

$$TimeSeries(G1, [s, i], [800, 200], 0.01, 10, 1);$$

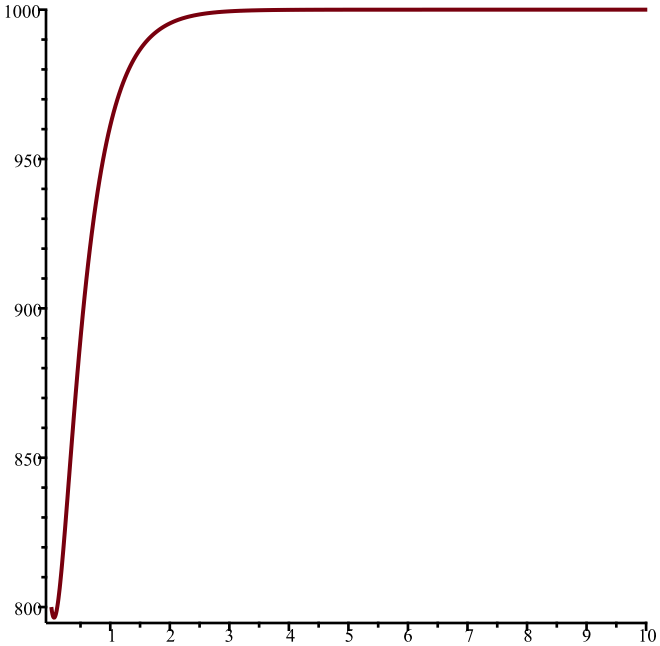
$$TimeSeries(G1, [s, i], [800, 200], 0.01, 10, 2);$$

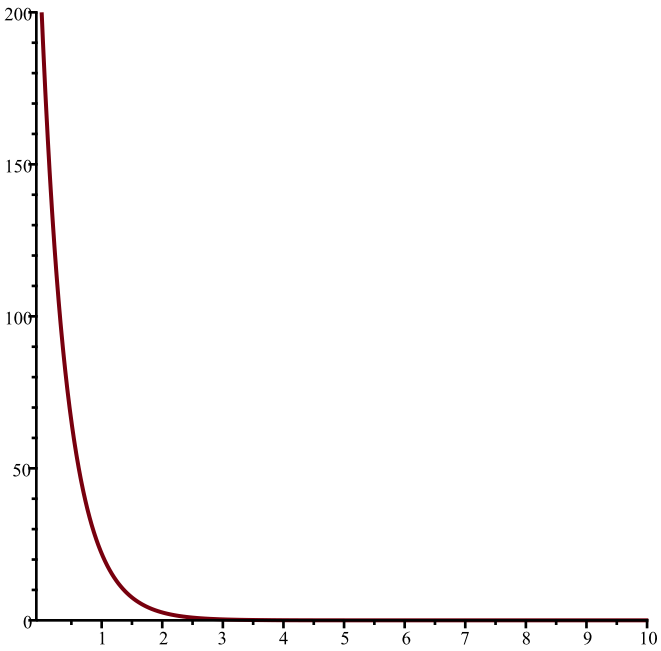
$$PhaseDiag(G1, [s, i], [800, 200], 0.01, 10);$$

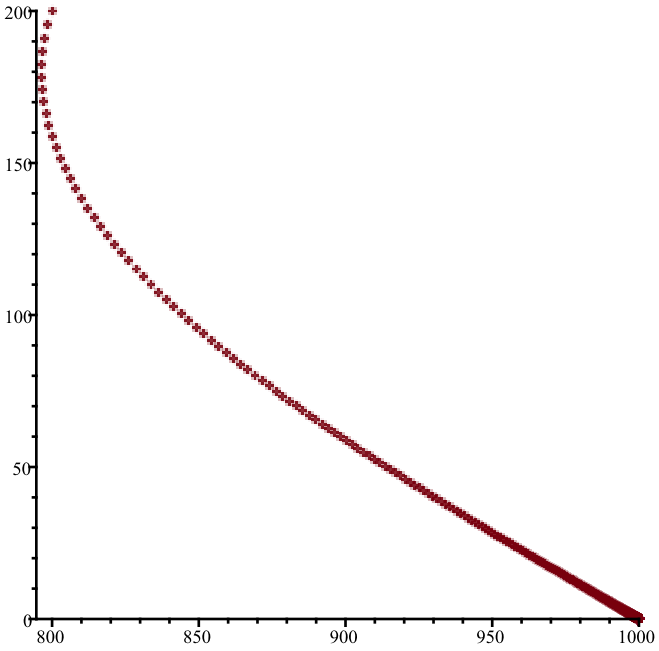
$$G1 := [-0.0009000000000\, s\, i + 6000 - 6\, s - 6\, i,\ 0.0009000000000\, s\, i - 3\, i]$$

$$\{[1000., 0.], [3333.333333, -1555.555556]\}$$

$$\{[1000., 0.]\}$$






> $G2 := SIRS\left(s, i, 0.9 \cdot \left(\frac{3}{1000}\right), 6, 3, 1000\right);$

$EquP(G2, [s, i]);$

$SEquP(G2, [s, i]);$

$TimeSeries(G2, [s, i], [800, 200], 0.01, 10, 1);$

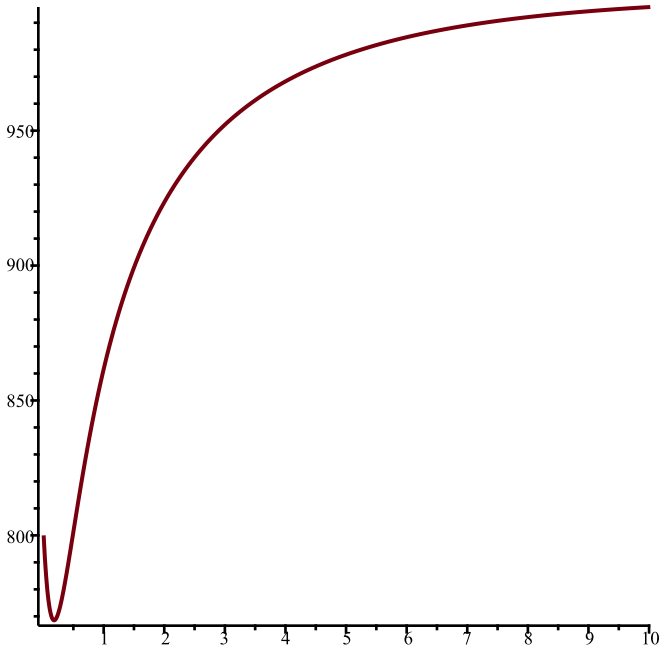
$TimeSeries(G2, [s, i], [800, 200], 0.01, 10, 2);$

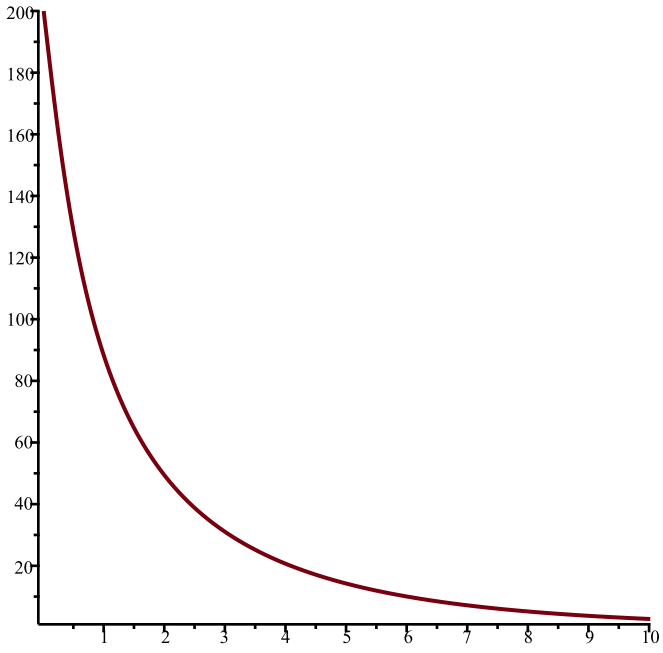
$PhaseDiag(G2, [s, i], [800, 200], 0.01, 10);$

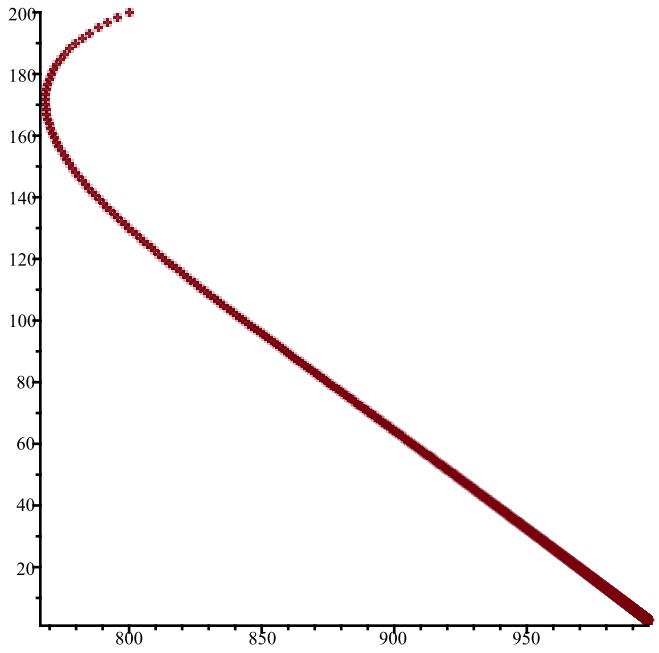
$$G2 := [-0.002700000000\, s\, i + 6000 - 6\, s - 6\, i,\ 0.002700000000\, s\, i - 3\, i]$$

$$\{[1000., 0.], [1111.111111, -74.07407407]\}$$

$$\{[1000., 0.]\}$$


950
900
850
800
1
2
3
4
5
6
7
8
9
10


200
180
160
140
120
100
80
60
40
20
1
2
3
4
5
6
7
8
9
10



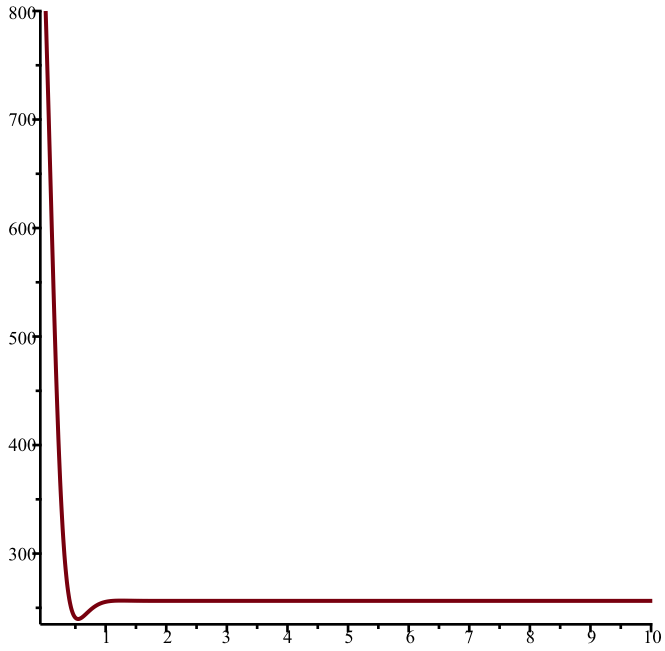

> $G3 := SIRS\left(s, i, 3.9 \cdot \left(\frac{3}{1000}\right), 6, 3, 1000\right);$

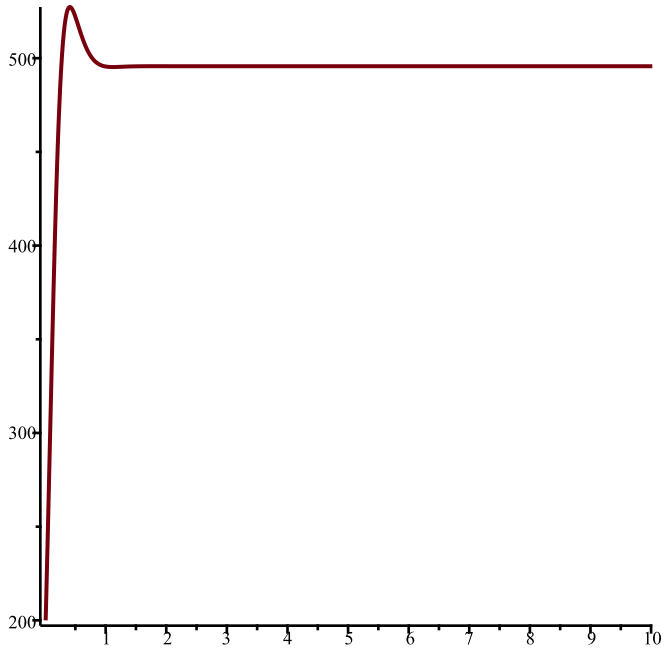
$EquP(G3, [s, i]);$
$SEquP(G3, [s, i]);$
$TimeSeries(G3, [s, i], [800, 200], 0.01, 10, 1);$
$TimeSeries(G3, [s, i], [800, 200], 0.01, 10, 2);$
$PhaseDiag(G3, [s, i], [800, 200], 0.01, 10);$

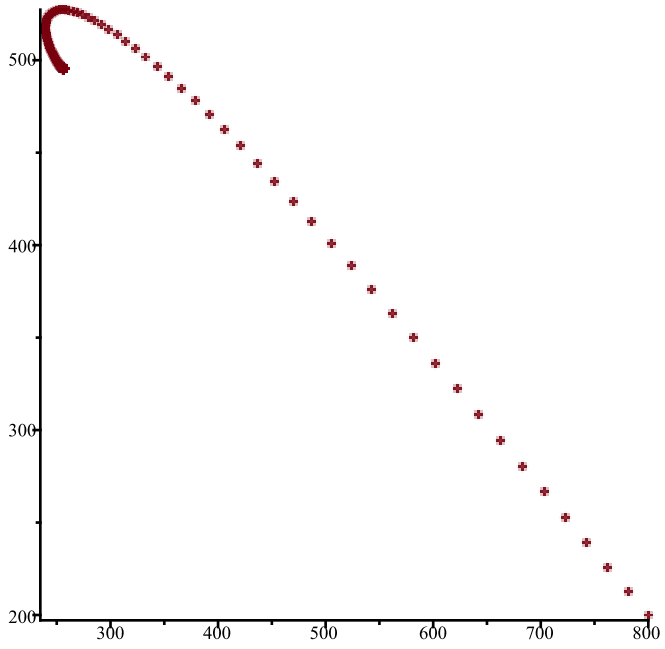
$$G3 := [-0.01170000000\, s\, i + 6000 - 6\, s - 6\, i,\ 0.01170000000\, s\, i - 3\, i]$$

$$\{[256.4102564, 495.7264957], [1000., 0.]\}$$

$$\{[256.4102564, 495.7264957]\}$$






> *#1 (iii)*

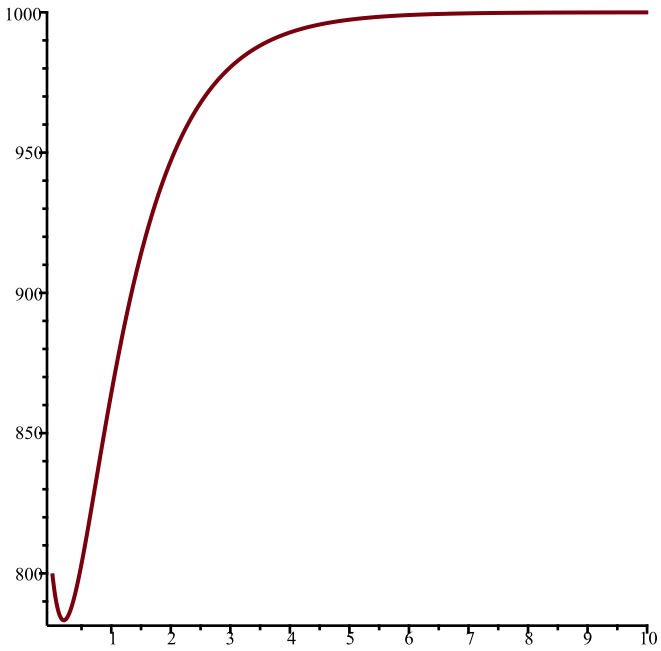
$$H1 := SIRS\left(s, i, 0.3 \cdot \left(\frac{4}{1000}\right), 1, 4, 1000\right);$$

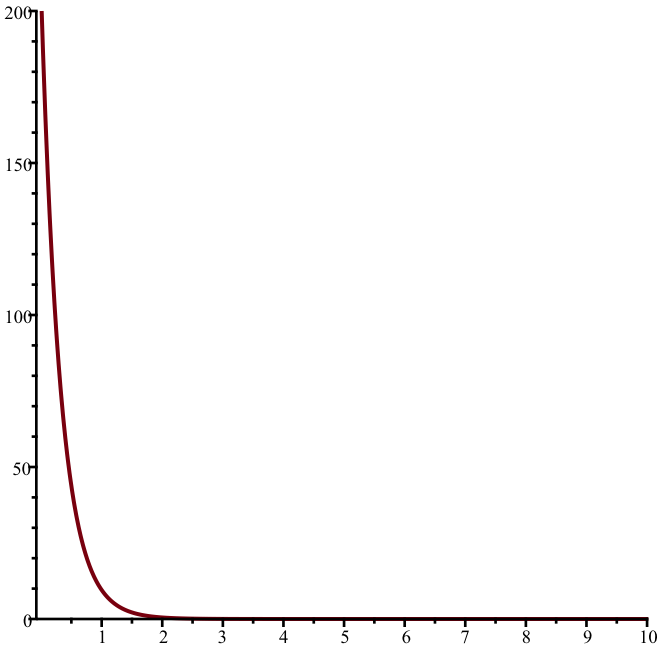
$EquP(H1, [s, i]);$
$SEquP(H1, [s, i]);$
$TimeSeries(H1, [s, i], [800, 200], 0.01, 10, 1);$
$TimeSeries(H1, [s, i], [800, 200], 0.01, 10, 2);$
$PhaseDiag(H1, [s, i], [800, 200], 0.01, 10);$

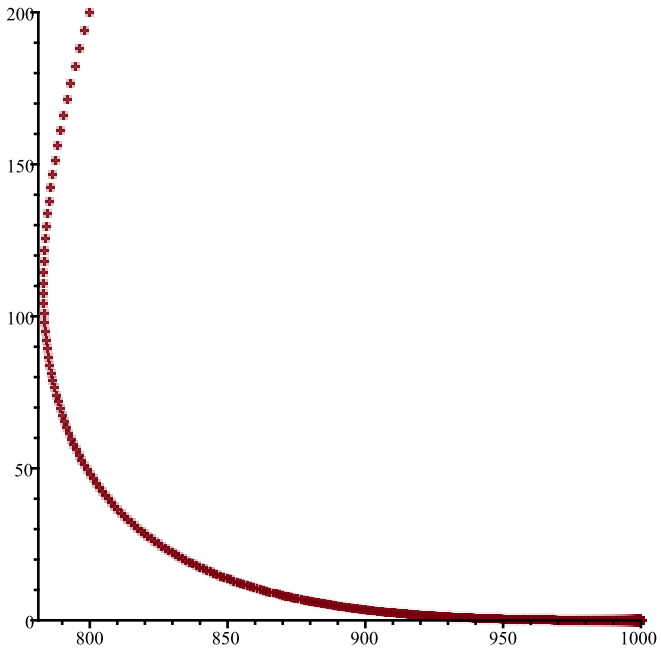
$$H1 := [-0.001200000000\, s\, i + 1000 - s - i,\ 0.001200000000\, s\, i - 4\, i]$$

$$\{[1000., 0.], [3333.333333, -466.6666667]\}$$

$$\{[1000., 0.]\}$$


1000
950
900
850
800
1
2
3
4
5
6
7
8
9
10


200
150
100
50
0
1
2
3
4
5
6
7
8
9
10



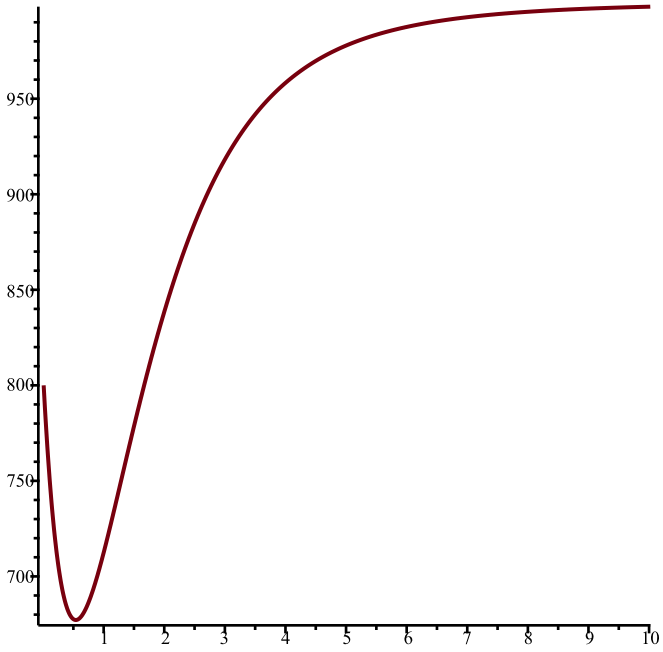

> $H2 := SIRS\left(s, i, 0.9 \cdot \left(\frac{4}{1000}\right), 1, 4, 1000\right);$

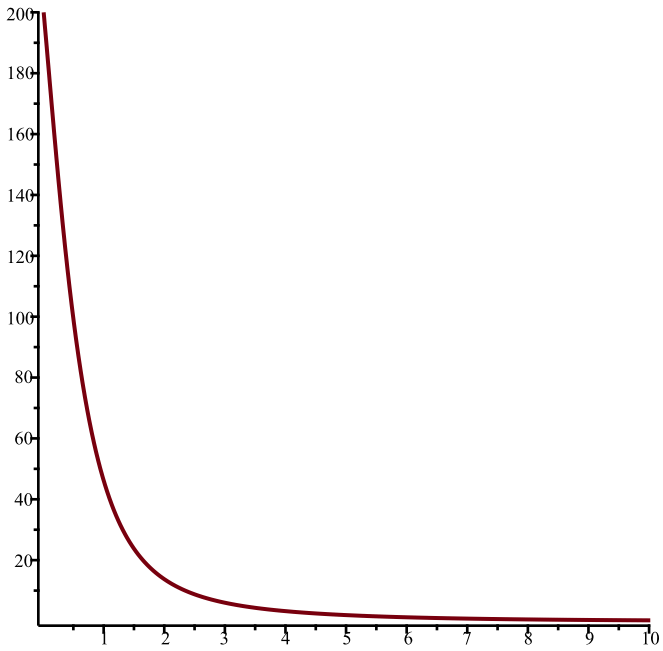
$EquP(H2, [s, i]);$
$SEquP(H2, [s, i]);$
$TimeSeries(H2, [s, i], [800, 200], 0.01, 10, 1);$
$TimeSeries(H2, [s, i], [800, 200], 0.01, 10, 2);$
$PhaseDiag(H2, [s, i], [800, 200], 0.01, 10);$

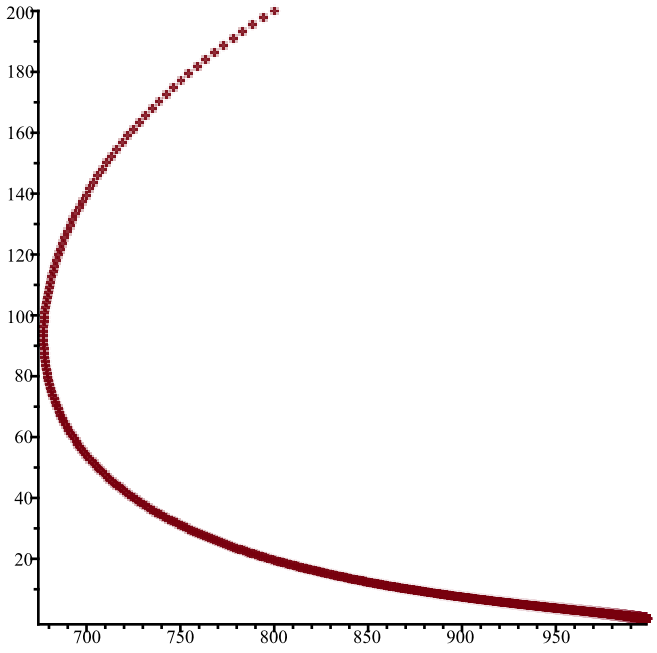
$$H2 := [-0.003600000000\, s\, i + 1000 - s - i,\ 0.003600000000\, s\, i - 4\, i]$$

$$\{[1000., 0.], [1111.111111, -22.22222222]\}$$

$$\{[1000., 0.]\}$$






> $H3 := SIRS\left(s, i, 3.9 \cdot \left(\frac{4}{1000}\right), 1, 4, 1000\right);$

$EquP(H3, [s, i]);$

$SEquP(H3, [s, i]);$

$TimeSeries(H3, [s, i], [800, 200], 0.01, 10, 1);$

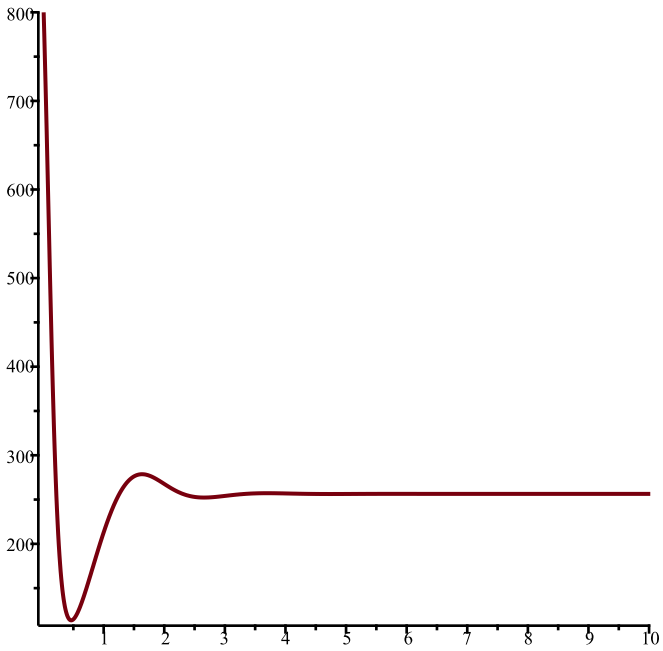
$TimeSeries(H3, [s, i], [800, 200], 0.01, 10, 2);$

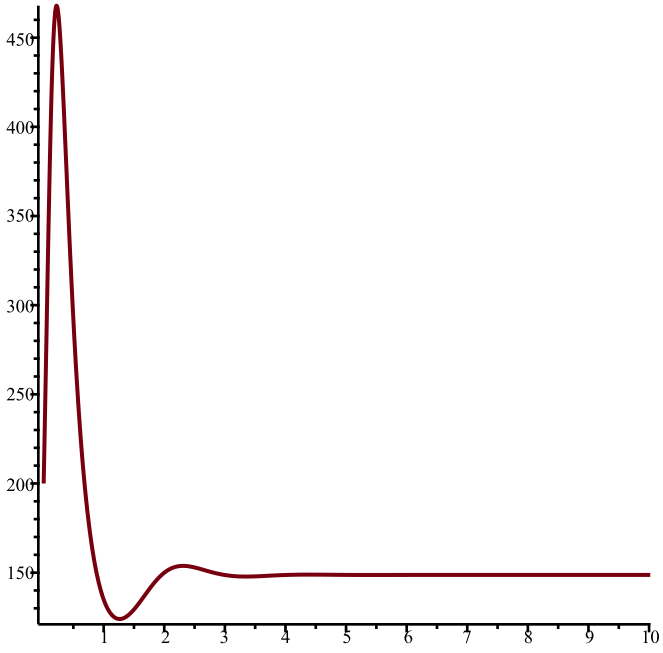
$PhaseDiag(H3, [s, i], [800, 200], 0.01, 10);$

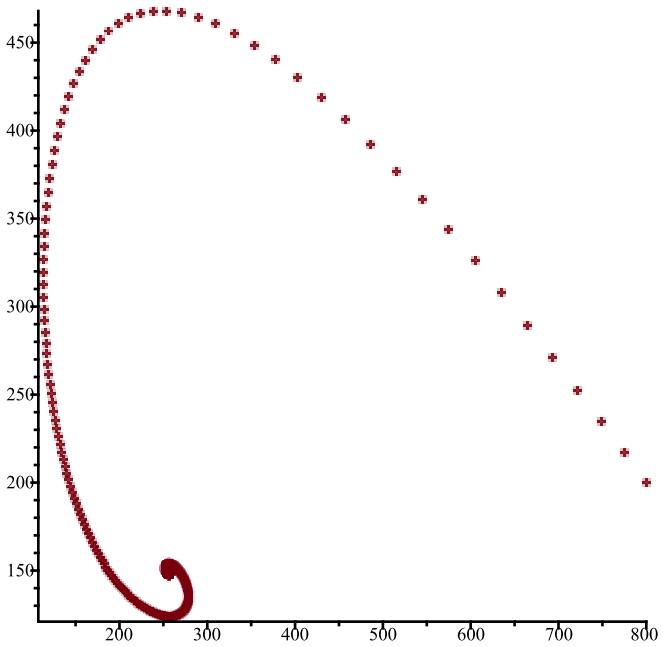
$$H3 := [-0.01560000000\, s\, i + 1000 - s - i,\ 0.01560000000\, s\, i - 4\, i]$$

$$\{[256.4102564, 148.7179487], [1000., 0.]\}$$

$$\{[256.4102564, 148.7179487]\}$$


800
700
600
500
400
300
200
1
2
3
4
5
6
7
8
9
10


450
400
350
300
250
200
150
1
2
3
4
5
6
7
8
9
10




> *#1 (iv)*

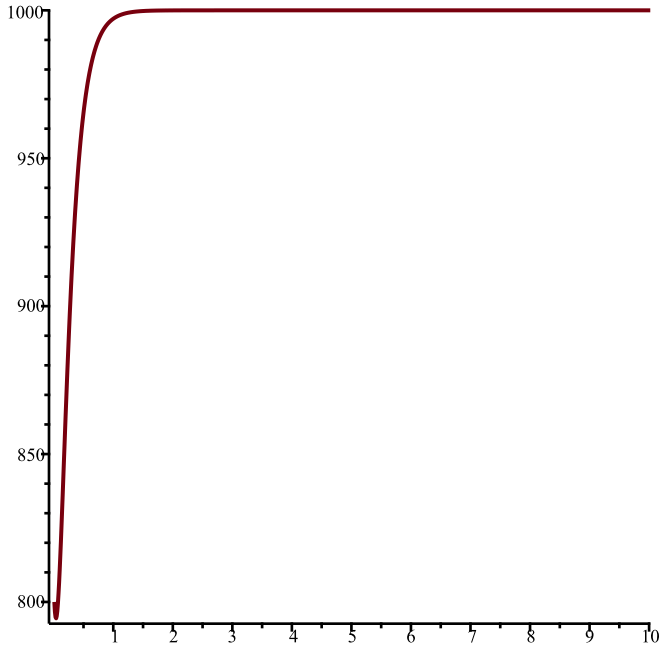
$$K1 := SIRS\left(s, i, 0.3 \cdot \left(\frac{7}{1000}\right), 10, 7, 1000\right);$$

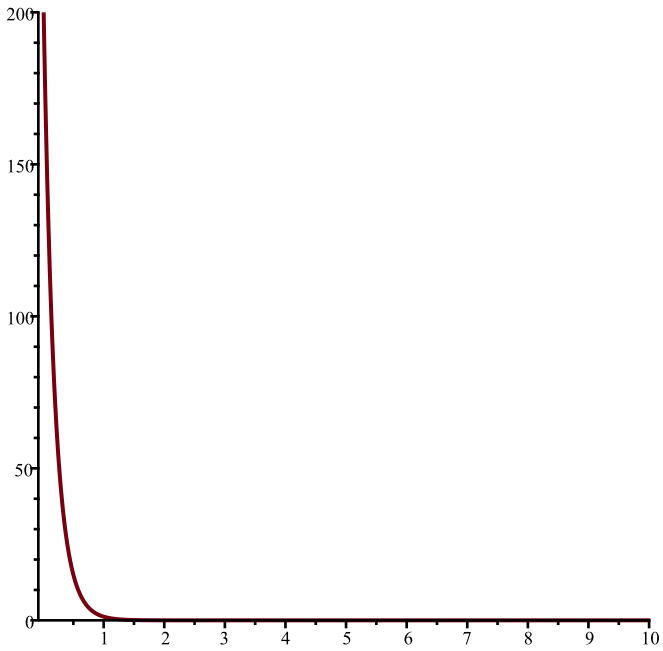
$EquP(K1, [s, i]);$
$SEquP(K1, [s, i]);$
$TimeSeries(K1, [s, i], [800, 200], 0.01, 10, 1);$
$TimeSeries(K1, [s, i], [800, 200], 0.01, 10, 2);$
$PhaseDiag(K1, [s, i], [800, 200], 0.01, 10);$

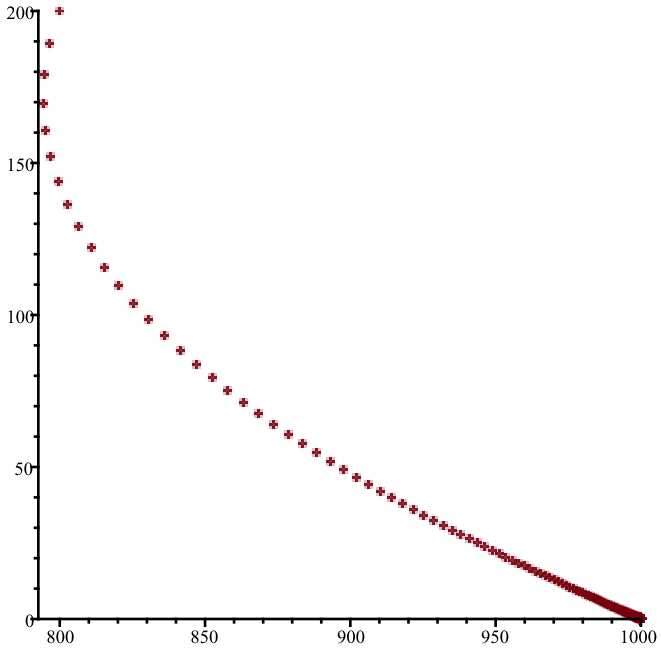
$$K1 := [-0.002100000000\, s\, i + 10000 - 10\, s - 10\, i,\ 0.002100000000\, s\, i - 7\, i]$$

$$\{[1000., 0.], [3333.333333, -1372.549020]\}$$

$$\{[1000., 0.]\}$$





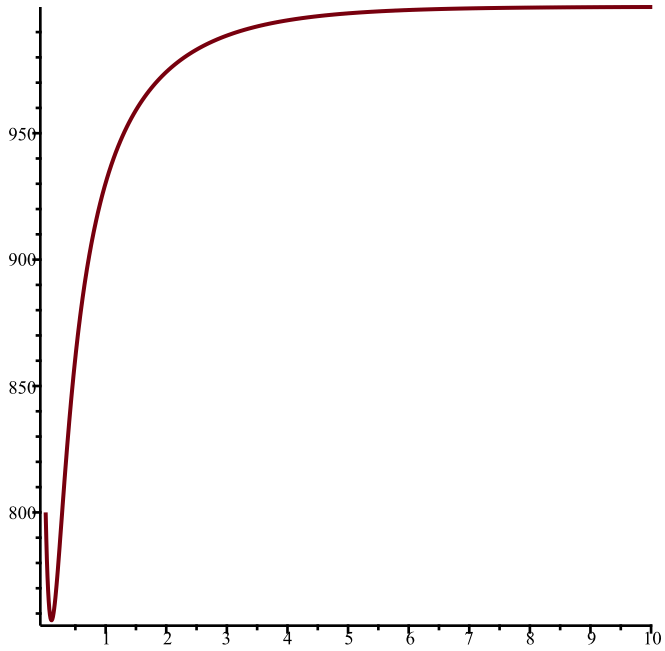

> $K2 := SIRS\left(s, i, 0.9 \cdot \left(\frac{7}{1000}\right), 10, 7, 1000\right);$

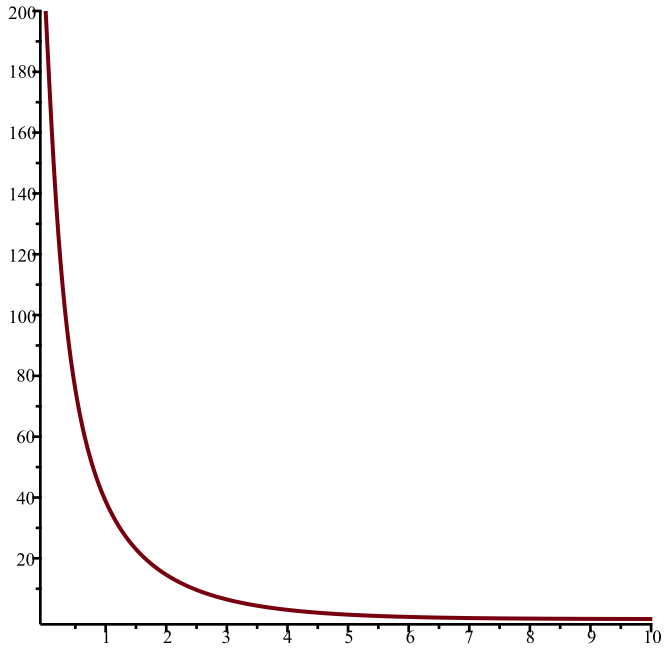
$EquP(K2, [s, i]);$
$SEquP(K2, [s, i]);$
$TimeSeries(K2, [s, i], [800, 200], 0.01, 10, 1);$
$TimeSeries(K2, [s, i], [800, 200], 0.01, 10, 2);$
$PhaseDiag(K2, [s, i], [800, 200], 0.01, 10);$

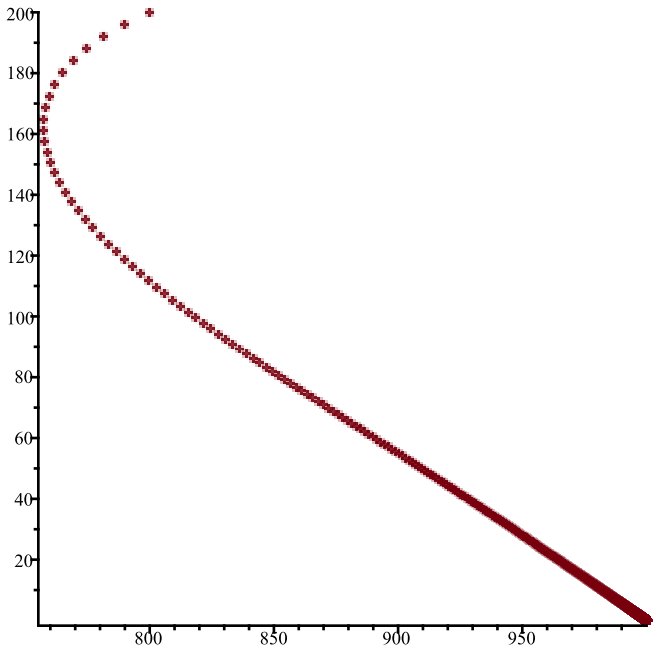
$$K2 := [-0.006300000000\, s\, i + 10000 - 10\, s - 10\, i,\ 0.006300000000\, s\, i - 7\, i]$$

$$\{[1000., 0.], [1111.111111, -65.35947712]\}$$

$$\{[1000., 0.]\}$$


950
900
850
800
1
2
3
4
5
6
7
8
9
10


200
180
160
140
120
100
80
60
40
20
1
2
3
4
5
6
7
8
9
10



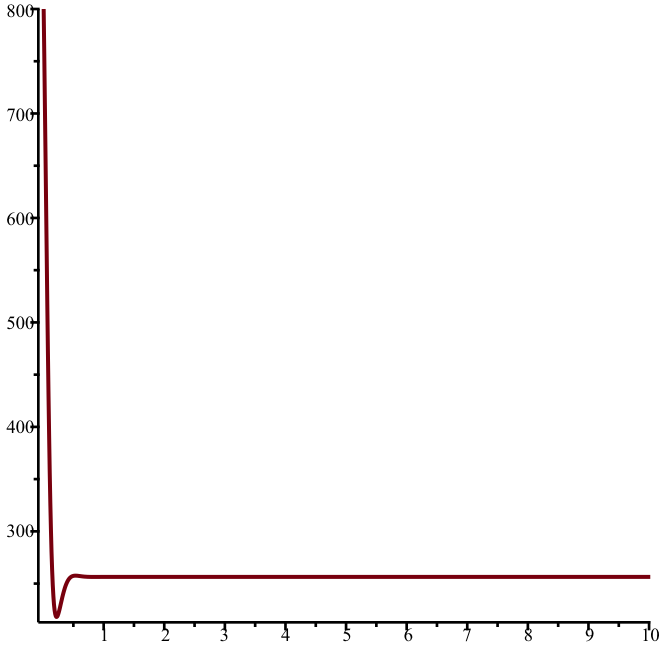

> $K3 := SIRS\left(s, i, 3.9 \cdot \left(\frac{7}{1000}\right), 10, 7, 1000\right);$

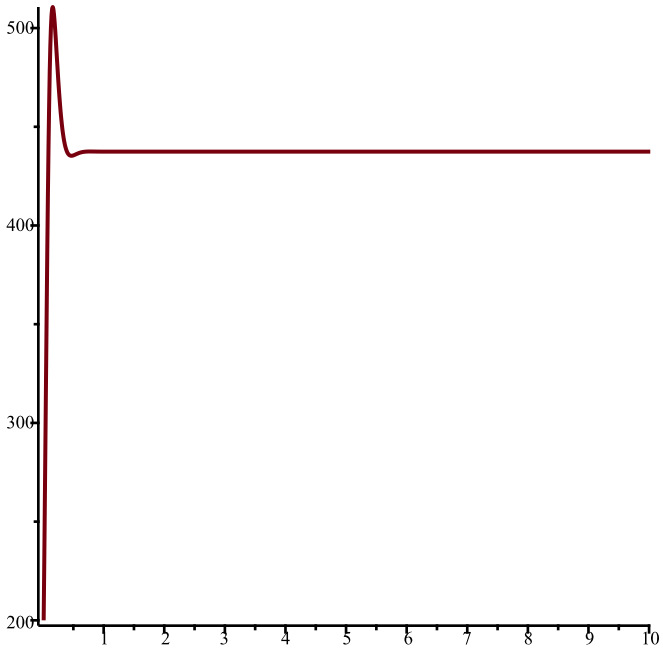
$EquP(K3, [s, i]);$
$SEquP(K3, [s, i]);$
$TimeSeries(K3, [s, i], [800, 200], 0.01, 10, 1);$
$TimeSeries(K3, [s, i], [800, 200], 0.01, 10, 2);$
$PhaseDiag(K3, [s, i], [800, 200], 0.01, 10);$

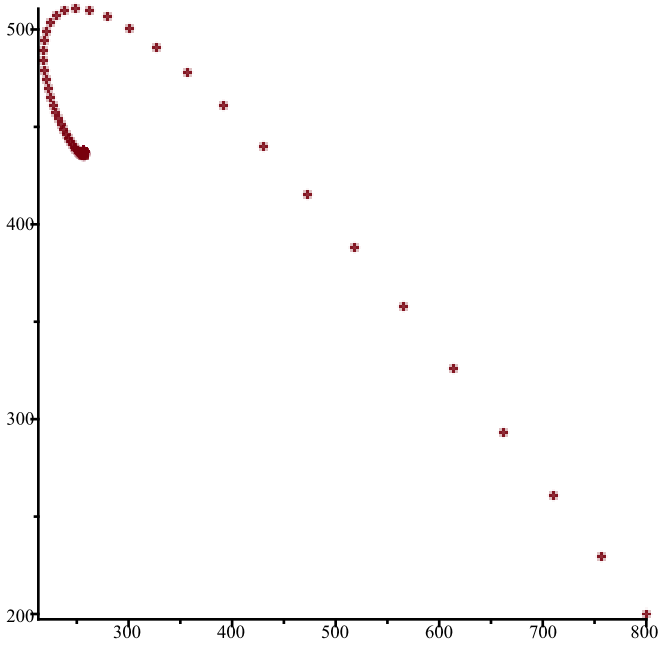
$$K3 := [-0.02730000000\, s\, i + 10000 - 10\, s - 10\, i,\ 0.02730000000\, s\, i - 7\, i]$$

$$\{[256.4102564, 437.4057315], [1000., 0.]\}$$

$$\{[256.4102564, 437.4057315]\}$$






> *#2 (i)*

$F := RandNice([x, y], 3);$

$EquP(F, [x, y]);$

$SEquP(F, [x, y]);$

$$F := [(2 - 2x - 3y)(2 - x - 3y), (1 - x - 2y)(3 - 2x - 2y)]$$

$$\left\{[-1, 1], [1, 0], \left[\frac{5}{2}, -1\right], \left[\frac{5}{4}, \frac{1}{4}\right]\right\}$$

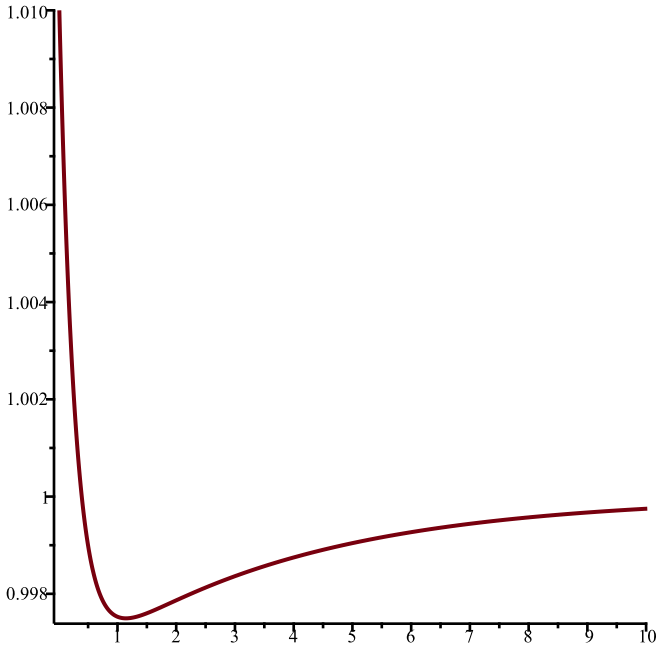
$$\{[1., 0.]\}$$

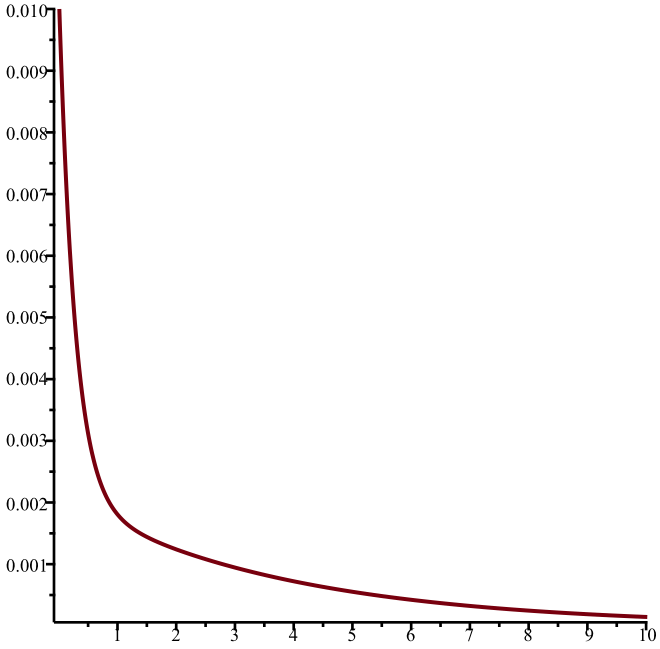
**(2)**

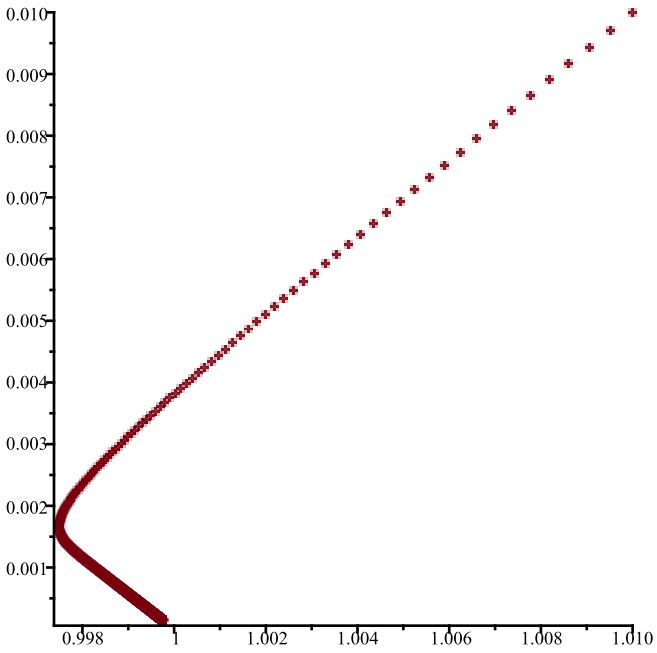
> $TimeSeries(F, [x, y], [1.01, 0.01], 0.01, 10, 1);$

$TimeSeries(F, [x, y], [1.01, 0.01], 0.01, 10, 2);$

$PhaseDiag(F, [x, y], [1.01, 0.01], 0.01, 10);$


1.010
1.008
1.006
1.004
1.002
1
0.998
1
2
3
4
5
6
7
8
9
10


0.010
0.009
0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
1
2
3
4
5
6
7
8
9
10




> *#2 (ii)*

$F := RandNice([x, y], 3);$

$EquP(F, [x, y]);$

$SEquP(F, [x, y]);$

$$F := [(1 - 2x - y)(1 - 2x - 2y), (3 - 3x - 2y)(2 - x - y)]$$

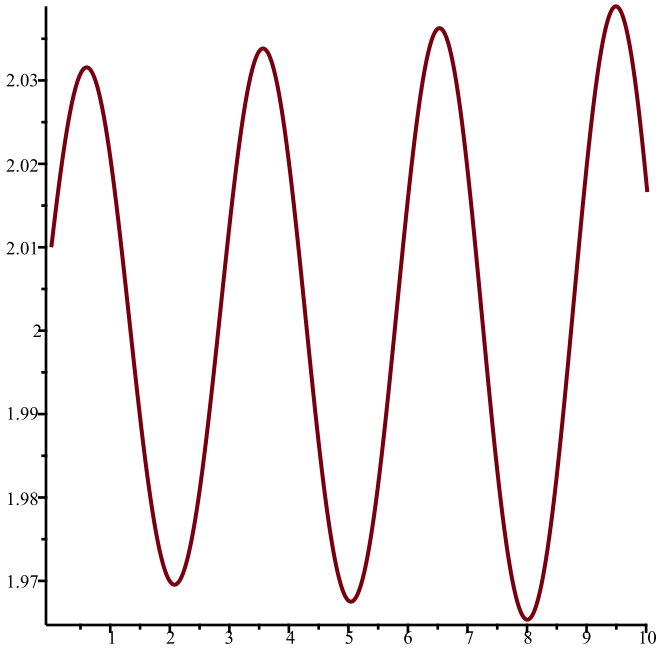
$$\left\{[-1, 3], \left[2, -\frac{3}{2}\right]\right\}$$

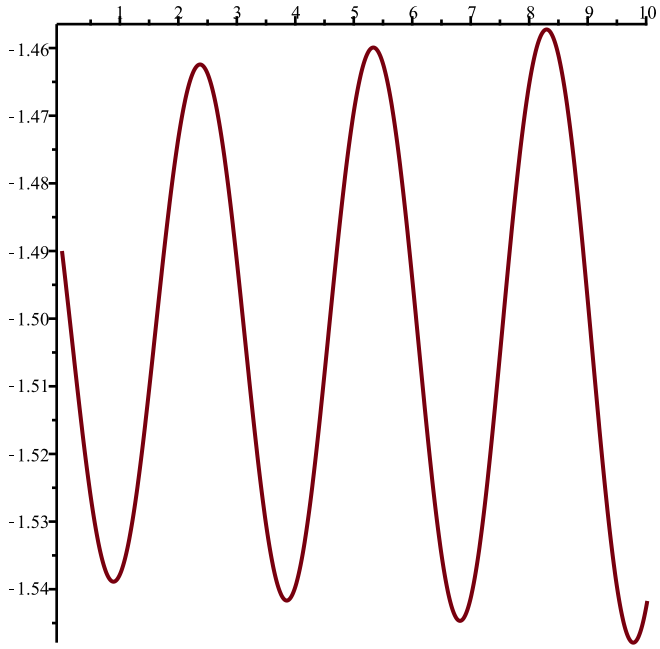
$$\{[2., -1.500000000]\}$$ **(3)**

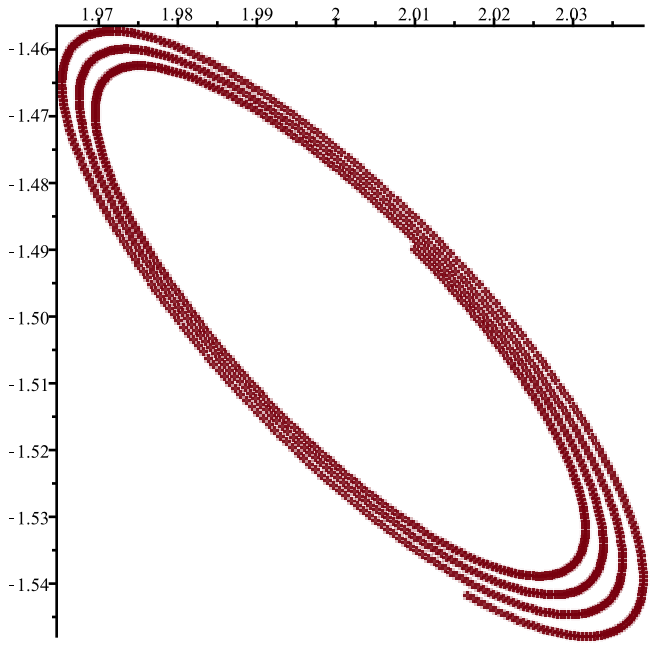
> $TimeSeries(F, [x, y], [2.01, -1.49], 0.01, 10, 1);$

$TimeSeries(F, [x, y], [2.01, -1.49], 0.01, 10, 2);$

$PhaseDiag(F, [x, y], [2.01, -1.49], 0.01, 10);$






> #2 (iii)

$F := RandNice([x, y], 3);$

$EquP(F, [x, y]);$

$SEquP(F, [x, y]);$

$$F := [(3 - x - 2y)(1 - 2x - 3y), (3 - 3x - 2y)(1 - 3x - 3y)]$$

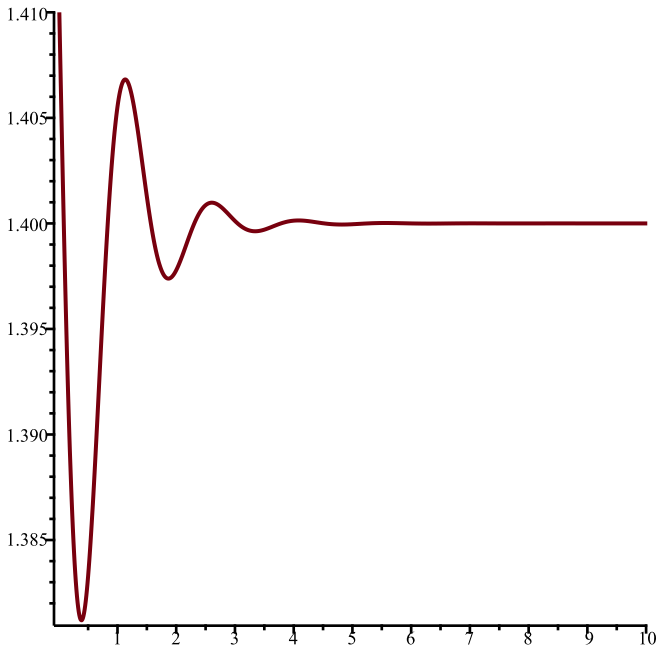
$$\left\{\left[0, \frac{1}{3}\right], \left[0, \frac{3}{2}\right], \left[-\frac{7}{3}, \frac{8}{3}\right], \left[\frac{7}{5}, -\frac{3}{5}\right]\right\}$$

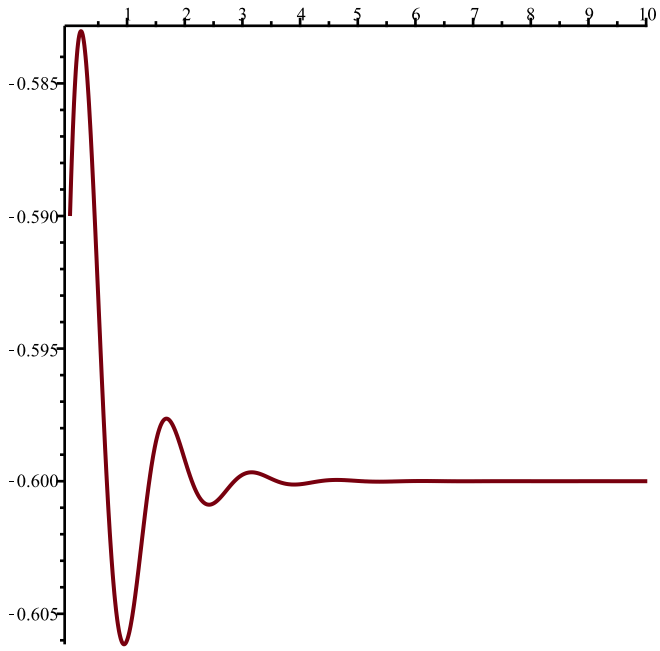
$$\{[-2.333333333, 2.666666667], [1.400000000, -0.6000000000]\}$$ **(4)**

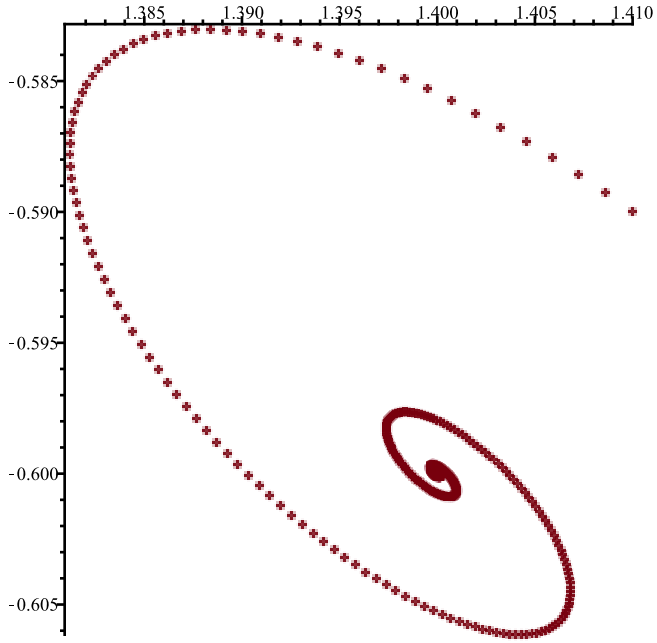
> $TimeSeries(F, [x, y], [1.41, -0.59], 0.01, 10, 1);$

$TimeSeries(F, [x, y], [1.41, -0.59], 0.01, 10, 2);$

$PhaseDiag(F, [x, y], [1.41, -0.59], 0.01, 10);$


1.410
1.405
1.400
1.395
1.390
1.385
1
2
3
4
5
6
7
8
9
10


1
2
3
4
5
6
7
8
9
10
-0.585
-0.590
-0.595
-0.600
-0.605




> #3

$$Orbk\left(4, z, \frac{(3+z[2]+z[3]+z[4])}{(1+z[1]+z[3])}, [1.0, 0.5, 1.2, 0.05], 1000, 1010\right);$$

[0.9314401593, 4.556182446, 0.9314401593, 4.556182446, 0.9314401593, 4.556182446, 0.9314401593, 4.556182446, 0.9314401593, 4.556182446, 0.9314401593] **(5)**

> $$Orbk\left(4, z, \frac{(3+z[2]+z[3]+z[4])}{(1+z[1]+z[3])}, [1.0, 1.0, 1.0, 1.0], 1000, 1010\right);$$

[1.567364551, 2.139552296, 1.567364551, 2.139552296, 1.567364551, 2.139552296, 1.567364551, 2.139552296, 1.567364551, 2.139552296, 1.567364551] **(6)**

> $$Orbk\left(4, z, \frac{(3+z[2]+z[3]+z[4])}{(1+z[1]+z[3])}, [2.0, 2.0, 2.0, 2.0], 1000, 1010\right);$$

[1.866943730, 1.780228265, 1.866943730, 1.780228265, 1.866943730, 1.780228265, 1.866943730, 1.780228265, 1.866943730, 1.780228265, 1.866943730] **(7)**

> $$Orbk\left(4, z, \frac{(3+z[2]+z[3]+z[4])}{(1+z[1]+z[3])}, [0.0, 0.0, 0.0, 0.0], 1000, 1010\right);$$

[1.011465591, 3.921540044, 1.011465591, 3.921540044, 1.011465591, 3.921540044, 1.011465591, 3.921540044, 1.011465591, 3.921540044, 1.011465591] **(8)**

> $$Orbk\left(4, z, \frac{(3+z[2]+z[3]+z[4])}{(1+z[1]+z[3])}, [0.025, 0.025, 0.025, 0.025], 1000, 1010\right);$$

[1.031858535, 3.790348627, 1.031858535, 3.790348627, 1.031858535, 3.790348627, 1.031858535, 3.790348627, 1.031858535, 3.790348627, 1.031858535] **(9)**

> $$SFP\left(\frac{(3+z[2]+z[3]+z[4])}{(1+z[1]+z[3])}, [z[1], z[2], z[3], z[4]]\right);$$

*bad input*

*FAIL* **(10)**

```
> SEquP( (3 + z[2] + z[3] + z[4]) / (1 + z[1] + z[3]) , [z[1], z[2], z[3], z[4]] );
```

*bad input*

*FAIL* **(11)**

```
> #It does not seem like there is a stable equilibrium point for the given difference equation
```