Metabolic Modelling, Spring 2009, Exercises
24.3.2009
Solutions

Markus Heinonen

1. Building the stoichiometric matrix.

S =

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

```
>> v = [1  1  0 -1 -1  0  1  1]';
>> S*v

ans =

     0
     0
     1
     1
     0
    -3
     1
     0
     0
     1
    -1
    -1
```

Thus, metabolites 1, 2, 5, 8 and 9 are staying in constant concentration, while 3, 4, 7 and 10 are accumulating. Metabolites 6, 11 and 12 are diminishing.

2. Obtaining the adjacency matrix of the reaction and substrate graph. By definition, reaction graph contains as nodes the reactions, and an edge exists between two nodes if they share a common metabolite anywhere in their reactions. Substrate graph is the dual of this.

Listing 1: adjreactiongraph.m

```
function R = adjreactiongraph(S)
%ADJREACTIONGRAPH

Sabs = abs(S);
Q = Sabs' * Sabs;
R = Q > 0;
```

Listing 2: adjsubstrategraph.m

```
function R = adjsubstrategraph(S)
%ADJREACTIONGRAPH

Sabs = abs(S);
Q = Sabs * Sabs';
R = Q > 0;
```

This works because the `Sabs' * Sabs` takes the inner product of two columns, instead of the usual "row times column" multiplication. Thus, it counts the number of ones occurring on the same row, which means sharing a metabolite.

3. Graph statistics.

Listing 3: degreedist.m

```
function H = degreedist(R)
%DEGREEDIST

% takes the sum of each row of adjacency matrix (= number of neighbours)
% takes the histogram of those
H = histc(sum(R), 1:max(sum(R)));
```

Listing 4: pathdist.m

```
function H = pathdist(R)
%PATHDIST

% shortest paths between all nodes
P = graphallshortestpaths(sparse(R));

% sum the histogram matrix
H = sum(histc(P, 0:max(max(P)),2)';
```

4. Styer generic model.

You can load the file using `load()` (only if extra stuff has been edited out), or using the Import Data menu item. The degree distribution or path length distributions are not logarithmic.

```
>> Rrg = adjreactiongraph(S);
>> Rsg = adjsubstrategraph(S);
>> Hrg = degreedist(Rrg);
>> Hsg = degreedist(Rsg);

%% pad extra zeros to get compatible vector sizes
>> Hcombined = [Hrg; Hsg, zeros(1,length(Hrg)-length(Hsg))]';
>> bar(Hcombined)

>> Hrg = pathdist(Rrg);
>> Hsg = pathdist(Rsg);

%% pad extra zeros to get compatible vector sizes
>> plot(1:8, [Hrg,0], 1:8,Hsg)
>> loglog(1:8, [Hrg,0], 1:8,Hsg)
```

5. Scale free model fitting to $P(k) \approx k^{-gamma}$.

   First you need to make a `modelerror` function, which computes the sum of squared errors between your data and the power law distribution with parameter γ. Then you can use `fminsearch` to find the minimum of this function, i.e. get the γ, which gets closest fit to the data.

Listing 5: modelerror.m

```
function sse = modelerror(gamma, k, y)
%MODELERROR
%
% gamma = the parameter to be learned
% k = node degree numbers (1 2 3 4 5 ... kmax)
% y = observed degrees etc. (9 5 9 2 7 2 2 1 ... )
%
% sse = sum of squared errors

% normalize observed degrees to a distribution
ycurve = y / sum(y);

% compute the curve by gamma-parameter
curve = k .^ -gamma;
% compute the error between scale free model and observations
error = curve - ycurve;
% take the squared sum of the error
sse = sum(error.^2);
```

Listing 6: fitcurve.m

```
function [opt, sse] = fitcurve(H)
```

```
degs = 1:length(H)
model = @modelerror;
start_point = rand(1,1);

[opt, sse] = fminsearch(model, start_point, [], degs, H);

ycurve = H / sum(H);
gammacurve = degs .^ -opt;
gammacurve = gammacurve / sum(gammacurve);

plot(degs, gammacurve, degs, ycurve);
```