```
restart : with(LinearAlgebra) :   with(Optimization) :
# This MAPLE file takes a list of triangulations computed in sage (via TOPCOM package) and gives as
    output a series of polynomial expressions.
#  Each set of expressions f_1,...,f_m can be used to produce a region of multistationarity for the n −site
    phosphorylation system as f_1 >0,...,f_m >0.

# set n
 n := 2 :
# Since the tables are going to be big, increase the maximum allowed size for tables .
 interface(rtablesize = 4 + 2·n) :
# set T[-1] and the matrices A, C and Csimple.
T[-1] := 1 :
A := Transpose(Matrix([[1, 0, 0], [0, 1, 0], [0, 0, 1], seq([1, i, -i], i = 1 ..n), seq([1, i, 1 − i], i = 1
    ..n), [0, 0, 0]]));
C := Transpose(Matrix([[1, 0, 0], [0, 1, 0], [0, 0, 1], seq([T[i], 0, 0], i = 0 ..n − 1), seq([K[i]·T[i
    -1] + L[i]·T[i], K[i]·T[i-1], L[i]·T[i]], i = 0 ..n − 1), [-S, -E, -F]]))   ;
Csimple := Transpose(Matrix([[1, 0, 0], [0, 1, 0], [0, 0, 1], seq([1, 0, 0], i = 0 ..n − 1), seq([1,
    M[i], 1 − M[i]], i = 0 ..n − 1), [-S, -E, -F]]))
```

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

$$\begin{bmatrix} 1 & 0 & 0 & T_0 & T_1 & L_0 T_0 + K_0 & K_1 T_0 + L_1 T_1 & -S \\ 0 & 1 & 0 & 0 & 0 & K_0 & K_1 T_0 & -E \\ 0 & 0 & 1 & 0 & 0 & L_0 T_0 & L_1 T_1 & -F \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 1 & -S \\ 0 & 1 & 0 & 0 & 0 & M_0 & M_1 & -E \\ 0 & 0 & 1 & 0 & 0 & 1 - M_0 & 1 - M_1 & -F \end{bmatrix} \quad \textbf{(1)}$$

```
# Here we define the procedure Foundoriginaltriang that we will use in the end of the present script.
# This will be used when we need to recover the triangulation in L1 that gave a element in L7 used to
    obtain a region of multistationarity.

Foundoriginaltriang :=proc(original, T)
 local k, aux;
 aux := T;
 for k from 1 to numelems(original) do
 if {op(T)} subset {op(original[k][2])} then aux := original[k][1] fi
 od:
aux;
 end proc:

# Here we define validpolytopesindex as the set of triples that index all non zero 3 x3 minors of Csimple.
# This will be used to pass from L2 to L3.

validpolytopesindex := [ ] :
```

```
for i1 from 1 to ColumnDimension(Csimple) do
for i2 from i1 + 1 to ColumnDimension(Csimple) do
for i3 from i2 + 1 to ColumnDimension(Csimple) do
if Determinant(Csimple[1 ..3, [i1, i2, i3]]) ≠ 0 then validpolytopesindex
    := [op(validpolytopesindex), [i1, i2, i3]];
end if
end do end do end do:
# Here we import L1 from a file outputed from SAGE, this is step (1) in the Algorithm.

L1 := parse(ImportData( )) :
# Here we do step (2) of Algorithm to obtain L2 from L1.
# The variables "originals","originals2",..., will keep track from the passage from L1 to L2, L2 to L3 and
    so on.

L2 := { } :

# We define L2 as a empty list and look at the elements of L1 one by one.
# In each triangulation of L1 we will take only the simplices that contain the last vertex and insert those
    in L2.
# In order to pass from L2 to L3 and so on the technique will be the same, start with a empty list and
    insert the right elements from the preivous one.

 # The originals list is a link between L2 and L3 used after to recover elements of L1 from L2.
originals := { } :

for i from 1 to numelems(L1) do
# Here we reset the variable "auxi2" that will hold the set of simplices of the triangulation L1[i] to be
    inserted in L2.
auxi2 := [ ];
for l from 1 to numelems(L1[i]) do
# Here we reset the variable "auxi" that will hold (the indexes of) the simplex we are testing.
auxi := [0, 0, 0, 0];
for j from 1 to 4 do
# This line is needed because on SAGE the vertex are indexed beginning with 0 and we want that they
    start from 1.
auxi[j] := L1[i][l][j] + 1;
od :
# The next "if" makes Step (2) passing from L1 to L2 only the simplexes with the last vertex.
if auxi[4] = ColumnDimension(C) then auxi2 := [op(auxi2), auxi]; fi:
od:
# In the next line we indeed insert in L2 the set of simplices "auxi2", but only if it is not there yet.
if not(member(auxi2, L2)) then
L2 := {op(L2), auxi2};
originals := originals union {[L1[i], auxi2]} :
 fi:
od:
# Here we do step (3) of Algorithm to obtain L3 from L2
    by removing all simplices with a corresponding matrix having a zero 3 x3 minor.
# The script is pretty much the same as step (2) but with distinct test condition.

L3 := { } :
```

```
# The originals2 list is a link between L2 and L3 used after to recover elements of L2 from L3.
originals2 := { } :

for i from 1 to numelems(L2) do
auxi := [ ];
for l from 1 to numelems(L2[i]) do
# The next "if" makes Step (3) passing from L2
      to L3 only simplexes whose corresponding matrix has no zero 3 x3 minor.
if { [L2[i][l][1], L2[i][l][2], L2[i][l][3]], [L2[i][l][1], L2[i][l][2], L2[i][l][4]], [L2[i][l][1],
      L2[i][l][3], L2[i][l][4]], [L2[i][l][2], L2[i][l][3], L2[i][l][4]]}
      subset {op(validpolytopesindex)} then
auxi := [op(auxi), L2[i][l]];
 fi:
 od:
if not(member(auxi, L3)) then
L3 := L3 union {auxi};
 originals2 := originals2 union {[L2[i], auxi]} :
 fi:
od:
# Here we do step (4) of Algorithm to obtain L4 from L3 changing any index 4, 5,..., n + 3 to 1.


L4 := { } :

# The originals3 list is a link between L2 and L3 used after to recover elements of L3 from L4.

originals3 := { } :

 for i from 1 to numelems(L3) do
auxi := [ ];
for l from 1 to numelems(L3[i]) do
auxi2 := L3[i][l] ;
for j from 4 to n + 3 do
 # The next line tests if j is a index of L3[i], if it is bb receives true and pp its position.
 bb := member( j, L3[i][l],'pp');
 if bb = true then
 auxi2[pp] := 1;fi;od;
 # After changing some index to 1 we sort the list of simplices to keep it in the lexicographic order.
auxi := sort( [op(auxi), sort(auxi2) ]);
 od:
 # The next line is need because after changing some index to we can have duplicates.
if not(member(auxi, L4)) then
L4 := L4 union {auxi};
# The originals3 list is a link between L3 and L4 used after to recover elements of L3 from L4.
originals3 := originals3 union {[L3[i], auxi]} :
 fi:
od:
# Here we do step (5) of Algorithm to obtain L5 from L4.

L5 := { } :
```

```
for i from 1 to numelems(L4) do
 # If the variable "auxi" is 0 we insert L4[i] in L5, and if it is 1 we do not.
 # We reset "auxi" as 0 in the next line.
auxi := 0;
 # The next loop for will test if L4[i] is contained in any L4[j] with j >i, if it is then we set auxi:=1.
for j from i + 1 to numelems(L4) while auxi = 0 do
if  numelems( {op(L4[i])} intersect {op(L4[j])} ) = numelems(L4[i]) then
auxi := 1;
 fi;
  od:
 if auxi = 0 then
L5 := L5 union {L4[i]}
  fi;
  od:
# The following  is just a information check.

print("This list L1 is the whole list.");
print("This list L2 consider only the simplices having the origin.");
print("This list L3 takes out the simplices which the corresponding matrix has a zero 3x3 minor.");
print("This list L4 replaces indexes 4,5,...,n+3 by 1.");
    print("This list L5 takes out the triangulations T such that there is another triangulation T'
    containing T.");
print("Number of elements of L1, L2,L3, L4, and L5 are.");
nops(L1); nops(L2); nops(L3); nops(L4); nops(L5);
```

"This list L1 is the whole list."

"This list L2 consider only the simplices having the origin."

"This list L3 takes out the simplices which the corresponding matrix has a zero 3x3 minor."

"This list L4 replaces indexes 4,5,...,n+3 by 1."

"This list L5 takes out the triangulations T such that there is another triangulation T' containing T.
"

"Number of elements of L1, L2,L3, L4, and L5 are."

44

25

15

7

6 **(2)**

```
# The following counts and displays how many elements of L5 has a determinated size.
# This can be used to guess what will be a good candidate for k.

count2 := [seq(0, i = 1 ..nops(L5[nops(L5)]))] :
 for i from 1 to numelems(L5) do
count := nops(L5[i]) :
count2[count] := count2[count] + 1 :
od:
 for i from 1 to nops(count2) do
printf("There is %d  configurations with %d valid polytopes.\n", count2[i], i);
 od;
```

for J **in** $L5$ **do**
$print(J)$;
od:

```
There is 0  configurations with 1 valid polytopes.
There is 0  configurations with 2 valid polytopes.
There is 4  configurations with 3 valid polytopes.
There is 0  configurations with 4 valid polytopes.
There is 2  configurations with 5 valid polytopes.
```

$$[[1, 2, 3, 8], [1, 2, 7, 8], [1, 2, 7, 8]]$$
$$[[1, 2, 3, 8], [1, 3, 6, 8], [1, 3, 6, 8]]$$
$$[[1, 2, 6, 8], [1, 3, 6, 8], [2, 3, 6, 8]]$$
$$[[1, 2, 7, 8], [1, 3, 7, 8], [2, 3, 7, 8]]$$
$$[[1, 2, 7, 8], [1, 3, 6, 8], [1, 6, 7, 8], [2, 3, 6, 8], [2, 6, 7, 8]]$$
$$[[1, 2, 7, 8], [1, 3, 6, 8], [1, 6, 7, 8], [2, 3, 7, 8], [3, 6, 7, 8]] \quad (3)$$

 # In the following we check for each element of L5 the conditions that are needed for it to be positively decorated by Csimple.

$allsolutions := \{\}$:

for J **in** $L5$ **do**
 # We only work with J in L5 with at least $2\cdot \text{floor}\left(\frac{n}{2}\right)+1$ *simplices.*
if $numelems(J) \geq 2\cdot \text{floor}\left(\frac{n}{2}\right)+1$ **then**
$Jused := [\ ]$:
 # The variable "solutions" will have pairs [I,C].
 # Each I is a list of (indexes of) simplices.
 # The corresponding C is a list of expressions f1,..., fm such that the simplices ***in*** *I are simultaneously positively decorated* **by** *Csimple* ***if and*** *only* ***if*** *f1 > 0,.., fm > 0.*
 # Each C has at least the conditions E, F **and** *S (that is* $E > 0, F > 0, S > 0$*) because these are total concentrations of chemical species.*
 # Each C also has 1-M[1],...,1-M[n-1] sinse these M[i] must be less than 1.
 # We include in each C the expression 1 as well since the obvious condition 1 >0 will help us to eliminate bad candidates.

$solutions := \{[Jused, \{1, E, F, S, seq(1-M[i], i=0..n-1)\}]\}$:
$solutionsaux := \{\}$:

 # The next loop does the following. Start with the first element of J, if it gives viable solutions keep it and discard it otherwise.

 # Then if the second gives condities compatible with the first one keep it and discard it otherwise, and so on.

for j **in** J **do**
 # Now we compute two sets of conditions for j to be positively decorated by Csimple, conditionsnewa and conditionsnewb, these correspond to the two possibilites of the alternating signs of the four 3x3 minors.

```
for i from 1 to 4 do  det[i] := Determinant(Csimple[1 ..3, subsop(i = NULL, j)]) :  od:
conditionsnewa := { -det[1], det[2], -det[3], det[4]};
conditionsnewb := {det[1], -det[2], det[3], -det[4]};
solutionsaux := { } :

    # Next we compare conditionsnewa and conditionsnewb with the previous conditions. We include
    only one of them, if there is a compatible one.
for l in solutions do
 Jused := l[1]; conditions := l[2];
if  evalb(numelems(conditions intersect conditionsnewa) ≥ 1 and numelems(conditions
    intersect conditionsnewb) ≥ 1) = true then
solutionsaux := solutionsaux union {[Jused, conditions]};
fi:
if  evalb(numelems(conditions intersect conditionsnewb) = 0) = true then
solutionsaux := solutionsaux union {[[op(Jused), j], conditions union  conditionsnewa]};
fi:
if evalb(numelems(conditions intersect conditionsnewa) = 0) = true then
 solutionsaux := solutionsaux union {[[op(Jused), j], conditions union  conditionsnewb]};
fi:
od:

solutions := solutionsaux;
 od:

 # Finally, in the variable "allsolutions" we keep the candidates that give at least k=2·floor(n/2)
    + 1 regions.

for k in solutions do
if numelems(k[1]) ≥ 2· floor(n/2) + 1 then allsolutions := allsolutions union {k}; fi:
od:

 fi:
 od:

printf("Number of solutions to try: %d.", numelems(allsolutions));
Number of solutions to try: 3.
# In this part we obtain L7 from "allsolutions".
# We do this searching in "allsolutions" for the elements for which there are viable parameters
    satisfying the conditions.
# This is the only numerical part of the whole script.
# In the end each J in L7 will contain:
# J[1] = list of simplexes;
# J[2]= corresponding conditions;
# J[3]= a list of real numbers which are viable values for the parameters.

interface(displayprecision = 6) : L7 := { } :

for j in allsolutions do
conditions := j[2] :
```

```
Jused := j[1]:
# The next command "Minimize" is used to find a numerical solution for the condition.
# If the "Minimize" is able to find one solution then the conditions are viable and are included in L7.
# Since "Minimize" works only with closed conditions we use ≥ 1/10000 instead of > 0.
# If "Minimize" is unable to find a solution it returns a error, because of that we need the "try"
    command. In this case the conditions are discarded.
try
Min := Minimize(1, {seq(conditions[j] ≥ 1/10000, j = 1 ..numelems(conditions))}, assume
    = nonnegative, iterationlimit = 100) :
L7 := L7 union {[j[1], j[2], Min[2]]} :
catch:
end try:
end do:


    # The next loop for is used to remove the conditions 1>0,E>0,...,1-M[i]>0,M[i]>0 from the
    elements of L7.
solutionsaux := { } :
for k from 1 to numelems(L7) do
solutionsaux := solutionsaux union {[L7[k][1], L7[k][2] minus {1, E, F, S, seq(1 − M[i], i = 0 ..n
    − 1), seq(M[i], i = 0 ..n − 1)}, L7[k][3]]} :
od:
L7 := solutionsaux :


    # The next two loops are used to remove a set of conditions C if it is contained in another. In this
    way we get only maximal regions.
solutionsaux := { } :
for k from 1 to numelems(L7) do
aux := 0 :
for j from k + 1 to numelems(L7) do
if evalb(L7[k][2] subset L7[j][2]) then
aux := 1
fi:
od:
if aux = 0 then
solutionsaux := solutionsaux union {[L7[k][1], L7[k][2], L7[k][3]]} :
fi:
od:
L7 := solutionsaux :


printf("There are %d maximal regions, in which there are %d positive solutions each. \n",
    numelems(L7), 2· floor(n/2) + 1);
```

```
printf("The original triangulations, simplices positively decorated, regions, and a point on each one
    are:");


for i from 1 to numelems( L7) do
# This line recovers the original triangulations from the final sets obtained.
Foundoriginaltriang(originals, Foundoriginaltriang(originals2, Foundoriginaltriang(originals3,
    L7[i][1])));
L7[i][1];
L7[i][2];
L7[i][3];
od;
```

```
There are 1 maximal regions, in which there are 3 positive
solutions each.
The original triangulations, simplices positively decorated,
regions, and a point on each one are:
```

$$[[0, 2, 5, 7], [0, 4, 6, 7], [0, 5, 6, 7], [1, 2, 6, 7], [1, 4, 6, 7], [2, 5, 6, 7]]$$

$$[[1, 2, 7, 8], [1, 3, 6, 8], [1, 6, 7, 8]]$$

$$\{M_0 - M_1, S M_0 - E, E M_0 + F M_0 - E, -E M_1 - F M_1 + E, -S M_1 - F + S, -E M_0 + E M_1 - F M_0 + F M_1 + S M_0 - S M_1\}$$

$$[E = 0.399860, F = 0.000234, S = 2.200280, M_0 = 0.999708, M_1 = 0.999166] \quad \textbf{(4)}$$