Dynamic Modeling HW24- Okay to Post

1) I got 2 wrong because I overcomplicated it. I was thinking more in a physics context, and began to think in terms of force diagrams instead of kinematics. If I thought strictly in kinematics I would have seen that it was similar to 1, and could have done it easily

2) $a'''(t) = 120\, m/s^5$, $v(0)=0$, $a(0)$, $a'(0)=0$, $a''(0)=0$

$\int a'''(t) = \int 120\, dt$

$a''(t) = 120t + c$

$0 = 0 + c \Rightarrow c = 0$

$\int a''(t) = \int 120t\, dt$

$a'(t) = 60t^2 + c$

$0 = 0 + c \Rightarrow c = 0$

$\int a'(t) = \int 60t^2 dt$

$a(t) = 20t^3 + c$

$0 = 0 + c \Rightarrow c = 0$

$a(t) = \int v'(t) = \int 20t^3 dt$

$v(t) = 5t^4 + c$

$0 = 0 + c \Rightarrow 0 = c$

$v(t) = \int x'(t) = \int 5t^4 dt$

$x(t) = t^5 + c$

$x(t) - c = t^5$

$\Delta x = t^5$

$\Delta x(2) = 2^5$

$\boxed{\Delta x(2) = 32\, m}$

3) (i) $\int x''(t) = \int -g\, dt$ * $10 = g$

$x'(t) = -gt + c$ $x'(0) = 0$

$0 = 0 + c \Rightarrow 0$

$\int x'(t) = \int -gt$

$x(t) = \frac{-g}{2}t^2 + c$ $x(0) = 100$

$100 = 0 + c \Rightarrow c = 100$

$x(t) = 100 - \frac{g}{2}t^2$

$0 = 100 - \frac{g}{2}t^2$

$-100 = -\frac{10}{2}t^2$

$\frac{100}{5} = \frac{5t^2}{5}$

$\sqrt{20} = \sqrt{t^2}$

$\boxed{t = 4.47s}$

(ii) $m x''(t) = 2m x'(t) - mg$ → *solved on Maple

$x''(t) = 2x'(t) - g$

4) (a) $x(n) = f(x(n-1))$

(b) $x(0) = x_0$

$[x_0, f(x_0), f(f(x_0)), f(f(f(x_0))), \ldots f(K-1)]$

(c) $f(K) = K$ where $f(x)$ is the underlying function of $x(n) = f(x(n-1))$

and $\lim_{n\to\infty} f(x(n-1)) = K$

(d) $f(K) = K$ where $f(x)$ is the underlying function of $x(n) = f(x(n-1))$

and for $f(x(n-1)) + \Delta h$ which is around K, $\lim_{n\to\infty} f(x(n-1)) + \Delta h = K$

5) (a) You can find the stable discrete equilibria using Orb by inputting the function, initial conditions, and which discrete times you want Orb to output. Orb numerically finds the value of $x(n)$ using the function given, so in order to find stable equilibria, you want your K1 and K2 to be large, so you can see the behavior of the function as $n \to \infty$. If the value of the function stays at one value, then that is the stable equilibrium. If it oscillates, you will need to use something like SFP to determine which of the fixed points are stable

(b) You find the set of fixed points by setting the underlying function $f(x)$ equal to x, and solving for x

(c) You find the subset of stable fixed points by taking the derivative of $f(x)$ and plugging the fixed points into $f'(x)$. If $|f'(x_0)| < 1$, then x_0 is stable

(d) (i) (b) $f(x) = \frac{x+1}{x+2}$

$$x = \frac{x+1}{x+2}$$

$$x(x+2) = x+1$$

$$x^2 + 2x = x+1$$

$$\to x^2 + x - 1 = 0$$

$$x = \frac{-1 \pm \sqrt{1-4(1)(-1)}}{2(1)}$$

$$\boxed{x = -\frac{1}{2} \pm \frac{\sqrt{5}}{2}}$$

(c) $$f'(x) = \frac{(x+2)-(x+1)}{(x+2)^2} = \frac{1}{(x+2)^2}$$

$$\boxed{x = -\frac{1}{2} + \frac{\sqrt{5}}{2} \text{ is stable}}$$

$$f'\left(-\frac{1}{2} - \frac{\sqrt{5}}{2}\right) = \frac{1}{\left(-\frac{1}{2} - \frac{\sqrt{5}}{2} + 2\right)^2} = 6.85 \times$$

$$f'\left(-\frac{1}{2} + \frac{\sqrt{5}}{2}\right) = \frac{1}{\left(-\frac{1}{2} + \frac{\sqrt{5}}{2} + 2\right)^2} = 0.145 \checkmark$$

(ii) (b) $f(x) = \frac{5}{2}x(1-x)$

$x = \frac{5}{2}x(1-x)$

$x = \frac{5}{2}x - \frac{5}{2}x^2$

$\frac{5}{2}x^2 - \frac{3}{2}x = 0$

$x\left(\frac{5}{2}x - \frac{3}{2}\right) = 0$ → $x = 0, \frac{3}{5}$

(c) $f'(x) = \frac{5}{2} - 5x$

$f'(0) = \frac{5}{2}$ ✗

$f'\left(\frac{3}{5}\right) = \frac{5}{2} - 3 = -\frac{1}{2}$ ✓

$x = \frac{3}{5}$ is stable

(iii) (b) $f(x) = \frac{7}{2}x(1-x)$

$x = \frac{7}{2}x - \frac{7}{2}x^2$

$2\left(\frac{7}{2}x^2 - \frac{5}{2}x = 0\right)$

$7x^2 - 5x = 0$

$x(7x-5) = 0$ → $x = 0, \frac{5}{7}$

(c) $f'(x) = \frac{7}{2} - 7x$

$f'(0) = \frac{7}{2}$ ✗

$f'\left(\frac{5}{7}\right) = \frac{7}{2} - 5 = -\frac{3}{2}$ ✗

Neither point is stable

```
> #Nikita John, Assignment 24
> ###########################################################################
## 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, AllenSIRg, Hassell, HW, HWg, May75, NicholsonBailey, NicholsonBaileyM,
    RT, Valery`) :

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, OrbF, Orbk, OrbkF, 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/AllenSIR.pdf`) :
print(`with parameters a,b,c. try:`) :
print(`AllenSIR(1,1/3,1/3,x,y);`) :
    print(`WARNING: To get the long-term behavior, use OrbF NOT Orb (or else Maple will go
    for ever)`) :
print(`Try the following: `) :
    print(`F:=AllenSIR(1,0.3,0.3,x,y);a:=OrbF(F,[x,y],[1.0, 2.0],1000,1010)[-1];evalf(subs({x=
    a[1],y=a[2]},F)-a);`) :


elif nargs = 1 and args[1] = AllenSIRg then
    print(`AllenSIRg(a,b,c,alpha,beta,x,y): The GENERALIZED Linda Allen discrete SIR model
    given in https://sites.math.rutgers.edu/~zeilberg/Bio21/AllenSIR.pdf`) :
print(`with parameters a,b,c. Try:`) :
print(`where the expnents of x_n and y_n are alpha and beta.  Note that`) :
print(`AllenSIRg(a,b,c,1,1,x,y) is the same as AllenSIR(a,b,c,x,y): Try:`) :
print(`AllenSIRg(1,1/3,1/3,1.2,1.2,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] = Hassell then
    print(`Hassell(L,a,b,N): The discrete-time, single-species dynamical model of Hassell (1975)
    given by Eq. (13) in Edelstein-Keshet section 3.1 (p. 75)`) :
    print(`where the variable is N (the population), and the parameters are L (called Lambda
    there), a, and b`) :
print(`Try:`) :
print(`Hassell(L,a,b,N);`) :
print(`Hassell(20,3,5,N);`) :

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] = May75 then
    print(`May75(r,K,N): The discrete-time, single-species dynamical model of May (1975) given
    by Eq. (8) in Edelstein-Keshet section 3.1 (p. 75)`) :
print(`where the variable is N (the population), and the parameters are r and K`) :
print(`Try:`) :
print(`May75(r,K,N);`) :
print(`May75(3/2,2,N);`) :


elif nargs = 1 and args[1] = NicholsonBailey then
    print(`NicholsonBailey(L,a,c): The discrete-time, double-species dynamical model of
    Nicholson and Bailey (1935), given by Eqs. (21a)(21b) in Edelstein-Keshet section 3.2 (p. 81)
    `) :
    print(`where the variables are  N (hosts) and parasites (P)  and the parameters are L (called
    Lambda there), a, and c`) :
print(`Try:`) :
print(`NicholsonBailey(L,a,c,N,P);`) :
print(`NicholsonBailey(2,0.068,1,N,P);`) :
```

```
elif nargs = 1 and args[1] = NicholsonBaileyM then
    print(`NicholsonBaileyM(a,r,K,N,B): The discrete-time, double-species dynamical model of the MODIFIED Nicholson and Bailey model (1935), given by Eqs. (28a)(28b) in Edelstein-Keshet section 3.4 (p. 84)`) :
print(`where the variables are  N (hosts) and parasites (P)  and the parameters are r and K`) :
print(`Try:`) :
print(`NicholsonBaileyM(r,a,K,N,P);`) :
print(`plot(OrbF(NicholsonBaileyM(0.5,0.11,15,N,P),[N,P],[3.,5.],1,1000),style=point);`) :


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] = OrbF then
print(`OrbF(F,x,x0,K1,K2):  Same as Orb(F,x,x0,K1,K2) but in floating-point`) :
    print(`Inputs a transformation F in the list of variables x with initial point pt, outputs the trajectory `) :
    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(`OrbF(5/2*x*(1-x),[x], [0.5], 1000,1010);`) :
print(`OrbF((1+x+y)/(2+x+y),[x,y], [2.,3.], 1000,1010);`) :
print(`OrbF(AllenSIR(1,1/3,1/3,x,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] = OrbkF then
    print(`OrbkF(k,z,f,INI,K1,K2): Same as Orbk(k,z,f,INI,K1,K2), but in floating-point (to get
    around Maple's annoying habit not to automatically convert to floating point exp
    (floatingpoint)`) :
print(`To investigate the long-term behavior Linda Allen's Conjecture 2 of `) :
print(`https://sites.math.rutgers.edu/~zeilberg/Bio21/AllenSIR.pdf`) :
print(`with initial conditions x(0)=0.3, x(1)=0.4, a=3, b=2 Type: `) :
    print(`a:=0.3; b:=0.2; OrbkF(2,z,z[1]*(1-b)+ (1-z[1])*(1-exp(-a*z[2])),[0.3,0.4],1000,
    1010);`) :
print(`then type `) :
print(`solve(b*y-(1-y)*(1-exp(-a*y)),y);`) :


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] = Valery then
    print(`Valery(L,a,b,N): The discrete-time, single-species dynamical model of Valery,
    Gradwell, and Hassel (1973) given by Eq. (2) in Edelstein-Keshet section 3.1 (p. 74)`) :
    print(`where the variable is N (the population), and the parameters are L (called Lambda
    there), is the reproduction rate, and a (called alpha there) and b`) :
    print(`are the other two parameters such that  1/(a*N^(-b)) is the faction of the population
    that survives. L,a,b, can be symbolic or numeric`) :
print(`Try:`) :
print(`Valery(L,a,b,N);`) :
print(`Valery(3,2,1,N);`) :

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 - 1 do

x1 := [seq(subs( {seq(x[i2] = x1[i2], i2 = 1 ..nops(x))}, F[i1]), i1 = 1 ..nops(F))] :
 L := [op(L), expand(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:



#OrbkF(k,z,f,INI,K1,K2): Like Orbk(k,z,f,INI,K1,K2) but in floating-point
#OrbkF(1,z,5/2*z[1]*(1-z[1]),[0.5],1000,1010);  should be the same as
#OrbkF(5/2*z[1]*(1-z[1]),z[1],[0.5],1000,1010);
#Try:
#OrbkF(2,z,(5/4)*z[1]-(3/8)*z[2],[1,2],1000,1010);

OrbkF := 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 := evalf(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/AllenSIR.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:
```

#AllenSIRg(a,b,c,alpha,beta,x,y): The GENERALIZED Linda Allen discrete SIR model given in https://sites.math.rutgers.edu/~zeilberg/Bio21/AllenSIR.pdf
#with parameters a,b,c. Try:
#where the expnents of x_n and y_n are alpha and beta. Note that
#AllenSIRg(a,b,c,1,1,x,y) is the same as AllenSIR(a,b,c,x,y): Try:
#AllenSIRg(1,1/3,1/3,1.2,1.2,x,y);

```
AllenSIRg :=proc(a, b, c, alpha, beta, x, y)
[x^alpha*(1-b-c) + y^beta*(1-exp(-a*x)), (1-y^beta)*b + y^beta*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 paramers 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:


#Valery(L,a,b,N): The discrete-time, single-species dynamical model of Valery, Gradwell, and Hassel (1973) given by Eq. (2) in Edelstein-Keshet section 3.1 (p. 74)

#where the variable is N (the population), and the parameters are L (called Lambda there), is the reproduction rate, and a (called alpha there) and b

#are the other two parameters such that  1/(a*N^(-b)) is the faction of the population that survives. L,a,b, can be symbolic or numeric
#Try:
#Valery(L,a,b,N);
#Valery(3,2,1,N);
Valery := proc(L, a, b, N) :
[(L/a) * N^(1-b)] :
end:


#May75(r,K,N): The discrete-time, single-species dynamical model of May (1975) given by Eq. (8) in Edelstein-Keshet section 3.1 (p. 75)
#where the variable is N (the population), and the parameters are r and K
#Try:
#May75(r,K,N);
#May75(3/2,2,N);
May75 := proc(r, K, N) :
[N* exp(r* (1-N/K))] :
end:


#Hassell(L,a,b,N): The discrete-time, single-species dynamical model of Hassell (1975) given by Eq. (13) in Edelstein-Keshet section 3.1 (p. 75)

#where the variable is N (the population), and the parameters are L (called Lambda there), a, and b
#Try:
```

```
#Hassell(L,a,b,N);
#Hassell(20,3,5,N);
Hassell := proc(L, a, b, N) :
[L*N*(1 + a*N)^(-b)]:
end:
```

#NicholsonBailey(L,a,c): The discrete-time, double-species dynamical model of Nicholson and Bailey (1935), given by Eqs. (21a)(21b) in Edelstein-Keshet section 3.2 (p. 81)

#where the variables are N (hosts) and parasites (P) and the parameters are L (called Lambda there), a, and c

```
#Try:
#NicholsonBailey(L,a,c,N,P);
#NicholsonBailey(2,0.068,1,N,P);
NicholsonBailey := proc(L, a, c, N, P)
[L*N*exp(-a*P), c*N*(1-exp(-a*P))]:
end:
```

#NicholsonBaileyM(a,r,K,N,B): The discrete-time, double-species dynamical model of the MODIFIED Nicholson and Bailey model (1935), given by Eqs. (28a)(28b) in Edelstein-Keshet section 3.4 (p. 84)

```
#where the variables are N (hosts) and parasites (P) and the parameters are r and K
#Try:
#NicholsonBaileyM(r,a,K,N,P);
#NicholsonBaileyM(0.5,0.1,14,N,P);
NicholsonBaileyM := proc(r, a, K, N, P)
[N*exp(r*(1-N/K)-a*P), N*(1-exp(-a*P))]:
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)**

```
> #Taking 10 as g
  X := diff(x(t), t, t) - 2·diff(x(t), t) + 10 = 0;
  dsolve({X, x(0) = 100, D(x)(0) = 0}, x(t));
```

$$X := \frac{d^2}{dt^2} x(t) - 2 \frac{d}{dt} x(t) + 10 = 0$$

$$x(t) = -\frac{5 e^{2t}}{2} + 5t + \frac{205}{2} \quad \textbf{(2)}$$

> $evalf\left(solve\left(-\frac{5 e^{2t}}{2} + 5t + \frac{205}{2} = 0, t\right)\right);$

$$-20.50000000, 1.90112928 \quad \textbf{(3)}$$

> *#It will take 1.9 seconds to reach the ground*

> *#5 (i) (a)*

$Orb\left(\left[\frac{(x+1)}{(x+2)}\right], [x], [1.0], 1000, 1010\right);$

[[0.6180339888], [0.6180339888], [0.6180339888], [0.6180339888], [0.6180339888], [0.6180339888], [0.6180339888], [0.6180339888], [0.6180339888], [0.6180339888], [0.6180339888]] **(4)**

> *#5 (ii) (a)*

$Orb\left(\left[\frac{5}{2} \cdot x \cdot (1 - x)\right], [x], [2.0], 1000, 1010\right);$

[[Float(-∞)], [Float(-∞)], [Float(-∞)], [Float(-∞)], [Float(-∞)], [Float(-∞)], [Float(-∞)], [Float(-∞)], [Float(-∞)], [Float(-∞)], [Float(-∞)]] **(5)**

> $Orb\left(\left[\frac{5}{2}\cdot x\cdot(1-x)\right], [x], [0.63], 1000, 1010\right);$

[[0.6000000000], [0.6000000000], [0.6000000000], [0.6000000000], [0.6000000000], [0.6000000000], [0.6000000000], [0.6000000000], [0.6000000000], [0.6000000000], [0.6000000000]] **(6)**

> *#5 (iii) (a)*

$Orb\left(\left[\frac{7}{2}\cdot x\cdot(1-x)\right], [x], \left[\frac{6}{7}\right], 1000, 1010\right);$

$$\left[\left[\frac{6}{7}\right], \left[\frac{3}{7}\right], \left[\frac{6}{7}\right], \left[\frac{3}{7}\right], \left[\frac{6}{7}\right], \left[\frac{3}{7}\right], \left[\frac{6}{7}\right], \left[\frac{3}{7}\right], \left[\frac{6}{7}\right], \left[\frac{3}{7}\right], \left[\frac{6}{7}\right]\right]$$ **(7)**

> $Orb\left(\left[\frac{7}{2}\cdot x\cdot(1-x)\right], [x], [0.1], 1000, 1010\right);$

[[0.8269407062], [0.5008842111], [0.8749972637], [0.3828196827], [0.8269407062], [0.5008842111], [0.8749972637], [0.3828196827], [0.8269407062], [0.5008842111], [0.8749972637]] **(8)**

> $Orb\left(\left[\frac{5}{2}\cdot x\cdot(1-x)\right], [x], [0.0], 1000, 1010\right);$

[[0.], [0.], [0.], [0.], [0.], [0.], [0.], [0.], [0.], [0.], [0.]] **(9)**

>