DEMO OF DETERMINANT OF JACOBIAN OF CHEM REACTION NETWORK SOFTWARE

Here, we consider the differential equation

$$\frac{dx}{dt} = f(x) = Sv(x)$$

which acts on the non-negative orthant $R^d_{\geq 0}$,
where S is a d x d' Stoichiometric Matrix,
and v(x) is a vector with d' components consisting of monomials multiplied by rate constants. These are called fluxes.
This differential equation is associated to a chemical network modeled with mass-action kinetics.
Runs in MMa 4.2 and higher.

Input the Differential Equation

Input the Stoichiometric matrix S

```
In[1]:= S = {{-1, 0, 2}, {-1, -1, 0}, {0, -1, -1}, {1, 0, 0}, {0, 1, 0}}(*stoichiometric matrix*)

Out[1]= {{-1, 0, 2}, {-1, -1, 0}, {0, -1, -1}, {1, 0, 0}, {0, 1, 0}}

In[2]:= S // MatrixForm

Out[2]//MatrixForm=
```

$$\begin{pmatrix} -1 & 0 & 2 \\ -1 & -1 & 0 \\ 0 & -1 & -1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

```
In[3]:= (* All of the reactions in the network associated to the above stoichiometric
           matrix are irreversible, because none of the negatives of the three
           columns appear.  If all the reactions in a reaction network are reversible,
          then it is only necessary to input one column for each reaction
           (representing either the forward or backward direction),
          because the makeReversible command can be used later.
         *)

In[4]:= (*this command is equivalent to MatrixRank
           (which is not implemented in earlier mathematica versions)*)

        mr3[m_] := Length[Select[RowReduce[m], ! (Dot[#, #] === 0) &]];
        mr3[S]

Out[5]= 3
```

Make the flux vector and list of species concentrations automatically from the Stoiciometric matrix

```
In[6]:= (* Each column of S corresponds to a chemical reaction,
          modeled by a monomial multiplied by a rate constant (flux).
            In particular, writing S = (S_ij), x = (x_1,...,x_d)^T,
          and v(x) = (v_1(x),...,v_d'(x))^T, we have
           v_j(x) = k_j ∏_{i=1}^{d} x_i^{|min{0,S_ij}|}.

               The function 'makeMonomial' will construct v(x), and take S as input.*)

In[7]:= makeMonomial[Smatrix_] := Module[{tmp, lista, pow, mat},
          refine[S_] := S /. Map[# → 1 &, Variables[S]]; mat = refine[Smatrix];
          pow = Table[0, {Length[mat[[1]]]}];
          tmp = Table[If[mat[[i, j]] ≥ 0, 1, If[pow[[j]] > 0, a[i]^(-Smatrix[[i, j]]), pow[[j]]++;
              k[j] a[i]^(-Smatrix[[i, j]]))]], {i, 1, Length[mat]}, {j, 1, Length[mat[[1]]]}];
          lista = Map[Apply[Times, #] &, Transpose[tmp]]; lista]

In[8]:= (* 'makeMonomialNMA' is a command to make the flux vector v(x) in the more general
            situation of non-mass-action kinetics.  The entry of the flux vector for
            the jth reaction will be of the form k[j][a[i_1],a[i_2],...,a[i_n]],
          where a[i_1],...,a[i_n] are the reactants.  In other words,
          we assume the reaction rate is an indeterminate
           function of the reactant concentrations. *)

In[9]:= makeMonomialNMA[Smatrix_] :=
         Module[{reactantlist, tmp}, tmp = Smatrix;
          tmp = Map[If[# < 0, 1, 0] &, tmp, {2}]; tmp = Transpose[tmp]; reactantlist =
           Table[tmp[[i, j]] * a[j], {i, 1, Dimensions[tmp][[1]]}, {j, 1, Dimensions[tmp][[2]]}];
          reactantlist = DeleteCases[reactantlist, 0, {2}];
          Table[Apply[k[i], reactantlist[[i]]], {i, 1, Length[reactantlist]}]]

In[10]:= (* The function 'variables' takes a vector of monomials (fluxes) as input,
           and returns a vector of all the variables (species) appearing in the vector of
            monomials.  WARNING: if the stoichiometric matrix is not fully reversible,
           then not all species may appear as variables in the monomial vector,
           so this command may return too few variables. *)
```

```
In[11]:= variables[monom_] :=
          Module[{list}, list = Select[Variables[monom], Head[#] == Head[a[1]] ||
                (Head[#] == Power && Head[Level[#, 1][[1]]] == Head[a[1]]) &];
            Table[ If[ Length[list[[i]]] == 1, list[[i]], ( Level[list[[i]], 1])[[1]] ],
             {i, 1, Length[list]}]] // Union

In[12]:= (* The command ' svars' can be used to generate the species variables whenever the
            flux vector has been generated using the makeMonomial or makeMonomialNMA
            commands.  It takes a stoichiometric matrix as input and simply returns a list
            {a[1], a[2], ..., a[n]} where n is the number of rows in the stoichiometric
            matrix (a.k.a. the number of species).  This can be preferable to using the
            above command since it avoids the problem of returning too few variables. *)

In[13]:= svars[Smatrix_] := Table[a[i], {i, 1, Length[Smatrix]}];

In[14]:= (* The command ' makeReversible' takes a stoichiometric matrix
            that only has one direction entered for each reaction and makes
            it fully reversible (so the number of columns will be doubled)*)

In[15]:= makeReversible[S_] := Module[{tmp}, tmp = Transpose[S]; Flatten[
              Table[ {tmp[[i]], -tmp[[i]]}, {i, 1, Length[tmp]}], 1] // Transpose]

In[16]:= rS = makeReversible[S]

Out[16]= {{-1, 1, 0, 0, 2, -2}, {-1, 1, -1, 1, 0, 0},
          {0, 0, -1, 1, -1, 1}, {1, -1, 0, 0, 0, 0}, {0, 0, 1, -1, 0, 0}}

In[17]:= monomialvec = makeMonomial[rS]; monomialvec // MatrixForm

Out[17]//MatrixForm=
```

$$\begin{pmatrix} a[1]\,a[2]\,k[1] \\ a[4]\,k[2] \\ a[2]\,a[3]\,k[3] \\ a[5]\,k[4] \\ a[3]\,k[5] \\ a[1]^2\,k[6] \end{pmatrix}$$

```
In[18]:= (* ' vars' will contain all the variables of `monomialvec'= v (x) *)

In[19]:= vars = variables[monomialvec]

Out[19]= {a[1], a[2], a[3], a[4], a[5]}
```

Right - hand side of the ODE

```
In[20]:= (* We' ll compute fns = dx/dt = Sv (x) = S.monomialvec *)
```

```
In[21]:= fns = rS.monomialvec;
         fns // MatrixForm
```

```
Out[22]//MatrixForm=
⎛ -a[1] a[2] k[1] + a[4] k[2] + 2 a[3] k[5] - 2 a[1]^2 k[6] ⎞
⎜ -a[1] a[2] k[1] + a[4] k[2] - a[2] a[3] k[3] + a[5] k[4] ⎟
⎜ -a[2] a[3] k[3] + a[5] k[4] - a[3] k[5] + a[1]^2 k[6]    ⎟
⎜ a[1] a[2] k[1] - a[4] k[2]                               ⎟
⎝ a[2] a[3] k[3] - a[5] k[4]                               ⎠
```

```
In[23]:= (* the Jacobian of fns wrt Vars *)
         jac[Fns_, Vars_] :=
          Table[D[Fns[[i]], Vars[[j]]], {i, 1, Length[Fns]}, {j, 1, Length[Vars]}]
```

Computations start here

The Jacobian of the RHS of the ODE and their Determinants

```
In[24]:= (* Computation of Sv' (x)*)
```

```
In[25]:=   J = jac[fns, vars];
         J // MatrixForm
```

```
Out[26]//MatrixForm=
⎛ -a[2] k[1] - 4 a[1] k[6]   -a[1] k[1]               2 k[5]              k[2]   0     ⎞
⎜ -a[2] k[1]                 -a[1] k[1] - a[3] k[3]   -a[2] k[3]          k[2]   k[4]  ⎟
⎜ 2 a[1] k[6]                -a[3] k[3]               -a[2] k[3] - k[5]   0      k[4]  ⎟
⎜ a[2] k[1]                  a[1] k[1]                0                   -k[2]  0     ⎟
⎝ 0                          a[3] k[3]                a[2] k[3]           0      -k[4] ⎠
```

```
In[27]:= (* Computing the Craciun-Feinberg Determinant
             (cfdet) = Det (Sv' (x) - I)= Det (J - I)   *)
```

```
In[28]:= cfDet[Jac_] := Det[Jac - IdentityMatrix[Length[Jac]]] // Expand;
```

```
In[29]:= cf = cfDet[J]
```

```
Out[29]= -1 - a[1] k[1] - a[2] k[1] - k[2] - a[2] k[3] - a[3] k[3] - a[1] a[2] k[1] k[3] -
          a[2]^2 k[1] k[3] - a[2] a[3] k[1] k[3] - a[2] k[2] k[3] - a[3] k[2] k[3] - k[4] -
          a[1] k[1] k[4] - a[2] k[1] k[4] - k[2] k[4] - k[5] - a[1] k[1] k[5] - a[2] k[1] k[5] -
          k[2] k[5] - a[3] k[3] k[5] + a[2] a[3] k[1] k[3] k[5] - a[3] k[2] k[3] k[5] -
          k[4] k[5] - a[1] k[1] k[4] k[5] - a[2] k[1] k[4] k[5] - k[2] k[4] k[5] -
          4 a[1] k[6] - 4 a[1]^2 k[1] k[6] - 4 a[1] k[2] k[6] - 4 a[1] a[2] k[3] k[6] -
          4 a[1] a[3] k[3] k[6] - 2 a[1]^2 a[2] k[1] k[3] k[6] - 4 a[1] a[2] k[2] k[3] k[6] -
          4 a[1] a[3] k[2] k[3] k[6] - 4 a[1] k[4] k[6] - 4 a[1]^2 k[1] k[4] k[6] - 4 a[1] k[2] k[4] k[6]
```

```
In[30]:= (* The functions ' coeffsList' and ' coeffs'
             count the number of terms in a determinant expansion,
            find all the numerical coefficients of all the terms in the determinant expansion,
            and count the number of times each coefficient appears in the expansion *)
```

```
In[31]:= coeffsList[expansion_] :=
          DeleteCases[ Flatten[CoefficientList[expansion, Variables[expansion]]], 0]
```

```
In[32]:= coeffs[expansion_] := Module[{tmp}, tmp = coeffsList[expansion];
           Print["The number of terms in the det expansion is ", Length[tmp], ","];
           Print["and (a,b) says that the number of terms with coeff. a is b:"];
           Map[{#, Count[tmp, #]} &, Union[tmp]]]

In[33]:= (* Looking at the coefficients of the Craciun-Feinberg determinant (cfdet) *)

In[34]:= coeffs[cf]
```

The number of terms in the det expansion is 37,

and (a,b) says that the number of terms with coeff. a is b:

```
Out[34]= {{-4, 10}, {-2, 1}, {-1, 25}, {1, 1}}
```

The Core Determinant

The core determinant is the determinant of the Jacobian, fns'(x), restricted to range S.
It is natural for analyzing chem reactions with no outflows, (see Helton-Klep-Gomez).

```
In[35]:= (* To compute the Core Determinant we use the formula
                (CoreDet) = lim_{t→0} 1/t^(d-rank (S)) Det (Sv' (x) - tI)        *)

         coreDet[Jac_, Smatrix_] := Module[{d, cdet}, d = Length[Jac] - mr3[Smatrix];
           cdet = (Det[Jac - t IdentityMatrix[Length[Jac]]]) / t^d // Expand;
           cdet /. t → 0]

In[36]:= core = coreDet[J, rS]

Out[36]= a[2] a[3] k[1] k[3] k[5] - a[3] k[2] k[3] k[5] - a[1] k[1] k[4] k[5] - a[2] k[1] k[4] k[5] -
          k[2] k[4] k[5] - 2 a[1]^2 a[2] k[1] k[3] k[6] - 4 a[1] a[2] k[2] k[3] k[6] -
          4 a[1] a[3] k[2] k[3] k[6] - 4 a[1]^2 k[1] k[4] k[6] - 4 a[1] k[2] k[4] k[6]

In[37]:= (* looking at the coefficients of the Core Determinant expansion *)

In[38]:= coeffs[core]
```

The number of terms in the det expansion is 10,

and (a,b) says that the number of terms with coeff. a is b:

```
Out[38]= {{-4, 4}, {-2, 1}, {-1, 4}, {1, 1}}
```