```
In[*]:= BeginPackage["EpidCRN`"];
Global`ome;Global`u;(*Global`v;*)
ACM::usage = "A2=ACM[A,k] yields additive compound matrix";asoRea::usage = "transforms cl
association";
Bifp::usage = "Bifp[mod_,cN_,indX_,bifv_,pl0_:0,pL_:10,y0_:-1, yM_:10,cR0_:0]
 gives the bifurcation plot of the dynamics wrt to one par ";
 red::usage = "recl=red[re,cond] erases from the output of a Reduce all the
conditions in cond";
reCL::usage = "recl=red[re,cond] erases from the output of a Reduce all the
conditions in cond";minSiph;
 CofP::usage = "co=CofP[list] yields coefficients of a
polynomial as required by Routh-Hurwicz theory, ie
normalized so the free coefficient is 1
(see for example Röst, Tekeli, Thm 4A)";
expM::usage = "expM[var,expo] gives the vector var at power in matrix expo";
CofRH::usage = "co=CofRH[mat] yields coefficients of
CharacteristicPolynomial, as required by Routh-Hurwicz theory, ie
normalized so the free coefficient is 1
(see for example Röst, Tekeli, Thm 4A):
Drop[Reverse[CoefficientList[(-1)^(Length@A)
CharacteristicPolynomial[A,x],x]],1]";
cons::usage = "con=cons[mat,cp_:{}] parametrizes positively
the  left kernel of mat, using also conditions cp;cp is not necessary
if mat is numeric*)";
seZF::usage = "seZF[so_] removes in  a list of lists those
with a 0";
onePR::usage = "onePR[cof_,cp_:{}] outputs conditions that the first and
last coefs of a list have different signs";
DFE::usage = "DFE[mod_,inf_] yields the DFE of the model";
expon::usage= "Eponent[p,Variable[p]] computes the maximum power
 of an expanded form p";
FHJ::usage="FHJ[comp_List,edges_List,rates_List, ver_:{},groups_List:{}]
generates the Feinberg-Horn-Jackson graph. The first argument, comp_List,
 represents the set of complexes,  edges_List defines the reaction edges, rates_List
 specifies the reaction rates. The optional
argument ver_ determines the node sizes, and groups_List, used for distinguishing linkage
 colors the first specified class in green, the second in red, ...,following a given list
colors. The complement of the groups specified is collored in yellow.";
fix::usage = "fix[mod_,cn_:{}]";
phasePl2::usage = "phasePl2[mod_,plc_:{},cn_:{}] plots a 2dim phase-plot
of  mod, for the components not excluded in plc";
H4::usage = "H4[co] gives the 4'th Hurwitz det,  needed in
Routh-Hurwitz theory (see for example Röst, Tekeli, Thm 4A).
H4[CofRH[M]] gives the 4'th Hurwitz det of the
matrix M, and could be used in Hur4M[mat]"; H6;
Hur2::usage = "ine=Hur2[co] yields stability cond";
```

```
Hur3M::usage = "{co,h3,ine}=Hur3M[A] yields ine=Append[inec,h3>0]";
Hur4M::usage = "{co,h4,ine}=Hur4M[A]";
Hur5M::usage="{co,h5,ine,H5}=Hur5M[jac]";
posM::usage="keep all syntactically positive terms";
FposEx::usage="extracts first syntactically positive term in a nonnegative matrix";
perR::usage="perR[M_, i_, j_]=ReplacePart[M, {i -> M[[j]], j -> M[[i]]}]";
perC::usage="perC[matrix_, cycle_List] performs a cyclic permutation
 on the rows (or columns) of the input matrix based on the indices
 in the list cycle_List. The rows (or columns) specified by cycle are rearranged
according to the right rotation of cycle, and the modified matrix is returned";
Idx;IaFHJ::usage = "{oU,taF}=IaFHJ[vert,edg]";
IkFHJ::usage = "Ik=IkHKF[vert,edg,tk]";
sym2Str;str2Sym;rul2Str;
toSum;toProd;
strEdg;
SpeComInc::usage = "SpeComInc[comp,spec]";
expM;
makeLPM::usage = "makeLPM[mat_] :=
Table[Det@mat[[1 ;; i, 1 ;; i]], {i, 1, Length@mat}] yields
the leading principal minors";countMS;
onlyP::usage ="onlyP[m_] checks whether all the coefficients
 of the numerator of a rational expression m  are nonnegative";
verHir::usage ="verHir[RHS,var,intRows] checks whether a network can be reduced
using the species indexed by intRows";
mat2Matl;

JTD::usage = "JTD[mod,cn_:{}]";
JTDP::usage = "JTDP[mod,ζ_:ζ,cn_:{}]";
NGM::usage = "NGM[mod_,inf_] yields {Jy,V1,F1,F,-V,K,chp(u)};they
 are the infectious Jacobian, two intermediate results, the new
infections, transitions, and next generation matrices,
and its char. pol.";
NGMs::usage = " simpler version of NGM[mod_,inf_], treats incorrectly denominators and ex
JR0::usage = "JR0[pol],{R0,co}";
extP::usage ="extP[mod_,inf_] yields the Bacaer equation
for approximate extinction probability";
Par::usage = "Par[dyn,var]";
Res1F;Deg;
RUR::usage = "RUR[mod,ind,cn_:{}] attempts to reduce the fixed point system to
one with variables specified by the list ind; only singleton ind is allowed currently;
outputs are ratsub,pol,and ln=pol//Length";
GBH::usage = "GBH[pol_,var_,sc_,cn_:{}]";
matl2Mat;matlr2Mat;l2L;
m2toM;Stodola;DerL;

mSim::usage = "mSim[mod,cN, cInit,T,excluded]";Bifp;
convNum;Hirono;
```

```
Sta::usage = "numeric";
Stab::usage = "Stab[mod_,cfp_,cn_:{}]";

L1Planar::usage = "L1Planar[fg,eq:{}]";
DerEq::usage = "DerEq[fg,eq:{}]; eq is condition";
GetVec::usage = "GetVec[A,om],used in L13,L23";
L13;L23;(*DerSc::usage ="DerSc[f]";
RescaleODE::usage ="RescaleODE[f,equilcon];necessary
before calling DerSc ";*)

Begin["`Private`"];
ACM[matrix_,k_]:=
D[Minors[IdentityMatrix[Length@matrix]+t*matrix,k],t]/. t→0;
(*A={{a11,a12,a13},{a21,a22,a23},{a31,a32,a33}};;
ACM[A,2]//MatrixForm
*)
mat2Matl[matrix_List]:=Module[{matStr},
(*Convert each row to a space-separated string and join rows with semicolons*)matStr=Stri
(*Surround the string with MATLAB matrix brackets*)
StringJoin["[",matStr,"]"]];
perR[M_, i_, j_] := ReplacePart[M, {i → M⟦j⟧, j → M⟦i⟧}];
perC[matrix_,cycle_List]:=
Module[{tempMatrix=matrix},tempMatrix⟦cycle⟧=tempMatrix⟦RotateRight[cycle]⟧;
tempMatrix];
expon:=Exponent[#,Variables[#]]&;
expM=Inner[OperatorApplied[Power],#2,#1,Times]&;
Par[RHS_,X_]:=Complement[Variables[RHS],X];
m2toM[a_List]:=
ReplaceAll[str_String:→Total[StringSplit[str,"+"]]][Rule@@@StringSplit[First@(List@@@a),
SpeComInc[comp_,spec_]:=Coefficient[#,spec]&/@comp;
(*creates the species-complex incidence matrix*)
countMS[m_]:=m//Together//(*put polynomials in standard form*)
NumeratorDenominator//(*get polys*)Map@CoefficientArrays//
(*get coefficients of polys*)
ReplaceAll[sa_SparseArray:→sa["NonzeroValues"]]//
(*get nonzero coeffs*)Flatten//(*preconditioner for AllTrue*)
Count[#, _?Negative] &;

onlyP[m_]:=m//Together//(*put polynomials in standard form*)
NumeratorDenominator//(*get polys*)Map@CoefficientArrays//
(*get coefficients of polys*)
ReplaceAll[sa_SparseArray:→sa["NonzeroValues"]]//
(*get nonzero coeffs*)Flatten//(*preconditioner for AllTrue*)
AllTrue[#,NonNegative]& (*has 0 if constant term is zero*);

CofRH[A_?MatrixQ]:=Module[{x},
Drop[Reverse[CoefficientList[(-1)^(Length@A)×
```

```
CharacteristicPolynomial[A,x],x]],1]];

CofP[co_?ListQ]:=Drop[Reverse[(-1)^(Length@co) *co],1];
red[re_,cond_:{}]:=re/. (♯ → True & /@ cond);
reCL[re_] :=DeleteCases[re, _Symbol > 0 |
Subscript[_, __] > 0, Infinity];

seZF[expr_] := Select[expr, FreeQ[♯, 0] &];

onePR[cof_,cp_:{}]:=Append[cp,(cof⟦♯⟧//First) ×
(cof⟦♯⟧//Last)<0]&/@Range[cof//Length];
makeLPM[mat_] :=
Table[Det@mat⟦1 ;; i, 1 ;; i⟧, {i, 1, Length@mat}];
Hur2[co_]:=Module[{co3, ine}, co3=co⟦3⟧;
ine= {co⟦1⟧ co3>0,co⟦2⟧ co3>0};ine];

cons[mat_,cp_:{}] := Module[{X, sol, dim, cv},
(*Parametrize the kernel to the left , using only pos
pars*)
X = Array[x, Length[mat]];
  sol = SolveValues
  [Join[Thread[X . mat == 0],cp], X, NonNegativeIntegers];
(*particularize  the Mathematica constants C[i] determining
a point in the conservations cone, by choosing exactly one
parameter to be one, and the rest to be 0*)
  dim = NullSpace[mat // Transpose] // Length;
  cv = Table[C[i], {i, dim}];
  Flatten /@
   Table[sol /. Thread[cv → IdentityMatrix[dim]⟦i⟧], {i, dim}]];

matl2Mat[matrix_String]:=Module[{formattedMatrix},
(*Step 1:Split the input by newlines to separate rows*)
formattedMatrix=StringSplit[matrix,"\n"];
(*Step 2:Replace multiple spaces with a single space*)
formattedMatrix=StringReplace[formattedMatrix,Whitespace..→" "];
(*Step 3:Replace spaces with commas*)
formattedMatrix=StringReplace[♯," "→", "]&/@formattedMatrix;
(*Step 4:Add curly braces around each row*)
formattedMatrix="{"<>♯<>"}"&/@formattedMatrix;
(*Step 5:Join rows with commas and wrap everything in curly braces*)
formattedMatrix="{"<>StringRiffle[formattedMatrix,",\n"]<>"}";
(*Convert the string into an actual Mathematica expression*)
ToExpression[formattedMatrix]];
matlr2Mat[str_String]:=Module[{formattedString,result},(*Step 1:Remove curly braces and
(*Step 2:Split the string by spaces to separate the elements*)formattedString=StringSplit
(*Step 3:Convert each element to an integer*)result=ToExpression[formattedString];
(*Step 4:Remove any Null values*)DeleteCases[result,Null]];
```

```
(*Helper function to switch reaction network representation from classic to list of assoc
Map[Function[r,Association["Substrates"→parseSide[r⟦1⟧],"Products"→parseSide[r⟦2⟧]]],RN

(*Corrected minimal siphon finder*)
minSiph[species_List,reactions_List]:=Module[{ns,sm,specs,constraints,solutions,siphons,r
sm=AssociationThread[species→Range[ns]];
specs=Array[Symbol["s"<>ToString[#]]&,ns];
(*Build constraints*)constraints={Or@@specs};(*At least one species in siphon*)Do[Module
products=reaction["Products"];
(*Convert species names to indices*)subIdx=If[substrates==={}||substrates==={""},{},Sel
prodIdx=If[products==={}||products==={""},{},Select[Lookup[sm,products,Nothing],Integer
(*Add constraints for each product*)Do[If[Length[subIdx]==0,(*Empty product:product cann
Print["Constraints generated: ",Length[constraints]];
Print["Sample constraints: ",Take[constraints,Min[5,Length[constraints]]]];
(*Find solutions with moderate limit to avoid crashes*)solutions=FindInstance[constraint
If[solutions==={},Return[{}]];
siphons=Map[Flatten@Position[specs/. #,True]&,solutions];
siphons=DeleteDuplicates[siphons];
Print["All found siphons: ",siphons];
(*Proper minimality check:remove any siphon that contains another*)minimal={};
Do[If[Not[AnyTrue[siphons,Function[other,other=!=siphon&&SubsetQ[siphon,other]]]],Append
Print["After minimality filter: ",minimal];
minimal]
```

```
In[ ]:= (*minSiph[species_List,reactions_List]:=Module[{ns,sm,specs,constraints,siphons,status,m
(*map each species name to its index*)
sm=AssociationThread[species→Range[ns]];
(*Boolean variables s1…s_ns;s_i==True means species i is in the siphon*)
specs=Array[Symbol["s"<>ToString[#]]&,ns];
(*initial constraint:at least one species must be in the siphon*)constraints={Or@@specs}
(*for each reaction,add the Julia-style constraint*)Do[subIdx=Lookup[sm,reaction["Substr
prodIdx=Lookup[sm,reaction["Products"],{}];
Do[If[subIdx==={},(*if ∅→something,that product cannot be in the siphon*)AppendTo[constr
siphons={};
(*find the first satisfying assignment*)status=FindInstance[constraints,specs,1,Method→"
(*iterate until UNSAT,each time banishing any superset of the found siphon*)While[status=
siphon=Flatten@Position[specs/. model,True];
AppendTo[siphons,siphon];
(*Add constraint:at least one of those True variables must now be False to forbid superse
AppendTo[constraints,Or@@(Not/@specs⟦siphon⟧)];
status=FindInstance[constraints,specs,1,Method→"Boolean"];];
(*remove any siphon that strictly contains another*)DeleteCases[siphons,s_/;AnyTrue[siph
(*test species={"A","B","C"};
reactions={<|"Substrates"→{},"Products"→{"A"}|>,<|"Substrates"→{"A","B"},"Products"→{"C"
 minSiph[species,reactions]*)
```

```
*)
Bifp[mod_,cN_,indX_,bifv_,pl0_:0,pL_:10,y0_:-1, yM_:10,cR0_:0]:=
Module[{dyn, X,fp,pl,epi,plf},dyn=mod⟦1⟧/.cN;X=mod⟦2⟧;
fp=Quiet[Solve[Thread[(dyn)==0],X]//N];
epi={Text["\!\(\*SubscriptBox[\(c\), \(R0\)]\)",Offset[{10,10},{ cR0,0}]],
{PointSize[Large],Style[Point[{ cR0,0}],Purple]}};
pl=Plot[Evaluate@(X⟦indX⟧/.fp),{bifv,pl0,pL},
PlotStyle→{Blue,Green,Red,Brown}];
plf=Show[{pl},Epilog→epi,PlotRange→{{pl0,pL},{y0,yM}},AxesLabel→{bifv,"Fixed points"}];
{fp,plf}];

Idx[set_,n_PositiveInteger]:=Module[{seq},
seq=(Table[Count[set,i],{i,n}]/.List→Sequence);seq];
FHJ[comp_List,edges_List,rates_List, ver_:{},groups_List:{}]:=
Module[{colorList,shapeList,vertexColors,options,vertexShapes,defaultColor=Yellow},
colorList={Green,Red,Yellow,Purple,Orange};
shapeList={"Square","Circle","ConcaveHexagon","Triangle","Hexagon","Pentagon","Star"};
vertexColors=Join[Flatten[MapIndexed[Thread[#1→colorList⟦#2⟦1⟧⟧]&,groups]],
#→defaultColor&/@Complement[comp,Flatten[groups]]];
vertexShapes=Flatten[MapIndexed[Thread[#1→shapeList⟦#2⟦1⟧⟧]&,groups]];
options={VertexShapeFunction→vertexShapes,VertexStyle→vertexColors,VertexSize→ver,
VertexLabels→{_ → Placed[Automatic, Center]},EdgeStyle → {{Black, Thick}},
PerformanceGoal→"Quality",
EdgeLabels→Thread[edges→rates],EdgeLabelStyle→Directive[Black,Bold,Background→White]};
LayeredGraphPlot[edges,Right,options]];

IaFHJ[vert_,edg_]:=Module[{gg,oU,taF},gg[a_,b_]:=
Which[a===b⟦1⟧,-1,a===b⟦2⟧,1,True,0];
oU=Outer[gg,vert,edg];
taF=TableForm[oU,TableHeadings→{vert,edg},
TableAlignments→{Right,Top}];
{oU,taF}
];
IkFHJ[vert_,edg_,tk_]:=Module[{tri,gg,oU},
tri=MapThread[Append,{edg,tk}];gg[a_,b_]:=
Which[a===b⟦1⟧,b⟦3⟧,a===b⟦2⟧,0,True,0];
oU=Outer[gg,vert,tri]//Transpose
];

convNum[vertices_List] := Module[
  {basis, processTerm, parseVertex},

  (* Define basis vectors for A and B *)
  basis = Association[{"A" → {1, 0}, "B" → {0, 1}}];

  (* Function to process each term and convert to vector *)
  processTerm[term_] := Module[{coef, letter},
```

```
    (* Extract coefficient and letter, default coefficient is 1 if missing *)
    {coef, letter} =
     StringCases[term, {a : DigitCharacter .. ~~ " " ~~ l : ("A" | "B") ⧴ {ToExpression[
                       l : ("A" | "B") ⧴ {1, l}}][[1]];
    coef * basis[letter]
   ];

   (* Parse each vertex string into terms and sum the resulting vectors *)
   parseVertex[vertex_String] :=
    Total[processTerm /@ StringSplit[vertex, " + "]];

   (* Apply the conversion to the entire list of vertices *)
   parseVertex /@ vertices
  ]
 sym2Str=Replace[Thread[#1→#2],x_Symbol⧴ToString[x],All]&;
 str2Sym= #//. s_String ⧴ToExpression[s]&;
 varLS= (#//. s_String ⧴ToExpression[s])//Variables&;

 rul2Str=# /. r_Rule ⧴ ToString /@ r &;

 (* Example usage
 expr={{2 "x"+"y"→4 "x"+5 "y"},{5 "x"→7 "y"}};
 convEdg[expr]
 vert = {"B", "A", "2 A + B", "A + 2 B"};
 vertn = convNum[vert]
 edg = {"B" → "A", "2 A + B" → "A + 2 B"};
 wei = {k1, k2};

 FHJn[vert, edg, wei, {{"A", "B"}},  .20]*)
  Hur3M[A_]:=Module[{co,h3,inec,ineSys,ω},
 co=CoefficientList[(-1)^Length[A] CharacteristicPolynomial[A,ω],ω];
 h3=co⟦ 2⟧* co⟦ 3⟧-co⟦ 1⟧ *co⟦ 4⟧;inec={co⟦ 1⟧>0,co⟦ 2⟧>0};
 ineSys=Append[inec,h3>0];
 {co,h3,ineSys}];
 (*A={{-j[1]-j[3]+j[4],j[4],j[3]},{-2 j[4],-j[2]-j[4],0},{j[3],0,-j[3]}};
 Hur3M[A]*)

 Hur4M[mat_]:=Module[{lm,ch,cot,co,H4,h4,ine},
 lm=mat//Length;
 ch=((-1)^lm * CharacteristicPolynomial[mat,λ]//Factor);
 cot=CoefficientList[ch,λ];
 co=Reverse[Drop[cot,-1]];(*co⟦0⟧=1 is lead coef*)
 H4={{co⟦1⟧,1,0,0},
 {co⟦3⟧,co⟦2⟧,co⟦1⟧,1},
 {0,co⟦4⟧,co⟦3⟧,co⟦2⟧},
 {0,0,0,co⟦4⟧}};h4=Det[H4];
 ine=Thread[co>0];{co,h4,ine}];
```

```
H4[co_]:={{co⟦1⟧,1,0,0},
{co⟦3⟧,co⟦2⟧,co⟦1⟧,1},
{0,co⟦4⟧,co⟦3⟧,co⟦2⟧},
{0,0,0,co⟦4⟧}};

Hur5M[jac_]:=Module[{lm,ch,cot,co,H5,h5,ine},
lm=jac//Length;
ch=((-1)^lm * CharacteristicPolynomial[jac,λ]//Factor);
cot=CoefficientList[ch,λ];
co=Reverse[Drop[cot,-1]];
H5={{co⟦1⟧,1,0,0,0},{co⟦3⟧,co⟦2⟧,co⟦1⟧,1,0},
{co⟦5⟧,co⟦4⟧,co⟦3⟧,co⟦2⟧,co⟦1⟧},
{0,0,co⟦5⟧,co⟦4⟧,co⟦3⟧},
{0,0,0,0,co⟦5⟧}};h5=Det[H5];
ine=Append[Thread[co>0],co⟦1⟧×co⟦2⟧>co⟦3⟧];{co,h5,ine,H5}];
(*
H5[co_]:=Module[{hm},hm={{co⟦1⟧,1,0,0,0},
{co⟦3⟧,co⟦2⟧,co⟦1⟧,1,0},
{co⟦5⟧,co⟦4⟧,co⟦3⟧,co⟦2⟧,co⟦1⟧},
{0,0,co⟦5⟧,co⟦4⟧,co⟦3⟧},
{0,0,0,0,co⟦5⟧}}];*)

H6[co_]:=Module[{hm},hm={{co⟦1⟧,1,0,0,0,0},
{co⟦3⟧,co⟦2⟧,co⟦1⟧,1,0,0},
{co⟦5⟧,co⟦4⟧,co⟦3⟧,co⟦2⟧,co⟦1⟧,1},
{0,co⟦6⟧,co⟦5⟧,co⟦4⟧,co⟦3⟧,co⟦2⟧},
{0,0,0,co⟦6⟧,co⟦5⟧,co⟦4⟧},
{0,0,0,0,0,co⟦6⟧}}];

JTD[mod_,cn_:{}]:=
Module[{dyn,X,jac,tr,det},dyn=mod⟦1⟧;X=mod⟦2⟧;
jac=Grad[dyn,X]/.cn;
tr=Tr[jac];
det=Det[jac];
{jac,tr,det}];
JTDP[mod_,ζ_:ζ,cn_:{}]:=
Module[{dyn,X,jac,tr,det,chp,cof},dyn=mod⟦1⟧;X=mod⟦2⟧;
jac=Grad[dyn,X]/.cn;
tr=Tr[jac];
det=Det[jac];
chp=CharacteristicPolynomial[jac,ζ];cof=CoefficientList[chp,ζ];
{jac,tr,det,cof,chp}];
(*Collect[JTDP[SIRG,x]⟦4⟧,x]
JTDP[SIRG]⟦1⟧//MatrixForm*)
Res1F[mod_,csr_,pol_,in_,cn_:{}]:=
Module[{jac,det,res,chp,cof},
```

```
jac=JTDP[mod]⟦1⟧/.csr/.cn;
det=Numerator[Together[Det[jac]]];
res=Resultant[det,pol,in]//Factor
];

DFE[mod_,inf_:{},cn_:{}]:=Module[{dyn,X},
    dyn=mod⟦1⟧/.cn;X=mod⟦2⟧;
    Quiet[Solve[Thread[dyn⩵0]/.Thread[X⟦inf⟧→0],X]]];

fix[mod_,cn_:{}]:=Module[{dyn,X,fp,Xp},(*mostly numerical*)
   dyn=mod⟦1⟧//.cn;X=mod⟦2⟧;
   fp=X/.Quiet[Solve[Thread[(dyn)⩵0],X]];
   If[cn≠{},Xp=Cases[_?(AllTrue[NonNegative]@#&)]@fp;
   fp=SortBy[Xp,{ #⟦1⟧&,#⟦2⟧&}]];fp];

phasePl2[mod_,cn_:{},plc_:{},in_:1]:=Module[{dyn,X,pl,fp,jac,jacE,Xp,Xs,sp,Gp,cP,xM,yM,
   r1,r2},
   dyn=mod⟦1⟧//.cn;X=mod⟦2⟧;pl=Complement[Range[Length[X]],plc];
   fp=X/.Quiet[NSolve[Thread[(dyn)⩵0],X]](*works if fp takes a short time*);
   jac=Grad[dyn,X]; jacE=jac/.{Thread[X→fp⟦1⟧]}];
   Xp=Cases[_?(AllTrue[NonNegative]@#&)]@fp(*selects positive fp*);
   xM=Max/@Transpose[Xp](*determines the maximum values of x and y for plotting*);
   Xs=SortBy[Xp,{ #⟦1⟧&,#⟦2⟧&}](*sorts fixed points in ascending order of x and y*);
   r1={X⟦pl⟦1⟧⟧,-xM⟦pl⟦1⟧⟧-.5,xM⟦pl⟦1⟧⟧+.5};
   r2={X⟦pl⟦2⟧⟧,-xM⟦pl⟦2⟧⟧-.5,xM⟦pl⟦2⟧⟧+.5};
   sp=StreamPlot[{dyn⟦pl⟦1⟧⟧,dyn⟦pl⟦2⟧⟧},r1,r2,StreamStyle→Arrowheads[Medium],
   ColorFunction→"Rainbow",StreamPoints→Fine,
   Frame→True,
   FrameLabel→{"x[t]","y[t]"},
   PlotLabel→Style["Phase portrait",Large],LabelStyle→18];
   Gp=Graphics[{PointSize[0.03],{Red,Black,Cyan},Point[Xp]}];
   cP=ContourPlot[{dyn⟦1⟧,dyn⟦2⟧},r1,r2,
   FrameLabel→{"x[t]","y[t]"},ContourStyle→{Blue,Red},
   LabelStyle→Directive[Black,Medium]];
   {Xs, jacE,Show[sp,cP,Gp]}
   ]

   posM= Replace[#,{_?Negative→0,e_⧴Replace[Expand[e],
 {Times[_?Negative,_]→0,t_Plus⧴Replace[t,_?Negative|Times[_?Negative,_]→0,1]}]},{2}]&;

 FposEx=With[{pos=First@SparseArray[#]["NonzeroPositions"]},SparseArray[{pos→Extract[#,
 Dimensions@#]]&;

NGMs[mod_,inf_:{}]:=Module[{dyn,X,infc,M,V,F,F1,V1,K,chp},
   dyn=mod⟦1⟧;X=mod⟦2⟧;
   infc=Complement[Range[Length[X]],inf];
   Jy=Grad[dyn⟦inf⟧,X⟦inf⟧];
```

```
    chp=CharacteristicPolynomial[Jy,u];
    (*The jacobian of the infectious equations*)
    V1=-Jy/.Thread[X⟦infc⟧→0];
    (*V1 is a first guess for V, retains all gradient terms which
    disappear when the non infectious components are null*)
    F1=Jy+V1/.Thread[X⟦inf⟧→0];
    (*F1 is a first guess for F, containing all other
    gradient terms*)
    F=ReplaceAll[F1, _. _?Negative → 0];
    (*all terms in F1 containing minuses are set to 0*);
    V=F-Jy;
    K=(F . Inverse[V])/.Thread[X⟦inf⟧→0]//FullSimplify;
    Kd=( Inverse[V] . F)/.Thread[X⟦inf⟧→0]//FullSimplify;
  {Jy,V1,F1,F,V,K,Kd,chp}]

  NGM[mod_,inf_:{}]:=Module[{dyn,X,infc,M,V,F,F1,V1,K,chp},
    dyn=mod⟦1⟧;X=mod⟦2⟧;
    infc=Complement[Range[Length[X]],inf];
    Jy=Grad[dyn⟦inf⟧,X⟦inf⟧];
    chp=CharacteristicPolynomial[Jy,u];
    (*The jacobian of the infectious equations*)
    V1=-Jy/.Thread[X⟦infc⟧→0];
    (*V1 is a first guess for V, retains all gradient terms which
    disappear when the non infectious components are null*)
    F1=Jy+V1/.Thread[X⟦inf⟧→0];
    (*F1 is a first guess for F, containing all other
    gradient terms*)
    F=posM[F1];
    (*all terms in F1 containing minuses are set to 0*);
    V=F-Jy;
    K=(F . Inverse[V])/.Thread[X⟦inf⟧→0]//FullSimplify;
    Kd=( Inverse[V] . F)/.Thread[X⟦inf⟧→0]//FullSimplify;
  {Jy,V1,F1,F,V,K,Kd,chp}]


  (*K=NGM[SEIR,Range[2]]⟦4⟧;eig=Eigenvalues[K]/.Thread[X⟦inf⟧→0];*)


JR0[pol_,u_]:=Module[{co,co1,cop,con,R0J},
co=CoefficientList[pol,u];
  Print["the  factor  has degree ",Length[co]-1];
  Print["its leading  coefficient  is ",co⟦Length[co]⟧];
  co1=Expand[co⟦1⟧ ];
  Print["its  constant coefficient  is ",co1];
  cop=Replace[co1, _. _?Negative → 0, {1}](*level 1 here ?*);
  con=cop-co1;
  Print["R0J is"];
```

```
  R0J=con/cop//FullSimplify;
{R0J,co}
]

 Hirono[S_, intRows_, intCols_] :=
(*Hirono-Okada Network Reduction Module*)
 Module[
  {S11, S12, S21, S22, S11plus, Sred},

  S11 = S⟦intRows, intCols⟧;
  S12 = S⟦intRows, Complement[Range[Dimensions[S]⟦2⟧], intCols]⟧;
  S21 = S⟦Complement[Range[Dimensions[S]⟦1⟧], intRows], intCols⟧;
  S22 = S⟦Complement[Range[Dimensions[S]⟦1⟧], intRows], Complement[Range[Dimensions[S]⟦2

  S11plus = PseudoInverse[S11];
  Sred = Simplify[S22 - S21 . S11plus . S12];
  Sred
]

 (*Verify Hirono module*)
verHir[RHS_,var_,intRows_]:=Module[
{extRows,sub,rhsRed,fpRed},extRows=Complement[Range[Length[var]],intRows];
sub=Solve[RHS⟦intRows⟧==0,var⟦intRows⟧]⟦1⟧;
rhsRed=RHS⟦extRows⟧/. sub//Simplify;
fpRed=Solve[rhsRed==0,var⟦extRows⟧]⟦1⟧;
{sub,rhsRed,fpRed}]

(*Test using Example 7
RHS={k1-k2 a+k5 d,k2 a-k3 b,k3 b-k4 c-k6 c,k4 c-k5 d};
var={a,b,c,d};
intRows={1,2};
verHir[RHS,var,intRows]*)

extP[mod_,inf_]:=
Module[{X,Xi,qv,ov,ngm,fv,eq},X=mod⟦2⟧;Xi=X⟦inf⟧;
 qv=Array[q,Length[Xi]];
 ov=Table[1,{j,Length[Xi]}];ngm=NGM[mod,inf];F=ngm⟦4⟧;V=ngm⟦5⟧;fv=ov . F;
 eq=(qv . F)*qv-qv*fv+(ov-qv) . V];

RUR[mod_, ind_:{1}, cn_ : {}] (*ind is a list*):=
Module[{RHS, var, par, elim,ratsub,pol,rat1},
       RHS = mod⟦1⟧/.cn; var = mod⟦2⟧; par = mod⟦3⟧;
       elim = Complement[Range[Length[var]], ind];
       ratsub = seFZ[Solve[Delete[Thread[RHS == 0], ind],
       var⟦elim⟧]];
      pol =Numerator[Together[RHS//.ratsub]];
        RHS⟦ind⟧/.ratsub;(*Collect[GroebnerBasis[num,
```

```
        Join[par, var⟦ind⟧], var⟦elim⟧,
        MonomialOrder→EliminationOrder],var⟦ind⟧]; *)
      rat1=Append[(ratsub/.var⟦ind⟧→1),var⟦ind⟧→1];
    {ratsub, pol,rat1}
      ]

GBH[pol_,var_,sc_,cn_:{}]:=Module[{li,pa},
li={pol,sc};pa=Complement[Variables[li],{var}];
GroebnerBasis[{Numerator[Together[pol]],
Numerator[Together[sc]]}/.cn,pa,{var},
MonomialOrder→EliminationOrder]];
Stodola[pol_,var_] :=Equal@@Sign[CoefficientList[pol,var]]
```

```
mSim[mod_,cN_, cInit_,T_:100,exc_:{}]:=
Module[{dyn, X,vart,diff,diffN,initcond,eqN,ndesoln,ind},
dyn=mod⟦1⟧;X=mod⟦2⟧;vart=Through[X[t]];
diff= D[vart,t] - (dyn/.Thread[X→vart]);
diffN=diff//.cN;
initcond = (vart/.t→0)- cInit;
eqN=Thread[Flatten[{diffN, initcond}] == 0];
ndesoln = Chop[NDSolveValue[eqN,vart,{t, 0, T}]];
ind=Complement[Range[Length[X]],exc];
pl=Plot[ndesoln⟦ind⟧,{t,0,T},AxesLabel→{"t"," "}];pl];
```

```
Stab[mod_,cfp_,cn_:{}]:=Module[{dyn,X,par,jac,jacfp,eig},
 dyn=mod⟦1⟧;X=mod⟦2⟧;par=mod⟦3⟧;
 jac=Grad[dyn,X];
 jacfp=jac//.cfp;
 eig=Eigenvalues[jacfp/.cn]
]
(*Stab[SEIR,cfp⟦1⟧]//FullSimplify*)

Sta[jac_,X_,Xv_]:=Map[Max[Re[Eigenvalues[jac/.Thread[X→♯]]]]&,Xv];

```

```
(*The first focal value for the differential equation
Overscript[x, .]=-ωy+Underscript[∑, i+j≥2]Subscript[F, ij]/(i!j!)x^iy^j, Overscript[y,
Subscript[L, 1]=Subscript[F, 30]+Subscript[F, 12]+Subscript[G, 03]+Subscript[G, 21]+1/ω
L1Planar[fg_,var_,equilcon_:{}]:=
Module[{J,xyshift,Tm,Tinvuv,FG,derivatives,a,b,i,j,L1,x,y,F,G,normForm},
(*Variables*){x,y}=var;
(*Jacobian at equilibrium*)
J=Simplify[D[fg,{var}]/. equilcon];
(*Shift variables to equilibrium*)
xyshift={x→x+(x/. equilcon),y→y+(y/. equilcon)};
(*Transformation matrix*)
```

```
Tm={{1,0},{-a/Global`ome,-b/Global`ome}};
Tinvuv=Inverse[Tm] . {Global`u,Global`v};
(*Transformed FG*)
FG=(Tm . fg/. xyshift)/. {x→Tinvuv⟦1⟧,y→Tinvuv⟦2⟧}/. {a→J⟦1,1⟧,b→J⟦1,2⟧};
(*Compute derivatives*)
derivatives={};
For[i=0,i≤3,i++,For[j=0,j≤3-i,j++,derivatives=Join[derivatives,{Subscript[F,i,j]→(D[FG⟦
(*Compute L1 coefficient*)
L1=Subscript[F,3,0]+Subscript[F,1,2]+Subscript[G,0,3]+Subscript[G,2,1]+1/Global`ome*(Sub
(*Substitute derivatives into L1*)
L1=L1/. derivatives;
(*Construct the normal form up to cubic terms*)
F1=Sum[Subscript[F,i,j] Global`u^i Global`v^j,{i,0,3},{j,0,3-i}];
G1=Sum[Subscript[G,i,j] Global`u^i Global`v^j,{i,0,3},{j,0,3-i}];
normForm={F1,G1}/. derivatives;
(*Return L1 coefficient and normal form*)
{L1,FG,normForm}
];
DerEq[f_,var_,equilcon_]:=Module[
{derivatives,order,deriv,i,j,k,A,B,CC,DD,EE,x,y,z,par,cp},
n=Length[f];
A=D[f,{var}]/.equilcon;
{x,y,z}=var;par=Par[f,var];cp=Thread[par>0];
derivatives={};
order=2;
For[i=0,i≤order,i++,For[j=0,j≤order-i,j++,For[k=0,k≤order-i-j,k++,
deriv=Simplify[D[f,{x,i},{y,j},{z,k}]/.equilcon];
derivatives=Join[derivatives,Simplify[{Subscript[F, i,j,k]→deriv⟦1⟧,Subscript[G, i,j,k]-
]]];
B[x_,y_]:=Sum[{Subscript[F, Idx[{k,l},n]],Subscript[G, Idx[{k,l},n]],Subscript[H, Idx[{
(*CC[x_,y_,z_]:=Sum[{Subscript[F, Idx[{k,l,m},n]],Subscript[G, Idx[{k,l,m},n]],Subscrip
(* because of the bimolecularity, all derivatives of order 3 and higher vanish *)
CC[x_,y_,z_]:={0,0,0};
DD[x_,y_,z_,s_]:={0,0,0};
EE[x_,y_,z_,s_,t_]:={0,0,0};
(* the reason for the double symbols CC, DD, EE is that C, D, E are protected in Mathemat
{A,B,CC(*,DD,EE*)}
];
GetVec[A_,om_]:=Module[{n,mtx,pconj,q,qconj,normalize},
n=Length[A];
mtx=A-om I IdentityMatrix[n];
q=NullSpace[mtx⟦Range[1,n-1]⟧]⟦1⟧;
(* Notice that q is not normalised. The normalisation has relevance only when the second
mtx=A^T-om I IdentityMatrix[n];
pconj=NullSpace[mtx⟦Range[1,n-1]⟧]⟦1⟧;
normalize=FullSimplify[pconj . q];
pconj=pconj/normalize;
```

```
qconj=FullSimplify[ComplexExpand[q*]];
{pconj,q,qconj}
];
L13[A_,B_,CC_,cp_]:=
Module[{n,pconj,q,qconj,v1,v2,v3,c1,numer,denom,a,b,c,d,L1κω},
n=Length[A];
{pconj,q,qconj}= GetVec[A,ome];
v1=CC[q,q,qconj];
v2=B[q,Inverse[-A] . B[q,qconj]];
v3=B[qconj,Inverse[2I ome IdentityMatrix[n]-A] . B[q,q]];
c1=pconj . (1/2 v1+v2+1/2 v3);
(* We take the real part in a bit complicated way:
Re (a+bⅈ)/(c+dⅈ)=((ac+bd)/(c^2+d^2)).
It seems to be faster than the standard solution would be. *)
numer=Numerator[c1];
denom=Denominator[c1];
a=Simplify[ComplexExpand[Re[numer]],cp];
b=Simplify[ComplexExpand[Im[numer]],cp];
c=Simplify[ComplexExpand[Re[denom]],cp];
d=Simplify[ComplexExpand[Im[denom]],cp];
L1κω=Simplify[(a c+b d)/(c^2+d^2)]];
L23[A_,B_,CC_:{0,0,0},DD_:{0,0,0},EE_:{0,0,0}]:=
Module[{n,Id,omega,invA,inv2,inv3,pconj,q,qconj,h,prec,c,invbig},
n=Length[A];
Id=IdentityMatrix[n];
omega=Sqrt[Det[A]/Tr[A]];
invA=Inverse[A];
inv2=Simplify[Inverse[2 omega I Id-A]];
inv3=Simplify[Inverse[3 omega I Id-A]];
{pconj,q,qconj}= GetVec[A,omega];
q=FullSimplify[q/. {ω→omega}];
pconj=FullSimplify[pconj/. {ω→omega}];
Subscript[h, 2,0]=FullSimplify[inv2 . B[q,q]];
Subscript[h, 1,1]=FullSimplify[-invA . B[q,q*]];
prec=FullSimplify[CC[q,q,q*]+2 B[q,Subscript[h, 1,1]]+B[q*,Subscript[h, 2,0]]];
Subscript[c, 1]=FullSimplify[1/2 (pconj . prec)];
invbig=FullSimplify[Inverse[Join[Join[omega I Id-A,{q}ᵀ,2],{Join[pconj,{0}]}]]];
Subscript[h, 2,1]=FullSimplify[invbig . Join[FullSimplify[prec-2 Subscript[c, 1] q],{0}]
Subscript[h, 3,0]=FullSimplify[inv3 . (CC[q,q,q]+3 B[q,Subscript[h, 2,0]])];
Subscript[h, 3,1]=FullSimplify[inv2 . (DD[q,q,q,q*]+3 CC[q,q,Subscript[h, 1,1]]+3 CC[q,q
Subscript[h, 2,2]=FullSimplify[-invA . (DD[q,q,q*,q*]+4 CC[q,q*,Subscript[h, 1,1]]+CC[q*,
Subscript[c, 2]=FullSimplify[1/12 (pconj . (EE[q,q,q,q*,q*]+DD[q,q,q,Subscript[h, 2,0]*]+
ComplexExpand[Re[Subscript[c, 2]]]
];
(* Converts between products and sums *)
toSum= (# /. Times → Plus) &;
toProd=(# /.  Plus→Times ) &;
```

```
(*RescaleODE[f_,equilcon_]:=Module[{X,Y,Z,fscaled,fκ1,fnew,
αβγ,A,i,factor,κ2αβγ},
X=x/.equilcon;
Y=y/.equilcon;
Z=z/.equilcon;
fscaled=Simplify[DiagonalMatrix[{1/X,1/Y,1/Z}] . (f/.{x→u X,y→v Y,z→w Z}),κpositive];
fnew=fscaled/.{Subscript[κ, 1]→1,Subscript[κ, 2]→1,Subscript[κ, 3]→1,Subscript[κ, 4]→1
αβγ=Simplify[fscaled/fnew,κpositive];
(* now we hide in α,β,γ all the common factors *)
A=D[fnew,{{u,v,w}}]/.{u→1,v→1,w→1};
For[i=1,i≤Length[A],i++,{
factor=LCM[Denominator[A⟦i⟧/Max[Abs[A⟦i⟧]]]/.List→Sequence]/Max[Abs[A⟦i⟧]];
fnew⟦i⟧=fnew⟦i⟧factor;
αβγ⟦i⟧=αβγ⟦i⟧/factor;
}];
fnew=DiagonalMatrix[{α,β,γ}] . fnew;
κ2αβγ={α→αβγ⟦1⟧,β→αβγ⟦2⟧,γ→αβγ⟦3⟧};
{fnew,κ2αβγ}
];
DerSc[f_]:=Module[{equil,derivatives,order,deriv,i,j,k,A,B,CC,DD,EE},
(*Assumes {u→1,v→1,w→1}*)
n=Length[f];
equil={u→1,v→1,w→1};
A=D[f,{{u,v,w}}]/.equil;
derivatives={};
order=2;
For[i=0,i≤order,i++,For[j=0,j≤order-i,j++,For[k=0,k≤order-i-j,k++,
deriv=Simplify[D[f,{u,i},{v,j},{w,k}]/.equil];
derivatives=Join[derivatives,Simplify[{Subscript[F, i,j,k]→deriv⟦1⟧,Subscript[G, i,j,k]-
]]];
B[x_,y_]:=Sum[{Subscript[F, Idx[{k,l},n]],Subscript[G, Idx[{k,l},n]],Subscript[H, Idx[{
(*CC[x_,y_,z_]:=Sum[{Subscript[F, Idx[{k,l,m},n]],Subscript[G, Idx[{k,l,m},n]],Subscrip
(* because of the bimolecularity, all derivatives of order 3 and higher vanish *)
CC[x_,y_,z_]:={0,0,0};
DD[x_,y_,z_,s_]:={0,0,0};
EE[x_,y_,z_,s_,t_]:={0,0,0};
(* the reason for the double symbols CC, DD, EE is that C, D, E are protected in Mathemat
{A,B,CC,DD,EE}
];
*)
End[];
EndPackage[];
$ContextPath=DeleteDuplicates[Append[$ContextPath,"model`Private`"]];
```