Variational Kinetics

Author: Masoud Ahookhosh, Systems Biochemistry Group, Luxembourg Centre for Systems Biomedicine

Reviewers: Ronan Fleming, Sylvain Arreckx

INTRODUCTION

During this tutorial, you will learn how to investigate steady states and moiety conserved steady states of biochemical reaction systems [1].

Let us consider a biochemical network with m molecular species and n reversiable elementary reactions. We define forward and reverse stoichiometry matrices $(F, R \in Z_+^{m\times n})$ respectively, where F_{ij} denotes the stoichiometry of the i^{th} molecular species in the j^{th} forward reaction and R_{ij} denotes the the stoichiometry of the i^{th} molecular species in the j^{th} reverse reaction. We assume that every reaction conservs mass, that is, there exists at least one positive vector $l \in R_{++}^m$ satisfying $(R-F)^T l = 0$. The matrix $N = R - F$ represents net reaction stoichiometry and may viewed as the incidence matrix of a directed hypergraph. We are assume that there are less molecular species than there are net reactions, i.e., $m < n$. We assume the cardinality of each row of F and R is at least one and the cardinality of each column of N is at least two. We also assume that $\operatorname{rank}([F, R]) = m$, which is a requirement for kinetic consistency.

Steady state nonlinear system

Let $c \in R_{++}^m$ denote a variable vector of species concentrations. Assuming constant ningegative elementary kinetic parameters $k_f, k_r \in R_+^n$, we assume elementary reaction kinetics for forward and reverse elementary reaction rates as $s(k_f, c) := \exp(\ln(k_f) + F^T \ln(c))$ and $r(k_r, c) := \exp(\ln(k_r) + R^T \ln(c))$, respectively, where $\exp(\cdot)$ and $\ln(\cdot)$ denote the respective componentwise functions. Then, the deterministic dynamical equation for time evolution of molecular species concentration is given

$$\frac{dc}{dt} \equiv N(s(k_f, c) - r(k_r, c)) = N\left(\exp(\ln(k_f) + F^T \ln(c)\right) - \exp\left(\ln(k_r) + R^T \ln(c))\right) = f(x$$

where $\left[\cdot, \cdot\right]$ stands for horizental cancatenation operator. A vector c^* is a steady state if and only if it satisfies $f(c^*) = 0$, leading to the nonlinear system

$$f(x) = 0.$$

Moiety conserved steady state nonlinear system

Let us emphasize that a vector c^* is a steady state of the biochemical system if and only if

$$s(k_f, c^*) - r(k_r, c^*) \in \mathcal{N}(N),$$

where $\mathcal{N}(N)$ denotes the nul space of N. THerefore, the set of steady states $\Omega = \{c \in R^m_{++}, f(c) = 0\}$ is unchanged if we replace the matrix N with $N^{\cdot}$ with the same kernel. Suppose that $N^{\overline{\in}} Z^{r\times n}$ is the submatrix of N whose rows are linearly independent, then $\mathrm{rank}(N^{\overline{)}} = \mathrm{rank}(N) := r$. If one replaces N by $N^{\cdot}$ and considers the logarithmic scale, by setting $x = \ln(c) \in R^m$ and $k = [\ln(k_f)^T, \ln(k_r)^T]^T \in R^{2n}$, then we have

$$\overline{f}(x) := [N, \overline{-N}^{\overline{]}} \exp\left(k + [F, R]^T x\right).$$

Let also $L \in R^{m-r,m}$ denote a basis for the left nullspace of N, which implies $N^T L = 0$. We also have $\mathrm{rank}(L) = m - r$. WE say that the system satisfies moiety conservation if for any initial concentration $c_0 \in R^m_{++}$, it holds

$$L\,c = L\exp(x) = l_0,$$

where $l_0 \in R^m_{++}$. Therefore, the problem of finding the moiety conserved steady state of a biochemical reaction network is equivalent to solving the nonlinear system of equations

$$h(x) := \begin{pmatrix} \overline{f}(x) \\ L\exp(x) - l_0 \end{pmatrix} = 0.$$

We introduce an interface to software that enables the computation of the elementary modes, or extreme pathways, given a network and user-defined reaction bounds.

METHODOLOGY

In order to solve above-mentioned nonlinear system, we here address three classes of methods, i.e.,

1. Levenberg-Marquardt methods [1,2,5,6,9.10],
2. DC programming methods [3],
3. derivative-free methods for duplomonotone mappings [4],

where each class of solvers are described shortly as follows:

1. The Levenberg-Marquardt methods are standard techniques used to solve nonlinear systems that each of which is a combination of the gradient descent and the Gauss-Newton methods. Therefore, knowing the first-order information (function values and gradients) of the mapping is enough to proceed the algorithm. We here consider two classes of Levenberg-Marquardt methods, namely locally convergent Levenberg-Marquardt methods (LLM_YF, LLM_FY, LLM_F, LLM) and globally convergent Levenberg-Marquardt methods (GLM_YF, GLM_FY, LevMar, LMLS, LMTR), see [1] and [2], respectively.
2. In DC programming methods, one needs to rewrite the nonlinear system as a minimization of a difference of two convex function. Then, the DC subproblem is the minimization of a convex function, which is constructed by keeping the first function and linearlizing the second function. We here address two DC programming methods, namely, DCA and BDCA, see [3] for detailed information.

3. Derivative-free methods are a class of methods that only needs function values to minimize a nonlinear least-squares problem. We here consider three derivative-free methods, namely, BDF, CSDF, and DBDF, see Algorithms 1-3 in [4] for more details.

MATERIALS

- *Please ensure that the COBRA Toolbox has been properly installed and initialised.*

PROCEDURE

Computing steady states of biochemical systems

The mandatory inputs for computing steady states are a model involving F and R, the name of a solver to solve the nonlinear system, an initial point x_0, and parameters for the considered solvers. We first need to load data from a ".mat" file involve F, R, and kin (kinetic vector). For example, for "Ecoli core" model, we have

```
global CBTDIR
load([CBTDIR filesep 'tutorials' filesep 'variationalKinetics' filesep 'Ecoli_core_data.mat'])
```

Then, we need to make a struture "model" by

```
model.F = F;
model.R = R;
```

and specify a solver by

```
solver = 'LMTR';
```

and detemine the parameters for the selected algorithm

```
parms.MaxNumIter = 1000;
parms.adaptive = 1;
parms.kin = kin;
```

otherwise, the selected algorithm will be run by the default parameters assigned in the codes. We finally need to run the function "optimizeVKmodels.m" like

```
output = optimizeVKmodels(model, solver, x0, parms);
```

```
Running LMTR ...
Status = Maximum number of iterations is reached.
```

Let us emphasize that all the slovers (LLM_YF, LLM_FY, LLM_F, LLM,GLM_YF, GLM_FY, LevMar, LMLS, LMTR, DCA, BDCA, BDF, CSDF, DBDF) can be used to find steady states of biochemical systems; however, based on our experiments, "LMTR" and "LMLS" perform better than the others.

If you are not familiar with the solvers, we may suggest to use solvers with the dafault values for parameters.

Computing moiety conserved steady state of biochemical systems

The mandatory inputs for computing moiety conserved steady states are a model involving F, R, L, and l_0, the name of a solver to solve the nonlinear system, an initial point x_0, and parameters for the considered solvers. We first need to load data from a ".mat" file involve F, R, L, l_0, and kin (kinetic vector). For example, for "Ecoli core" model, we have

Then, we need to make a struture "model" by

```
model.F = F;
model.R = R;
model.L = L;
model.l0 = l0;
```

and specify a solver by

```
solver = 'LMLS';
```

and detemine the parameters for the selected algorithm

```
parms.MaxNumIter = 1000;
parms.adaptive = 1;
parms.kin = kin;
```

otherwise, the selected algorithm will be run by the default parameters assigned in the codes. We finally need to run the function "optimizeVKmodels.m" like

```
output = optimizeVKmodels(model, solver, x0, parms);
```

```
Running LMLS ...
Status = a solution is found.
```

Let us emphasize that among above-mentioned solvers one can use the Levenberg-Marquardt slovers (LLM_YF, LLM_FY, LLM_F, LLM,GLM_YF, GLM_FY, LevMar, LMLS, LMTR) to find moiety conserved steady states of biochemical systems; however, based on our experiments, "LMTR" and "LMLS" perform better than the others. If you are not familiar with the solvers, we may suggest to use solvers with the dafault values for parameters.

Optional inputs

The function can have some optional inputs for slover, x_0, and the parameters corresponding to the selected solver. Therefore, we here explain the most important optional inputs in the following with respect to the selected solver.

Parameters for all solvers:

- `MaxNumIter`: is the maximum number of iterations;

- MaxNumMapEval: is the maximum number of function evaluations;
- MaxNumGmapEval: is the maximum number of gradient evaluations;
- TimeLimit: is the maximum time limit;
- epsilon: is the accuracy parameter;
- kin: is a kinetic parameter in R^(2n);
- x_opt: is the optimizer (if available);
- psi_opt: is the optimum (if available);
- flag_x_error: is a flag to specify if the relative error of iteration points is needed (1) or not (0);
- flag_psi_error: is a flag to specify if the relative error of merit function is needed (1) or not (0);
- flag_time: is a flag to specify if saving time in each iteration is needed (1) or not (0);

Parameters for Levenberg-Marquardt solvers:

- `solver`: is one of the solver;

1. LLM_YF: the locally convergent Levenberg-Marquardt method of Yamashita and Fukushima [10];
2. LLM_FY: the locally convergent Levenberg-Marquardt method of Fan and Yuan [5];
3. LLM_F: the locally convergent Levenberg-Marquardt method of Fischer [6];
4. LLM: the locally convergent Levenberg-Marquardt method of Ahookhosh, Artacho, Fleming, and Phan [1];
5. GLM_YF: the globally convergent Levenberg-Marquardt method of Yamashita and Fukushima [10];
6. GLM_FY: the globally convergent Levenberg-Marquardt method of Fan and Yuan [5];
7. LevMar: the globally convergent Levenberg-Marquardt method of Ipsen, Kelley, and Pope [9];
8. LMLS: the globally convergent Levenberg-Marquardt method of Ahookhosh, Artacho, Fleming, and Phan [2];
9. LMTR: the globally convergent Levenberg-Marquardt method of Ahookhosh, Artacho, Fleming, and Phan [2];

- `adaptive:` is a flag to specify lambda should be updated adaptively (1) or not (0);
- eta: is a constant for Levenberg-Marquardt parameter;
- Stopping_Crit: is a stopping criterion;

1. 1: stop if the norm of gradients is less or equal than epsilon;
2. 2: stop if the norm of rhe mapping is less or equal than epsilon;
3. 3: stop if maximum number of iterations is reached;
4. 4: stop if maximum number of function evaluations is reached;
5. 5: stop if maximum number of gradient evaluations is reached;
6. 6: stop if time limit is reached;
7. 7: stop if ||grad||<=max(epsilon,epsilon^2*ngradx0)
8. 8: stop if ||nhxk||<=max(epsilon,epsilon^2*nhx0)
9. 9: stop if ||hxk||<=epsilon or maximum number of iterations is reached

Parameters for DC programming solvers:

- `solver`: is one of the solver;

1. DCA: DC programming algorithm of Artacho, Fleming, and Phan (Algorithm 1) [3];
2. BDCA: DC programming algorithm of Artacho, Fleming, and Phan (Algorithm 2 and 3) [3];

- alpha: is a constant for the line search;
- beta: is the backtarcking constant;
- lambda_bar: starting step-size for the line search;

- rho : is the strong convexity parameter;
- flag_line_search: is a flag determines either "Armijo" or "Quadratic_interpolation" should be used;
- Stopping_Crit: is a stopping criterion;

1. 1: stop if the norm of rhe mapping is less or equal than epsilon;
2. 2: stop if maximum number of iterations is reached;
3. 3: stop if maximum number of function evaluations is reached;
4. 4: stop if time limit is reached;
5. 5: stop if ||fxk||<=epsilon or maximum number of iterations is reached

Parameters for Derivative-free solvers:

- `solver`: is one of the solver;

1. BDF: backtracking derivative-free algorithm of Artacho and Fleming [4];
2. CSDF: constant step derivative-free algorithm of Artacho and Fleming [4];
3. DBDF: double backtracking derivative-free algorithm of Artacho and Fleming [4];

- alpha: is a constant with alpha < 2 sigma;
- beta: is the backtarcking constant;
- lambda_min: lower bound of the step-size;
- lambda_max: upper bound of the step-size;
- Stopping_Crit: is a stopping criterion;

1. 1: stop if the norm of rhe mapping is less or equal than epsilon;
2. 2: stop if maximum number of iterations is reached;
3. 3: stop if maximum number of function evaluations is reached;
4. 4: stop if time limit is reached;
5. 5: stop if ||fxk||<=epsilon or maximum number of iterations is reached;

For a complete list of optional inputs and their definition, you can also run the following command.

```
help optimizeVKmodels
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%% optimizeVKmodels.m %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 optimizeVKmodels is a function for finding a solution of the nonlinear
 system
                h(x) = f(x) = 0, x in R^m,                (I)
 or
                h(x) = (f(x)^T,l(x)^T)^T = 0, x in R^m,   (II)
 where
                f(x) = [F-R, R-F]*exp(k+[F,R]'*x),
                l(x) = L*exp(x)-l0,
 using the nonlinear unconstrained minimization
                       min  psi(x) = 1/2 ||h(x)||^2
                       s.t. x in R^m.

 with several solvers. For (I), one can use all the following solvers;
 however, for (II), one can only apply local solvers LLM_F, LLM_FY,
 LLM_YF, LLM and global solvers GLM_FY, GLM_YF, LevMar, LMLS, and LMTR.
 If sonver fiels is empty, the code will use LMTR as the default.

 INPUT:
```

```
model                % stracture includes F, R and/or L
   .F                % forward stoichiometric matrix
   .R                % reverse stoichiometric matrix
   .L                % basis for the left null-space of N = R-F
   .l0               % constant l0

solver
   Local Levenberg-Marquardt (LM) solvers:
   LLM               % LM of Ahookhosh et al.
   LLM_F             % LM of Fischer
   LLM_FY            % LM of Fun and Yuan
   LLM_YF            % LM of Yamashita and Fukushima
   Global Levenberg-Marquardt (LM) solvers:
   LM_FY             % LM of Fun and Yuan
   LM_YF             % LM of Yamashita and Fukushima
   LevMar            % LM of Kelly
   LMLS              % LM line search of Ahookhosh et al.
   LMTR              % LM trust-region of Ahookhosh et al.
   DC programming solvers:
   DCA               % DC programming algorithm
   BDCA              % boosted DC programming algorithm
   Duplomonotone derivative-free solvers:
   BDF               % Derivative-free duplomonotone method 1
   CSDF              % Derivative-free duplomonotone method 2
   DBDF              % Derivative-free duplomonotone method 3

Parameters for LLM_F, LLM_FY, LLM_YF, LLM, GLM_FY, GLM_YF, LevMar,
               LMLS, LMTR
parms                % structure including the parameteres of schemes

  .eta               % constant for Levenberg-Marquardt parameter
  .MaxNumIter        % maximum number of iterations
  .MaxNumMapEval     % maximum number of function evaluations
  .MaxNumGmapEval    % maximum number of gradient evaluations
  .TimeLimit         % maximum running time
  .epsilon           % accuracy parameter
  .x_opt             % optimizer
  .psi_opt           % optimum
  .adaptive          % update lambda adaptively
  .kin               % kinetic parameter in R^(2n)
  .flag_x_error      % 1 : saves x_error
                     % 0 : do not saves x_error (default)
  .flag_psi_error    % 1 : saves psi_error
                     % 0 : do not saves psi_error (default)
  .flag_time         % 1 : saves psi_error
                     % 0 : do not saves psi_error (default)
  .Stopping_Crit     % stopping criterion

                     % 1 : stop if ||grad|| <= epsilon
                     % 2 : stop if ||nhxk|| <= epsilon
                     % 3 : stop if MaxNumIter is reached
                     % 4 : stop if MaxNumMapEval is reached
                     % 5 : stop if MaxNumGmapEval is reached
                     % 6 : stop if TimeLimit is reached
                     % 7 : stop if
                              ||grad||<=max(epsilon,epsilon^2*ngradx0)
                     % 8 : stop if
                              ||nhxk||<=max(epsilon,epsilon^2*nhx0)
                     % 9 : stop if                          (default)
                           ||hxk||<=epsilon or MaxNumIter is reached

Parameters for DCA, BDCA
parms                % structure including the parameteres of schemes

mapp                 % function handle provides f(x) and gradient f(x)
x0                   % initial point
options              % structure including the parameteres of scheme
```

```
  .MaxNumIter        % maximum number of iterations
  .MaxNumMapEval     % maximum number of function evaluations
  .TimeLimit         % maximum running time
  .epsilon           % accuracy parameter
  .x_opt             % optimizer
  .psi_opt           % optimum
  .alpha             % constant for the line search
  .beta              % backtarcking constant
  .lambda_bar        % starting step-size for the line search
  .rho               % strong convexity parameter
  .kin               % kinetic parameter in R^(2n)
  .flag_line_search  % "Armijo" or "Quadratic_interpolation"
  .flag_x_error      % 1 : saves x_error
                     % 0 : do not saves x_error (default)
  .flag_psi_error    % 1 : saves psi_error
                     % 0 : do not saves psi_error (default)
  .flag_time         % 1 : saves psi_error
                     % 0 : do not saves psi_error (default)
  .Stopping_Crit     % stopping criterion

                     % 1 : stop if ||nfxk|| <= epsilon
                     % 2 : stop if MaxNumIter is reached
                     % 3 : stop if MaxNumMapEval is reached
                     % 4 : stop if TimeLimit is reached
                     % 5 : stop if                          (default)
                           ||hxk||<=epsilon or MaxNumIter is reached

Parameters for BDF, CSDF, DBDF
parms                % structure including the parameteres of schemes

mapp                 % function handle provides f(x) and gradient f(x)
x0                   % initial point
options              % structure including the parameteres of scheme

  .MaxNumIter        % maximum number of iterations
  .MaxNumMapEval     % maximum number of function evaluations
  .TimeLimit         % maximum running time
  .epsilon           % accuracy parameter
  .x_opt             % optimizer
  .psi_opt           % optimum
  .alpha             % constant with alpha < 2 sigma
  .beta              % is the backtarcking constant;
  .lambda_min        % lower bound of the step-size
  .lambda_max        % upper bound of the step-size
  .flag_x_error      % 1 : saves x_error
                     % 0 : do not saves x_error (default)
  .flag_psi_error    % 1 : saves psi_error
                     % 0 : do not saves psi_error (default)
  .flag_time         % 1 : saves psi_error
                     % 0 : do not saves psi_error (default)
  .Stopping_Crit     % stopping criterion

                     % 1 : stop if ||nfxk|| <= epsilon
                     % 2 : stop if MaxNumIter is reached
                     % 3 : stop if MaxNumMapEval is reached
                     % 4 : stop if TimeLimit is reached
                     % 5 : stop if                          (default)
                           ||hxk||<=epsilon or MaxNumIter is reached

OUTPUT:

output               % structure including more output information

  .x_best            % the best approximation of the optimizer
  .psi_best          % the best approximation of the optimum
  .T                 % running time
  .Niter             % total number of iterations
```

```
    .Nmap             % total number of mapping evaluations
    .Ngmap            % total number of mapping gradient evaluations
    .merit_func       % array including all merit function  values
    .x_error          % relative error norm(xk(:)-x_opt(:))/norm(x_opt)
    .psi_error        % relative error (psik-psi_opt)/(psi0-psi_opt))
    .Status           % reason of termination
    .Time             % running time of all iterations

  Author:

  Masoud Ahookhosh     % Systems Biochemistry Group, LCSB,
                         University of Luxembourg.


  LAST UPDATE:

  June 2017

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Output

The output of `optimizeVKmodels.m` is a structure "output" involving the fields

- x_best: is the best approximation of the optimizer;
- psi_best: is the best approximation of the optimum;
- T: is the running time;
- Niter: is the total number of iterations;
- Nmap: is total number of mapping evaluations;
- Ngmap: is total number of mapping gradient evaluations (if gradient used in the algorithm);
- merit_func: is an array including all merit function values;
- x_error: is the relative error of iteration points;
- psi_error: is the relative error of merit function;
- Time: is the running time of all iterations;
- Status: is the reason of termination;

TIMING

Running, the code is dependent on the size of models and the solver selected, which may take long from less than 1 second to few hours.

ANTICIPATED RESULTS

Finding steady states or moiety conserved steady state with one of the above-mentioned solvers (e.g., solver = 'LMTR') leads to the following results.

```
output = optimizeVKmodels(model, solver, x0, parms)
```

```
Running LMLS ...
Status = a solution is found.
output =
```

```
    x_best: [72×1 double]
  psi_best: 4.0949736181951e-11
         T: 0.133948
merit_func: [269×1 double]
     Niter: 269
      Nmap: 537
     Ngmap: 269
    Status: 'a solution is found.'
```

TROUBLESHOOTING

In order to compute moiety conserved steady states, one should not use DC programming algorithms (DCA and BDCA) or derivative-free algorithms (BDF, CSDF, DBDF) because the current version of these codes are designed to deal with steady state of biochemical systems.

REFERENCES

[1] Ahookhosh, M., Artacho, F.J.R., Fleming, R.M.T., Phan V.T., Local convergence of Levenberg-Marquardt methods under Holder metric subregularity, Submitted, (2017).

[2] Ahookhosh, M., Artacho, F.J.R., Fleming, R.M.T., Phan V.T., Global convergence of Levenberg-Marquardt methods under Holder metric subregularity, Submitted, (2017).

[3] Artacho, F.J.R., Fleming, R.M.T., Phan V.T., Accelerating the DC algorithm for smooth functions, Mathematical Programming, (2017).

[4] Artacho, F.J.R., Fleming, R.M.T., Globally convergent algorithms for finding zeros of duplomonotone mappings, Optimization Letters, 9(3), 569--584 (2015).

[5] Fan, J., Yuan, Y., On the quadratic convergence of the Levenberg-Marquardt method without nonsingularity assumption, Computing, 74(1), 23--39 (2005).

[6] Fischer, A., Local behavior of an iterative framework for generalized equations with nonisolated solutions, Mathematical Programming, 94(1), 91--124 (2002).

[7] Fleming, R.M.T., Thiele, I., Mass conserved elementary kinetics is sufficient for the existence of a non-equilibrium steady state concentration, Journal of Theoretical Biology, 314, 173--181 (2012).

[8] Fleming, R.M.T., Vlassis, N., Thiele, I., Saunders, M.A., Conditions for duality between fluxes and concentrations in biochemical networks, Journal of Theoretical Biology, 409, 1--10 (2016).

[9] Ipsen, I., Kelley, C., and Pope, S., Rank-deficient nonlinear least squares problems and subset selection, SIAM Journal on Numerical Analysis, 49, 3, 1244--1266 (2011).

[10] Yamashita, N., Fukushima, M., On the rate of convergence of the Levenberg-Marquardt method, In: G. Alefeld, X. Chen (eds.) Topics in Numerical Analysis, vol. 15, pp. 239--249, Springer Vienna, Vienna (2001).