```
> #Nikita John, Attendance 19
> #Maple code for Lecture 19

Help19 := proc( ) :
     print( `SIRSdemo(N,IN,gamma,nu,h,A),e.g. SIRSdemo(100,20,1, 1,0.01, 10); EquPts(F,var),
     StEquPts(F,var) ,  IsStable(M), RandNice(var,K) `) :end:
with(LinearAlgebra) :



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



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




     #StEquPts(F,var): All the Stable equilbrium points of the dynamical system x'(t)=F(x(t))
     where F is the underlying transformation in the set of variables var. For example
#to for the SIRS model with parameters beta,gamma,nu,N, try:
#StEquPts(SIRS(s,i,1,1,0.01,100),[s,i]);
StEquPts := proc(F, var) local d, pt, E, S, J, i, j, J0, i1, Ei0 :
d := nops(var) :

if nops(F) ≠ d then
 RETURN(FAIL) :
fi:

E := EquPts(F, var) :
S := { } :

J := [seq( [seq(diff (F[i], var[ j]), j = 1 ..d) ], i = 1 ..d) ] : #J is the general Jacobian
```

```
for pt in E do
 J0 := evalf(subs({seq(var[i1]=pt[i1], i1=1..d)}, J)) :
 if IsStable(J0) then
  S := S union {pt} :
 fi:
od:

S :
end:
```

```
#EquPts(F,var): All the equilbrium points of the dynamical system x'(t)=F(x(t)) where F is
the underlying transformation in the set of variables var. For example
#to for the SIRS model with parameters beta,gamma,nu,N, try:
#EquPts(SIRS(s,i,beta,gamma,nu,N),[s,i]);
EquPts :=proc(F, var) local sol, i1 :
if nops(F) ≠ nops(var) then
 RETURN(FAIL) :
fi:

sol := {solve({op(F)}, {op(var)})} :

{seq(subs(sol[i1], var), i1=1..nops(sol))} :
end:
```

```
SIRSdemo :=proc(N, IN, gamma, nu, h, A)  local L, beta, i :
 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 i from 1 by 2 to 40 do
beta := i/10*(nu/N) :
print(`beta is`, i/10, `times the threshold value`) :
L := Dis2(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:

#OLD STUFF
Help18 :=proc() : print(` Dis2(F,x,y,pt,h,A), SIRS(s,i,beta,gamma,nu,N) `) :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:


#Dis2(F,x,y,pt,h,A): The approximate orbit of the Dynamical system approximating the 2D for the autonomous  continuous dynamical  process
#dx/dt=F[1](x(t),y(t))
#dy/dt=F[2](x(t),y(t)) , x(0)=pt[1], y(0)=pt[2] with mesh size h from t=0 to t=A
Dis2:=proc(F, x, y, pt, h, A) local L, i :

L := Orb2([x + h*F[1], y + h*F[2]], x, y, pt, 0, trunc(A/h)) :

L := [seq([i*h, [L[i][1], L[i][2]]], i=1..nops(L))] :
end:



#OLD STUFF

Help17:=proc() : print(` HW3g(u,v,w,M), HW2g(u,v,M) `) :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)
HW2g :=proc(u, v, M) local LI, w :
LI := HW3g(u, v, w, M) :
normal(subs(w = 1 - u - v, [LI[1], LI[2]])) :
end:




#OLD STUFF

Help15 :=proc( ) : print(`HW3(u,v,w), HW2(u,v) , Dis1(F,x,x0,h,A), ToSys(k,z,f,INI) `) :end:



#ToSys(k,z,f,INI): 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))
#x2(n)=x1(n-1)
#...

#xk(n)=x[k-1](n-1). It gives the underlying transformation phrased in terms of z[1],...z[k], followed by the initial conditions. Try:
#ToSys:=proc(2,z,z[1]+z[2],[1,1])
ToSys :=proc(k, z, f, INI) local i :
[f, seq(z[i - 1], i = 2 ..k)], INI :
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:


#HW2(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
HW2 :=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)]) :end:





#Dis1(F,x,x0,h,A): The approximate orbit of the Dynamical system approximating the 1D for the autonomous  continuous dynamical  process dy/dt=F(y(t)) , y(0)=y0 with mesh size h from t=0 to t=A
Dis1 :=proc(F, x, x0, h, A) local L, i :
L := Orb(x + h * F, x, x0, 0, trunc(A/h)) :
```

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

##old stuff

#M13.txt: Maple code for Lecture 13 of Dynamical Modesl in Biology, Fall 2021 (taught by Dr. Z.)
Help13 := proc() :
print(` RT2(x,y,d,K), Orb2(F,x,y,pt0,K1,K2), FP2(F,x,y), SFP2(F,x,y), PlotOrb2(L), FP2drz(F,x,y), SFP2drz(F,x,y) `) :end:




#RT2(x,y,d,K): A random rational transformation of degree d from R^2 to R^2 with postiive integer coefficients from 1 to K The inputs are variables x and y and
#the output is a pair of expressions  of (x,y) representing functions. It is for generating examples
#Try:
#RT2(x,y,2,10);
RT2 := proc(x, y, d, K) local ra, i, j, f, g :
ra := rand(1..K) : #random integer from -K to K
f := add(add(ra()*x^i*y^j, j=0..d-i), i=0..d)/add(add(ra()*x^i*y^j, j=0..d-i), i=0..d) :
g := add(add(ra()*x^i*y^j, j=0..d-i), i=0..d)/add(add(ra()*x^i*y^j, j=0..d-i), i=0..d) :
[f, g] :
end:



#Orb2(F,x,y,pt,K1,K2): Inputs a mapping F=[f,g] from R^2  to R^2 where f and g describe functions of x and y, an initial point pt0=[x0,y0]
#outputs the orbit starting at discrete time K1 and ending in discrete time K2. Try
#F:=RT2(x,y,2,10);
#Orb2(F,x,y,[1.1,1.2],1000,1010);
Orb2 := proc(F, x, y, pt0, K1, K2) local pt, L, i :
pt := pt0 :

for i from 1 to K1-1 do
pt := subs({x=pt[1], y=pt[2]}, F) :
od:

L := [] :
for i from K1 to K2 do
L := [op(L), pt] :
pt := normal(subs({x=pt[1], y=pt[2]}, F)) :

od:
L :
```

```
end:

#FP2(F,x,y): The list of fixed points of the transformation [x,y]->F. Try
#FP2([x-y,x=y],x,y);
FP2 :=proc(F, x, y) local L, i :
L := [solve({F[1]=x, F[2]=y}, {x, y})] :

[seq(subs(L[i], [x, y]), i=1 ..nops(L))] :
end:


#SFP2(F,x,y): The list of Stable fixed points of the transformation [x,y]->F. Try
#SFP2([(1+x)/(1+y), (1+7*y)/(4+x)],x,y);
SFP2 :=proc(F, x, y) local L, J, S, J0, i, pt, EV :

L := evalf(FP2(F, x, y)) :
    #F is the list of ALL fixed points of the transformation [x,y]->F using the previous procedure
    FP2(F,x,y), but since we are interested in numbers we take the floating point version using
    evalf

J := Matrix(normal([[diff(F[1], x), diff(F[1], y)], [diff(F[2], x), diff(F[2], y)]])) :
    #J is the Jacobian matrix in general (in terms of the variables x and y). Note that J is a
    SYMBOLIC matrix featuring variables x and y

S := [ ] :  #S is the list of stable fixed points that starts out empty

for i from 1 to nops(L) do  #we examime it case by case
pt := L[i] : #pt is the current fixed point to be examined

J0 := subs({x=pt[1], y=pt[2]}, J) :
    #J0 is the NUMERICAL matrix obtained by plugging-in the examined fixed pt

EV := Eigenvalues(J0) :
    # We used Maple's command Eigenavalues to find the eigenvalues of this 2 by 2 matrix

if abs(EV[1]) < 1 and abs(EV[2]) < 1 then
 S := [op(S), pt] :
    #If both eigenvalues have absolute value less than 1 it means that they are stable, so we
    append the examined fixed point, pt, to the list of fixed points
fi:

od:
S : #the output is S
end:



###added Oct. 17, 20221
with(plots) :
```

```
PlotOrb1 :=proc(L) local i, d :

d := textplot([L[1], 0, 0]) :

for i from 2 to nops(L) do
d := d, textplot([L[i], 0, i-1]) :
od:
display(d) :
end:


PlotOrb2 :=proc(L) local i, d :

d := textplot([op(L[1]), 0]) :

for i from 2 to nops(L) do
d := d, textplot([op(L[i]), i-1]) :
od:
display(d) :
end:
###End added Oct. 17, 20221


###old stuff
#M11.txt: Maple code for Lecture 11 of Dynamical Models in Biology taught by Dr. Z.
Help11 :=proc( ) : print(`SFPe(f,x), Orbk(k,z,f,INI,K1,K2) `) :end:


    #SFPe(f,x): The set of fixed points of x->f(x) done exactly (and allowing symbolic
    parameters), followed by the condition of stability (if it is netween -1 and 1 it is stable)
#Try: FPe(k*x*(1-x),x);
#VERSION OF Oct. 12, 2021 (avoiding division by 0)
SFPe :=proc(f, x) local f1, L, i, M:
f1 := normal(diff(f, x)) :
L := [solve(numer(f-x), x)] :
M := [ ] :

for i from 1 to nops(L) do
 if subs(x=L[i], denom(f1) ) ≠ 0 then
  M := [op(M), [L[i], normal(subs(x=L[i], f1))]] :
 fi:
od:
M:

end:


#Added after class
```

```
#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:


####STAFT FROM M9.txt
#M9.txt: Maple Code for "Dynamical models in Biology" (Math 336) taught by Dr. Z., Lecture 9

Help9 := proc() :
  print(` Orb(f,x,x0,K1,K2), Orb2D(f,x,x0,K) , FP(f,x) , SFP(f,x) , Comp(f,x)  `) :end:


#Orb(f,x,x0,K1,K2): Inputs an expression f in x (desccribing) a function of x, an initial point,
x0, and a positive integer K, outputs
#the values of x[n] from n=K1 to n=K2. Try: where x[n]=f(x[n-1]), . Try:
#Orb(2*x*(1-x),x,0.4,1000,2000);
```

```
Orb := proc(f, x, x0, K1, K2) local x1, i, L :
x1 := x0 :

for i from 1 to K1 do
 x1 := subs(x = x1, f) :
    #we don't record the first values of K1, since we are interested in the long-time behavior of
    the orbit
od:

L := [x1] :

for i from K1 to K2 do
 x1 := subs(x = x1, f) :   #we compute the next member of the orbit
 L := [op(L), x1] : #we append it to the list
od:

L : #that's the output

end:

#Orb2D(f,x,x0,K): 2D version of Orb(f,x,x0,0,K), just for illustration
Orb2D := proc(f, x, x0, K) local L, L1, i :
L := Orb(f, x, x0, 0, K) :
L1 := [[L[1], 0], [L[1], L[2]], [L[2], L[2]] ] :
for i from 3 to nops(L) do
 L1 := [op(L1), [L[i-1], L[i]], [L[i], L[i]]] :
od:
L1 :
end:


#FP(f,x): The list of fixed points of the map x->f where f is an expression in x. Try:
#FP(2*x*(1-x),x);
FP := proc(f, x)
evalf([solve(f = x, x)]) :
end:


#SFP(f,x): The list of stable fixed points of the map x->f where f is an expression in x. Try:
#SFP(2*x*(1-x),x);
SFP := proc(f, x) local L, i, f1, pt, Ls :
L := FP(f, x) : #The list of fixed points (including complex ones)

Ls := [ ] :      #Ls is the list of stable fixed points, that starts out as the empty list

f1 := diff(f, x) :  #The derivative of the function f w.r.t. x

for i from 1 to nops(L) do
pt := L[i] :
```

```
if abs(subs(x=pt,f1)) < 1 then

 Ls := [op(Ls), pt] :  # if pt,  is stable we add it to the list of stable points

fi:

od:

Ls :   #The last line is the output

end:

#Comp(f,x): f(f(x))
Comp :=proc(f, x) : normal(subs(x=f,f) ) :end:




##added Oct. 17, 2021
#FP2drz(F,x,y): The list of fixed points of the transformation [x,y]->F. Dr. Z.'s way
#FP2([x-y,x+y],x,y);
FP2drz :=proc(F, x, y) local eq, i, L, S1 :
eq := [numer(F[1]-x), numer(F[2]-y)] :

L := Groebner[Basis](eq, plex(x, y)) :

S1 := evalf([solve(L[1], y)]) :
[seq([solve(subs(y=S1[i], L[2]), x), S1[i]], i=1 ..nops(S1))] :
end:


#SFP2drz(F,x,y): The list of Stable fixed points of the transformation [x,y]->F. Try
#SFP2drz([(1+x)/(1+y), (1+7*y)/(4+x)],x,y);
SFP2drz :=proc(F, x, y) local L, J, S, J0, i, pt, EV :

L := FP2drz(F, x, y) :
    #F is the list of ALL fixed points of the transformation [x,y]->F using the previous procedure
    FP2(F,x,y), but since we are interested in numbers we take the floating point version using
    evalf

J := Matrix(normal([[diff(F[1], x), diff(F[2], x)], [diff(F[1], y), diff(F[2], y)]])) :
    #J is the Jacobian matrix in general (in terms of the variables x and y). Note that J is a
    SYMBOLIC matrix featuring variables x and y

S := [ ] :  #S is the list of stable fixed points that starts out empty

for i from 1 to nops(L) do   #we examime it case by case
pt := L[i] : #pt is the current fixed point to be examined

J0 := subs({x=pt[1], y=pt[2]}, J) :
```

```
    #J0 is the NUMERICAL matrix obtained by plugging-in the examined fixed pt

  EV := Eigenvalues(J0) :
    # We used Maple's command Eigenavalues to find the eigenvalues of this 2 by 2 matrix

  if abs(EV[1]) < 1 and abs(EV[2]) < 1  then
   S := [op(S), pt] :
    #If both eigenvalues have absolute value less than 1 it means that they are stable, so we
    append the examined fixed point, pt, to the list of fixed points
  fi:

  od:
  S : #the output is S
  end
SFP2drz := proc(F, x, y)                                                                  (1)
   local L, J, S, J0, i, pt, EV;
   L := FP2drz(F, x, y);
   J := Matrix(normal([[diff(F[1], x), diff(F[2], x)], [diff(F[1], y), diff(F[2], y)]]));
   S := [ ];
   for i to nops(L) do
      pt := L[i];
      J0 := subs({x=pt[1], y=pt[2]}, J);
      EV := LinearAlgebra:-Eigenvalues(J0);
      if abs(EV[1]) < 1 and abs(EV[2]) < 1 then S := [op(S), pt] end if
   end do;
   S
end proc
> F := [(1 - 6·x - y)·(3 - x - y), (3 - 8·x - 3·y)·(1 - 4·x - 6·y)] :
  EquPts(F, [x, y]);
  evalf(StEquPts(F, [x, y]));
```

$$\left\{[0, 1], \left[-\frac{6}{5}, \frac{21}{5}\right], \left[\frac{5}{32}, \frac{1}{16}\right], \left[\frac{17}{2}, -\frac{11}{2}\right]\right\}$$

$$\{[0.1562500000, 0.06250000000]\} \quad (2)$$

```
>
```