Enumerate reaction networks composed with heteromultimer and single transformations November 2014

This particular code is to enumerate all possible reaction networks (with certain dimensions, $n \times m$ where n is speciese number m is reaction number) composed of three types of interactions:

$$A + B \rightarrow C$$
$$\mathrm{D} \rightarrow E + F$$
$$G \rightarrow H$$

Those are heterodimerization, disassociation and single transformation.

With those three types of elementary reactions, we could construct a set of reaction networks, then we could use DSR graphs and bipartie to characteristic those networks if they are multistationary and has closed competition loop.

But before to go through such checking, we need to preclude situations that clearly not a complex balanced reaction network, by which mean it obeys the following three constraints:
0. Only allow elementary reactions described above, which is the starting point to construct the matrix;
   (more complex version should be including $I \rightarrow 2\,J$ and $2\,K \rightarrow L$ in future)
   list all possibly combinations and select m of those into matrix (sequence does not matter), then seperate into pos and neg matrices

1. Mass conservation;
   (it's very difficult to check, currently implemented without this checking but manual checking afterwards)

2. Complex banlanced: each species has at least one in flow and one out flow;
   (easy to check)
   Check pos matrix and neg matrix, (there is no zeros in columSum)

Then we need to:
4. check competition: there are at least one species has two -1 and there is another -1 in each of the according reactions;
   (in neg matrix, check if there are any two intersections of colSum and rowSum $\geqslant -2$ in a row are -1)
   find col indices, then get the other two species (competitors)

5. check loop: take indices of competitors, do the network searching, find the loop from one to another and then from the other to this one.

Cluster matrix into four categories: bistable with closed competition loop, bistable without closed competition loop, monostable with closed competition loop, monostable without closed competition loop.

Some functions might need:
sprintf(fmt, x1, ..., xn)
StringTools[Join](stringList, sep)
StringTools[CaseJoin](stringList)

mkdir(dirName)
FileTools[RemoveDirectory](dirName, options)
FileTools[Remove](file, file2, ...)
ListDirectory(dir, opt1, opt2, ...)
ArrayTools[AddAlongDimension](A,dim)

Initializations

```
> restart :
> interface(rtablesize = 400) :
> with(ListTools) :
> with(LinearAlgebra) :
> with(VectorCalculus) :
> with(GraphTheory) :
> with(combinat) :
> #with(MTM) :
> with(ArrayTools) :
> _Envsignum0 := 0 :
>
```

▶ Functions for multistationality checking (execute before proceeding)

▼ Functions for constructing stoichiomatric matrix and examine the existence of competition and closed loop.

▼ 1. constructing stoichiomatric vectors based on the certain reaction patterns

▼ *List all reaction types based on the number of species.* $R_n = \binom{n}{2}\cdot 2 + \binom{n}{3}\cdot 6$

```
> listRs := proc(n)
      local R, r, se, i, j, k, sign, l :
      r := $\binom{n}{2}\cdot 2 + \binom{n}{3}\cdot 6$ :
      R := Matrix(r, n) :
      # Now construct the reaction pattern matrix
      i := 1 :

      # here we first consider single transformation
```

```
        for se from -1 to 1 by 2 do
          for j from 1 to n by 1 do
            for k from j + 1 to n by 1 do
              R[i, j] := se :
              R[i, k] := -se :
              i := i + 1 :
            end do:
          end do:
        end do:

        # now consider heterodimerization and disassociation
        for sign from -1 to 1 by 2 do
          for j from 1 to n by 1 do
            for k from j + 1 to n by 1 do
              for l from k + 1 to n by 1 do
                R[i, j] := sign :
                R[i, k] := -sign :
                R[i, l] := -sign :
                i := i + 1 :
                R[i, j] := sign :
                R[i, k] := sign :
                R[i, l] := -sign :
                i := i + 1 :
                R[i, j] := sign :
                R[i, k] := -sign :
                R[i, l] := sign :
                i := i + 1 :
              end do:
            end do:
          end do:
        end do:

        # in this case we don't consider homodimerization and disassociation

        # Here transpose the R
        # R:= Transpose(R) :
         # no need to ranspose, need to assign rows to untransposed A's rows
        return(R) :
      end proc:
```

Here is a long function to enumerate first two reaction pattern (will reduce large mount of symmetric reactions).

```
> listR2 := proc(n)
        local R2, r2, R3, r3, R4, r4 :
        if n ≥ 4 then
          r2 := 29·2 :
```

```
R2 := Matrix(r2, 4) :

# in this case we don't consider homodimerization and disassociation

## now we construct the reaction pattern with n = 4
R2[1] := ⟨1,-1⟩ : R2[2] := ⟨0, 0, 1,-1⟩ :
R2[3] := ⟨1,-1⟩ : R2[4] := ⟨0, 1,-1⟩ :
R2[5] := ⟨1,-1⟩ : R2[6] := ⟨1, 0,-1⟩ :
R2[7] := ⟨1,-1⟩ : R2[8] := ⟨0,-1, 1⟩ :
R2[9] := ⟨1,-1⟩ : R2[10] := ⟨-1, 0, 1⟩ :
R2[11] := ⟨1,-1⟩ : R2[12] := ⟨-1, 1⟩ :
R2[13] := ⟨1,-1⟩ : R2[14] := ⟨0, 1,-1,-1⟩ :
R2[15] := ⟨1,-1⟩ : R2[16] := ⟨1, 0,-1,-1⟩ :
R2[17] := ⟨1,-1⟩ : R2[18] := ⟨0,-1, 1,-1⟩ :
R2[19] := ⟨1,-1⟩ : R2[20] := ⟨-1, 0, 1,-1⟩ :
R2[21] := ⟨1,-1⟩ : R2[22] := ⟨0, 1, 1,-1⟩ :
R2[23] := ⟨1,-1⟩ : R2[24] := ⟨1, 0, 1,-1⟩ :
R2[25] := ⟨1,-1⟩ : R2[26] := ⟨0,-1, 1, 1⟩ :
R2[27] := ⟨1,-1⟩ : R2[28] := ⟨-1, 0, 1, 1⟩ :
R2[29] := ⟨1,-1⟩ : R2[30] := ⟨-1,-1, 1⟩ :
R2[31] := ⟨1,-1⟩ : R2[32] := ⟨1, 1,-1⟩ :
R2[33] := ⟨1,-1,-1⟩ : R2[34] := ⟨0, 1,-1,-1⟩ :
R2[35] := ⟨1,-1,-1⟩ : R2[36] := ⟨1, 0,-1,-1⟩ :
R2[37] := ⟨1,-1,-1⟩ : R2[38] := ⟨0,-1,-1, 1⟩ :
R2[39] := ⟨1,-1,-1⟩ : R2[40] := ⟨-1,-1, 0, 1⟩ :
R2[41] := ⟨1,-1,-1⟩ : R2[42] := ⟨0, 1, 1,-1⟩ :
R2[43] := ⟨1,-1,-1⟩ : R2[44] := ⟨1, 1, 0,-1⟩ :
R2[45] := ⟨1,-1,-1⟩ : R2[46] := ⟨0, 1,-1, 1⟩ :
R2[47] := ⟨1,-1,-1⟩ : R2[48] := ⟨-1, 1, 0, 1⟩ :
R2[49] := ⟨1,-1,-1⟩ : R2[50] := ⟨-1, 1, 1⟩ :
R2[51] := ⟨1, 1,-1⟩ : R2[52] := ⟨0, 1, 1,-1⟩ :
R2[53] := ⟨1, 1,-1⟩ : R2[54] := ⟨1, 1, 0,-1⟩ :
R2[55] := ⟨1, 1,-1⟩ : R2[56] := ⟨0, 1,-1, 1⟩ :
R2[57] := ⟨1, 1,-1⟩ : R2[58] := ⟨1,-1, 0, 1⟩ :


if n ≥ 5 then
  ## now we add the reaction pattern with n = 5

  r3 := 43·2 :
  R3 := Matrix(r3, 5) :
  R3[1 ..r2] := R2[ ]:

  R3[59] := ⟨1,-1⟩ : R3[60] := ⟨0, 0, 1,-1,-1⟩ :
  R3[61] := ⟨1,-1⟩ : R3[62] := ⟨0, 0, 1, 1,-1⟩ :
  R3[63] := ⟨1,-1,-1⟩ : R3[64] := ⟨0, 0, 1,-1,-1⟩ :
  R3[65] := ⟨1,-1,-1⟩ : R3[66] := ⟨1, 0, 0,-1,-1⟩ :
  R3[67] := ⟨1,-1,-1⟩ : R3[68] := ⟨0, 0,-1, 1,-1⟩ :
        R3[69] := ⟨1,-1,-1⟩ : R3[70] := ⟨-1, 0, 0, 1,-1⟩ :
        R3[71] := ⟨1,-1,-1⟩ : R3[72] := ⟨0, 0, 1, 1,-1⟩ :
        R3[73] := ⟨1,-1,-1⟩ : R3[74] := ⟨1, 0, 0, 1,-1⟩ :
        R3[75] := ⟨1,-1,-1⟩ : R3[76] := ⟨0, 0,-1, 1, 1⟩ :
        R3[77] := ⟨1,-1,-1⟩ : R3[78] := ⟨-1, 0, 0, 1, 1⟩ :
        R3[79] := ⟨1, 1,-1⟩ : R3[80] := ⟨0, 0, 1, 1,-1⟩ :
        R3[81] := ⟨1, 1,-1⟩ : R3[82] := ⟨1, 0, 0, 1,-1⟩ :
        R3[83] := ⟨1, 1,-1⟩ : R3[84] := ⟨0, 0,-1, 1, 1⟩ :
        R3[85] := ⟨1, 1,-1⟩ : R3[86] := ⟨-1, 0, 0, 1, 1⟩ :

        if n ≥ 6 then
          ## now we add the reaction pattern with n = 6

          r4 := 46·2 :
          R4 := Matrix(r4, n) :
          R4[1 ..r3] := R3[ ] :

          R4[87] := ⟨1,-1,-1⟩ : R4[88] := ⟨0, 0, 0, 1,-1,-1⟩ :
          R4[89] := ⟨1,-1,-1⟩ : R4[90] := ⟨0, 0, 0, 1, 1,-1⟩ :
          R4[91] := ⟨1, 1,-1⟩ : R4[92] := ⟨0, 0, 0, 1, 1,-1⟩ :

          return(R4) :
        else
          return(R3) :
        end if:
      else
        return(R2) :
      end if:
    else
      error "ERROR: n is smaller than 4"
    end if:

  end proc:
```

2. Now we can construct stoichiomatric matrix based on the reaction patterns, and examine their properties.

Here, we construct the stoichiomatric matrices.

The total number of stoichiomatric matrices is $\binom{R_n}{m}$, which is still a huge number. But currently there seems no other better options.

We only consider when $m \leq 6$.

The function(s) to examine existence of competition and loops a stoichiomatric matrix.

```
>
```

The function to check if the stoichiomatric matrix is mass conserved.

First check the passed matrix $A_{m\times n}$ (must be in a consistent form)

```
> ismassconserved := proc(A)
    local R, N, NS, m, n, absAdd, Add, x, y, z, e, i, nsAdd, nsAbsAdd, a, b, c :
    m := 0 :
    n := Dimension(A)[2] :
    absAdd := AddAlongDimension(|A|, 1) :
    z := Search(0, absAdd) :
    if z = 0 then
      Add := AddAlongDimension(A, 1) :
      x := Search(0, VectorAdd(absAdd, Add, 1, -1)) :
      if x = 0 then
        y := Search(0, VectorAdd(absAdd, Add, 1, 1)) :
        if y = 0 then
          N := NullSpace(A) :
          e := numelems(N) :
          if e > 0 then
            NS := Matrix(e, n) :
            for i from 1 to e by 1 do
              NS[i] := N[i] :
            end do:
            nsAbsAdd := AddAlongDimension(|NS|, 1) :
            a := Search(0, nsAbsAdd) :
            if a = 0 then
              nsAdd := AddAlongDimension(NS, 1) :
              b := Search(0, VectorAdd(nsAbsAdd, nsAdd, 1, 1)) :
              if b = 0 then
                m := 1 :
              end if:
            end if:
          end if:
        end if:
      end if:
    end if:
    return(m) :
  end proc:
```

Construct and examine the properties of all stoichiomatric matrices.

```
> constrM := proc(n, m)
    local R, A, r, g, h, i, j, k, l, total, right, mc, V, R2, r2, inject, inject0, inject1, injectEx,
     injectEx0, injectEx1, injectEx2, injectEx3, fileName, matrixData, tA, interV :
    r := $\binom{n}{2}\cdot 2 + \binom{n}{3}\cdot 6$ :
```

```
$A := Matrix(m, n)$ :

$R := listRs(n)$ :
$R2 := listR2(n)$ :

if $n = 4$  then  $r2 := 29 \cdot 2$ :  end if:
if $n = 5$  then  $r2 := 43 \cdot 2$ :  end if:
if $n \geq 6$  then  $r2 := 46 \cdot 2$ :  end if:
if $n < 4$ then error "ERROR: n is smaller than 4" end if:

$total := \frac{r2}{2} \cdot \binom{r}{m-2}$ :

# here we use some variable to count how many reactions are correct.
$right := 0$ : $inject0 := 0$ : $inject1 := 0$ : $injectEx0 := 0$ : $injectEx1 := 0$ : $injectEx2$
  $:= 0$ : $injectEx3 := 0$ :
for $l$ from 1 to $r2 - 1$ by 2 do
   $A[1] := R2[l]$:
   $A[2] := R2[l+1]$:
   for $g$ from 1 to $r$ by 1 do
      # here we should check whether this is duplicate of the fixed two reactions.
      #######
      $A[3] := R[g]$:
      for $h$ from $g + 1$ to $r$ by 1 do
         $A[4] := R[h]$:
         #for i from h+1 to r by 1 do
            #$A[5] := R[i]$:
            #for j from i+1 to r by 1 do
               #$A[6] := R[j]$:

               # now we have matrix A, we need to exam A with constraints.
               $mc := ismassconserved(A)$ :
               if $mc = 1$ then


 # before this we should preclude the reactions violating mass conservation!

 ### the idea is construct a sequence of species based on the mass (sorted by
 reactions)

 ### if there is any species with two different positions then it violates mass
 conservation.
                 ### Or just record the sequential position for each species,
                 ### then find if there are species with two or more position numbers
                 $right := right + 1$ :
                 $tA := Transpose(A)$ :

                 $inject := isinjective(tA)$ :
```

```
                            if inject = 0 then
                                inject0 := inject0 + 1 :
                                fileName
  := sprintf("%1dspecies/injectivity/noninjective_%d.csv", n, inject0) :
                                ExportMatrix(fileName, tA, target = csv, format = rectangular,
  mode = ascii) :

                                injectEx := isinjectiveextended(tA) :
                                if injectEx = 0 then
                                    injectEx0 := injectEx0 + 1 :
                                    fileName
  := sprintf("%1dspecies/bistability/needToolbox_%d.csv", n, injectEx0) :
                                    ExportMatrix(fileName, tA, target = csv, format = rectangular,
  mode = ascii) :

                                elif injectEx = 1 then
                                    injectEx1 := injectEx1 + 1 :

                                elif injectEx = 2 then
                                    injectEx2 := injectEx2 + 1 :
                                    fileName := sprintf("%1dspecies/bistability/bistable_%d.csv",
  n, injectEx2) :
                                    ExportMatrix(fileName, tA, target = csv, format = rectangular,
  mode = ascii) :

                                elif injectEx = 3 then
                                    injectEx3 := injectEx3 + 1 :
                                else
                                    error "ERROR: injectivity extended of A is not any of 0 to 3."
                                end if:
                            elif inject = 1 then
                                inject1 := inject1 + 1 :
                            else
                                error "ERROR: the injectivity of A is neither 0 nor 1."
                            end if:
                        end if:
                    #end do:
                #end do:
            end do:
        end do:
    end do:

    V := [injectEx0, injectEx1, injectEx2, injectEx3, inject0, inject1, right, total, r, r2] :
    return(V) :
end proc:
>
```

Testing

Here we test all functions:

```
> V := constrM(4, 4) # just count right matrices ~ 12s
                    V := [0, 0, 0, 0, 0, 0, 2359, 18270, 36, 58]            (3.1)
> V := constrM(4, 4) # also count injective matrices ~ 38s
                 V := [0, 0, 0, 0, 554, 1805, 2359, 18270, 36, 58]          (3.2)
> V := constrM(4, 4) # also count injective matrices and export noninjective matrices ~ 40s
                 V := [0, 0, 0, 0, 554, 1805, 2359, 18270, 36, 58]          (3.3)
> V := constrM(4, 4) # also count injective extended matrices ~ 44s
                  V := [0, 7, 0, 63, 70, 178, 248, 18270, 36, 58]           (3.4)
> V := constrM(4, 4)
      # also count injective extended matrices and export bistable matrices ~ 57s
                  V := [0, 7, 0, 63, 70, 178, 248, 18270, 36, 58]           (3.5)
>
>
```