ODE parameter estimator

Benjamin Andre
benjamin.andre@colorado.edu

Id: odeParamEst.tex 5 2008-11-18 04:11:57Z bjandre

This software and document are still a draft... Use at your own risk.

1 Introduction

The purpose of this software is to help estimate the parameters ($\vec{p}$) of a system of nonlinear ODEs and initial conditions

$$\frac{d\vec{y}}{dt} = f(t,\vec{y},\vec{p}) \tag{1}$$

$$\vec{y}(t_0) = \vec{c} \tag{2}$$

by fitting them to integrated experimental data ($\vec{y_e}$)

$$\vec{y_e}(t_0),\vec{y_e}(t_1),...,\vec{y_e}(t_N) \tag{3}$$

using the built-in Matlab functions ode45 and lsqnonlin. The basic algorithm is: use an estimate of the parameters to numerically integrate the ode over the range of the independent variable found in the data. The dependent variables are saved at times corresponding to the experimental data. Nonlinear least squares regression is used to refine the estimate of the parameters.

Numerical models, regression etc are only a tool. Don't blindly trust the results of this model. Use additional knowledge about your experiment or field that may not be captured in the data/model to judge the quality of the fit.

2 Use

2.1 Requirements

- access to a computer running Matlab v. 2007a or later with optimization toolbox?.
- some_name.csv : input data
- some_ode_function.m : function to evaluate the ode.
- some_driver.m : driver routine

2.2 Data File

The integrated data points are supplied in a csv file (comma separated values), exported from a spreadsheet or created by hand. The delimiters must be commas. There can be no comments, column headings, etc in the file. The columns are:

1 : independent variable, time, etc
2 : dependent variable 1 data
3 : dependent variable 1 weighting
4, 5, ... N, N+1 : additional columns alternating data and weighting,....

After the data and weighting, the user is allowed to supply additional columns which are stored and passed into the ode function (via the "userData" variable). See the pyrite example getvolume.m for details about how this may be used.

2.3 ODE function

The ode function is the same as described by the Matlab ode solver documentation. It should take two input variables, the current independent variable and a vector of dependent variables. It returns a single vector of xdot data. An example ode function would be saved in exODEfunc.m:

```
function xdot=exODEfunc(t,x)
global OPEuser

% params(1) = some constant
% params(2) = some other constant
a = OPEuser.params(1);
b = OPEuser.params(2);
xdot = zeros(2,1);
xdot(1) = b - a * x(1) * x(2));
xdot(2) = b * x(2);
```

The ode function file should have a single global variable, "OPEuser". OPEuser is a structure supplied by the odeParamEst routine that provides the following fields:

- OPEuser.params : the current estimate of the parameters
- OPEuser.time independent variable data supplied by the user
- OPEuser.expData : a copy of the experimental data is available if needed
- OPEuser.userData : user time series data extracted from the data file
- OPEuser.userParams : user defined parameters. passed through from the driver routine (OPEinput) unmodified.

Do not modify the params, time or expData fields inside the ode function!

2.4 Driver routine

The drive routine sets up the problem, calls "odeParamEst" to fit the parameters then plots and saves the results. Setup consists of specifying the path to the data file, the function handle, the number of dependent variables in the datafile/ode and the initial guess for the parameters. An example driver routine is:

```
odeOptions = odeset('Stats','on');
odeOptions = odeset(odeOptions,'MaxStep',50);

% fill the ode parameter estimator input data structure
OPEinput.numEqns = 2;
```

```
OPEinput.odeFunc = @exODEfunc;
OPEinput.dataFile = 'exData.cvs';
OPEinput.initialParams = [-0.4; 10^-6.07];
OPEinput.odeOptions = odeOptions;
OPEinput.verifyODE = true;

[params, time, plotData] = odeParamEst(OPEinput);
% plot or save data...
```

odeParamEst requires one input variable containing the user data. The user passes data from the driver program into the parameter estimator by using a data structure called "OPEinput". The parameter estimator passes data to the user ode function using the "OPEuser" data structure described above. The OPEinput structure may be modified internally by odeParamEst routine and should not be accessed by the user in the ode function routine (so don't make it global and try to use it).

2.4.1 OPEinput: required fields

- OPEinput.dataFile : the path to the data file (described above).
- OPEinput.numEqns : the number of dependent variables in the system of equations
- OPEinput.odeFunc : the user defined ode function (described above)
- OPEinput.initialParams : a meaningful initial guess for the parameters. All zeros or ones is not a meaningful guess. the order of the parameters here is the same as in the ode function, so make sure you are consistent.

2.4.2 OPEinput: optional fields

The following fields in OPEinput are optional fields that will be assigned a default value if not specified.

- OPEinput.initialConditions : must have the same number of unknowns as the data (without weights). Default uses the first row of the data
- OPEinput.odeOptions : matlab data created by the odeset() function containing any options you want to pass to the ode solver, i.e. setting the maximum step size, tolerances, etc. Before you start messing with the default variables, make sure your ode function is correct and the initial guess for the parameters is meaningful. see Matlab help for ode45 or odeset for details. By default, stats are turned on.
- OPEinput.minFunc : the minimization function used by lsqnonlin. If the user does not specify a function, the default is the function odeParamEstLSQ.m, which performs a weighted residuals calculation. Creating your own function is the only way there is currently to use a different ode solver. This is a slightly more advanced option and should only be used if the default is not working for you or you know what you are doing.

The following fields are optional and are ignored if not specified by the user.

- OPEinput.lowerBounds and OPEinput.upperBounds : these specify upper and lower bounds to the parameters (i.e. non negative, etc). If one is defined, then both must be defined, and their size must match the size of OPEinput.initialParams.
- OPEinput.minOptions : options to pass the minimization solver lsqnonlin. This is only available if upper and lower bounds are specified. See Matlab lsqnonlin documentation for details.
- OPEinput.userParams : user specified parameters. can be a vector, structure, etc. It is copied to OPEuser.userParams unmodified.

- OPEinput.verifyODE : true or false, flag that asks odeParamEst to plot the ode function over the data range specified in the input file, using the initial parameter guess. This allows the user to verify that the ode function is correct and that the initial parameters are valid. It is highly recommend that you use this while developing and debugging your code!

2.4.3 Return values

The odeParamEst function returns three variables,

- params : the final parameter estimates, in the order used by the ode function.
- time : the time (independent variable) column data extracted from the data file
- plotData : input data and resulting fit, (d1, f1, d2, f2, etc) it a way that can easily be plotted.

2.5 comparison plot

description of example plot function....

3 Usage help and tips

- Checked the user defined functions to make sure they are behaving correctly. Most of the problems you will encounter getting the solver working and getting meaningful results are because of an ODE function that isn't doing what you think it is. Try plotting the ode function with a known set of parameters (it is easy with "OPEinput.verifyODE = true")?
- Did you check the ode function? Don't you think you should try that? Why don't you do it now, I'll wait. I don't care if you think you've done it already, do it again. No seriously, do it.
- Are you using a meaningful set on initial conditions? i.e. does your data have a noisy initial data point? This is an IVP, if you don't have a good starting point, it won't work.
- Are you using a meaningful set of initial parameter values?
- did you check your units? Are you sure?
- This code is solving an unconstrained minimization problem. That means it can choose any value for the parameters. Does your ode function have special needs, i.e. no negative values? If so, pass upper and lower parameter bounds to the solver.
- Is the system of ODEs stiff or singular? Try reading the matlab ode solver documentation and changing the parameters with odeset() or changing the ode solver in odeParamEstLSQ.m.
- Is there something about the problem that will make it hard to solve correctly? For example: a very long time domain where very important, very short events occur at irregular intervals? (See the pyrite example.) You may need to adjust the ode options, etc.
- Beware of local minima! Once you get an estimate of the parameters, try several different initial guesses.

4 Examples

4.1 Kinetics Fe^{+2} oxidation by iron oxidizing bacteria

Note: single ode, very clean data set two very similar data sets give different parameter estimates.

Assume that the oxidation of aqueous Fe^{+2} by iron oxidizing bacteria (IOB) can be describe by a single ODE describing the substrate depletion. IOB are cultured in a shaken flask with aqueous substrate, and iron samples are analyzed approximately every hour. Initial and final biomass can be determined.

4.1.1 Equations

$$\frac{dS}{dt} = -\mu_{max} S \frac{(\frac{X_0}{Y} + S_0 - S)}{K_s + S} \tag{4}$$

where S is the substrate (Fe^{+2}) concentration, (mg/L), μ_{max} is the maximum specific growth rate, (hours), X_0 is the initial biomass concentration (mg cells / L), Y is the cell yield (mg cell / mg substrate), K_s is the half saturation constant (mg/L). Note that it is not possible to get separate estimates of X_0 and Y from this equation. The parameters being solved for are: params(1) = μ_{max}, params(2) = K_s, params(3) = $\frac{X_0}{Y}$. The estimates of μ_max and K_s are not independent, and you generally can not get a meaningful estimate of K_s from this data with this equation.

4.1.2 Description of data columns

.

1. time (hours)
2. $[Fe^{+2}]$ (mg/L)
3. weighting for $[Fe^{+2}]$ data

4.2 Pyrite oxidation by Fe^{+3}

Note: two ode's, three parameters, one bad data point, requires user defined parameters and time series data. Compare the inhibition and inhibition models, least squares regression can't tell you if one is better than the other, but additional domain knowledge shows that inhibition does occur.

4.2.1 Equations

$$r_{Fe2} = \frac{15}{14} k [Fe^{+2}]^a [Fe^{+3}]^b \tag{5}$$

$$r_{Fe3} = -k [Fe^{+2}]^a [Fe^{+3}]^b \tag{6}$$

4.2.2 Explanation of data

1. time seconds
2. Fe^{+2} moles
3. weighting for Fe^{+2} data
4. Fe^{+3} moles
5. weighting for Fe^{+3} data
6. volume (ml)
7. delta volume (ml)

4.3 T. Thioparus growth on sodium thiosulfate

Note: limited data, missing data point, two models, does cell death give a better fit to the data?

4.3.1 Equations

$$\frac{dS}{dt} = -\frac{\mu_{max}}{Y} X \frac{S}{K_s + S} \tag{7}$$

$$\frac{dX}{dt} = \mu_{max} X \frac{S}{K_s + S} \tag{8}$$

or taking cell death into account:

$$\frac{dX}{dt} = \mu_{max} X \frac{S}{K_s + S} - b \tag{9}$$

where μ_{max} is the maximum specific growth rate Ks is substrate half saturation constant Y is yield b is ?

4.3.2 Explanation of data

1. time hours
2. X = biomass (Thiobacillus Thioparus) (mg/L)
3. weighting for X
4. S = substrate (sodium thiosulfate) (mg/L)
5. weighting for S
6. pH

4.4 Runoff

Note: observed data is a function of the state variables