glocal_demo.m demonstrates how to use functions in this directory

Table of Contents

This function set overloads the terms "global" and "local" parameter estimation in nonstandard ways. "global" in the current case refers to a parameter value that is expected to be consistent across the individual experiments that are being fit simultaneously (similar in spirit to a fixed effect across patients, but across experiments), while "local" parameter takes on a different value for each experiment (if you are familiar with nonlinear mixed effects, this is like a random effect, but without any distributional assumptions).

This functionality comes in handy when some experimental conditions may effect a "local" parameter in unpredictable/unmeasured ways that need to be taken into account, for example with in vitro experiments that come from different batches/passages of a cell line, which could affect receptor density, for example, whereas other "global" parameters should be universal across all experiments.

Additionally this toolbox allows user to specify a transformation for each parameter:

*For a strictly positive parameter, 'log' is recommended *For a fraction between 0 and 1, 'logit' is recommended *For a real number, just use 'identity' or any other string that doesn't match the two supported strings, and it will be ignored.

lower and upper bounds are not explicitly handled by this package, maybe some other day, but they are enforced as a consequence of the transformation chosen above

D. Bottino, Takeda Pharmaceuticals

clear the workspace and close all figure windows

```
clear all; close all
```

initialize parameter settings

if you want to be fancy you can put this info in a spreadsheet, then just use xlsread to convert to a table like I'm making here. Make sure parameter names are rownames for the table.

```
paramnames = {'Y0';'phiR';'gR';'kS'}; % names of parameters as they
 are known to the models
```

```
fit = [-1 -1 1 1]'; % whether to fit each parameter 'locally' (-1) to
 each experiment, or 'globally' (1) one value across all experiments,
 or not at all (0)
value = [10 0.1 0.009 0.002]'; % default values for the parameters
xform = {'log','logit','log','log'}'; % transformations to real line
 for the parameters.

par = table(fit,value,xform,'RowNames',paramnames);
% but only if we don't allow specification of different
% initial guesses across experiments because that would lead to an
% inconsistently shaped thing

clear paramnames fit value xform % we will use par table hereafter

disp(par);
```

	fit	value	xform
Y0	-1	10	'log'
phiR	-1	0.1	'logit'
gR	1	0.009	'log'
kS	1	0.002	'log'

generate some anonymous functions we need later

don't touch this block

```
v2s = @(vec)vect2struct(vec,par.Properties.RowNames); % local function
 converting parameter vector to parameter structure
fxf = @(mat)fxform(mat,par.xform); % forward transform from parameter
 space to real numbers
ixf = @(mat)ixform(mat,par.xform); % inverse transform from real line
 to parameter space
```

specify the model.

a model is defined as a function taking in parameter and dependent variable values (say a time vector) and spitting out Y values for each dependent variable value. The model can either be an anonymous function or it can be its own standalone function. In this case I'm using a model that takes in a parameter STRUCTURE with fields that are parameter names. If your model already expects a parameter column vector and 'knows' the order of the parameters, you can do it that way too in which case you don't need the v2s function above.

```
tukmodel = @(p,T)(p.Y0*((1-p.phiR)*exp(-p.kS*T) +
 p.phiR*exp(p.gR*T))); % model for tumor volume
rsratio = @(p,T)(p.phiR/(1-p.phiR))*exp((p.gR+p.kS)*T); % model for
 resistant to sensitive cell ratio
```

now we generate the experiment structure

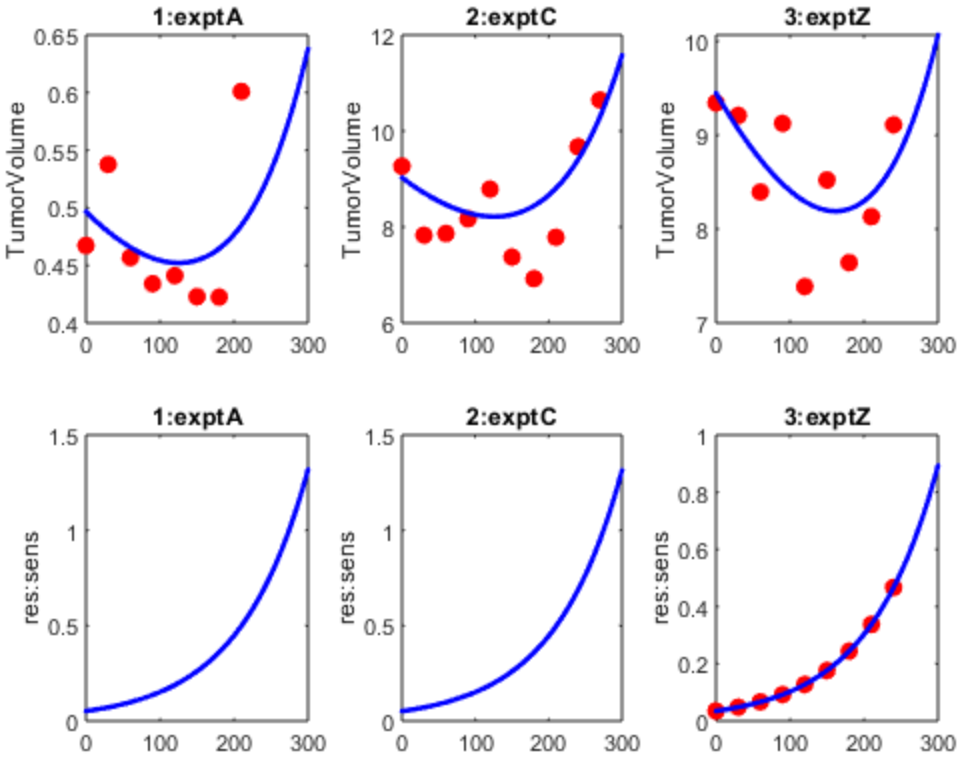
which in real life would be done by loading in experimental observations, not using the model to simulate the data, but this is a demo

```
Ne = 3; % number of distinct experiments to fit to
exptnames = {'exptA','exptC','exptZ'}; % labels for experiments
pmat_true =
 [par.value('Y0')*rand(1,Ne);par.value('phiR')*rand(1,Ne);par.value('gR')*ones(1,N
 "true" parameter values for generating synthetic data for this demo
for i = 1:Ne
 expt(i).name = exptnames{i};
 expt(i).Ynames = {'TumorVolume','res:sens'}; % these can be different
 for each experiment but it's good practice to have all of them for
 all experiments, if possible. The names are matched for parameter
 estimation, so don't make any spelling variations!
 expt(i).model = @(pv,T)[tukmodel(v2s(pv),T) rsratio(v2s(pv),T)]; %
 in this case, model generates two Y outputs for each time point,
 regardless of whether data is available for those two readouts
 expt(i).time = [0:30:200+100*rand]'; % possibly different time vector
 in each experiment. Or could be same.
 expt(i).obs(:,1) =
 tukmodel(v2s(pmat_true(:,i)),expt(i).time).*exp(0.085*randn(size(expt(i).time)));
 let's make some noiiiiise!!
 expt(i).obs(:,2) = NaN; % no data available for 2nd output
 (resistant/sensitive cell ratio). In general you need to pad the data
 matrix with NaNs for those Time/Ytype pairs where there is no data.
end

expt(3).obs(:,2) = [rsratio(v2s(pmat_true(:,3)),expt(3).time)]; %
 let's assume experiment #3 does have res/sens ratio data
```

plot the data and goodness of fit of true parameters...

plot_expt figures everything out

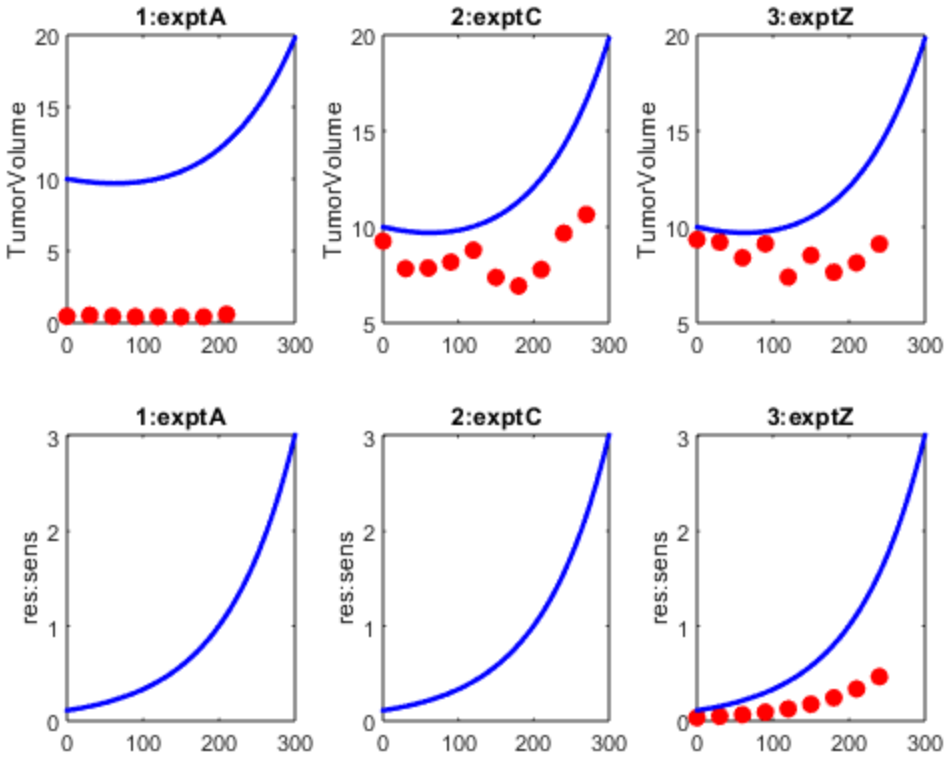
```
figure; plot_expt(expt,pmat_true,[0:300]');
```



plot the data and goodness of fit of (uniform) initial guesses for parameters...

plot_expt figures everything out

```
figure; plot_expt(expt,par.value*ones(1,Ne),[0:300]');
```



ok now setup parameter estimation problem

```
errfun = @(Ypred,Yobs)sum(sum((log(Ypred(~isnan(Yobs)))-
log(Yobs(~isnan(Yobs))))).^2)); % sum squared error function ignoring
 NaNs
[pbig0,prow,pcol] = psquash(fxf(par.value),par.fit,Ne); % for initial
 guesses we use value field in the par table
ofun = @(p)(objfun(p,expt,fxf(par.value),prow,pcol,ixf,errfun)); %
 single parameter vector objective function in transformed space

options = optimset('maxiter',10000,'maxfunevals',10000); % set the
 options for your favorite optimizer
pbigbest = fminsearch(ofun,pbig0,options); % run your favorite
 optimizer

pmat_best = ixf(pfluff(pbigbest,fxf(par.value),prow,pcol,Ne)); %
 "fluff" optimized parameters into pmat shape and inverse transform
 values into parameter space
% convert these into a table for display purposes
tmat_best = table('RowNames',par.Properties.RowNames);
tmat_true = tmat_best;
for i = 1:Ne
 tmat_best.(exptnames{i})=pmat_best(:,i);
 tmat_true.(exptnames{i})=pmat_true(:,i);
end
```

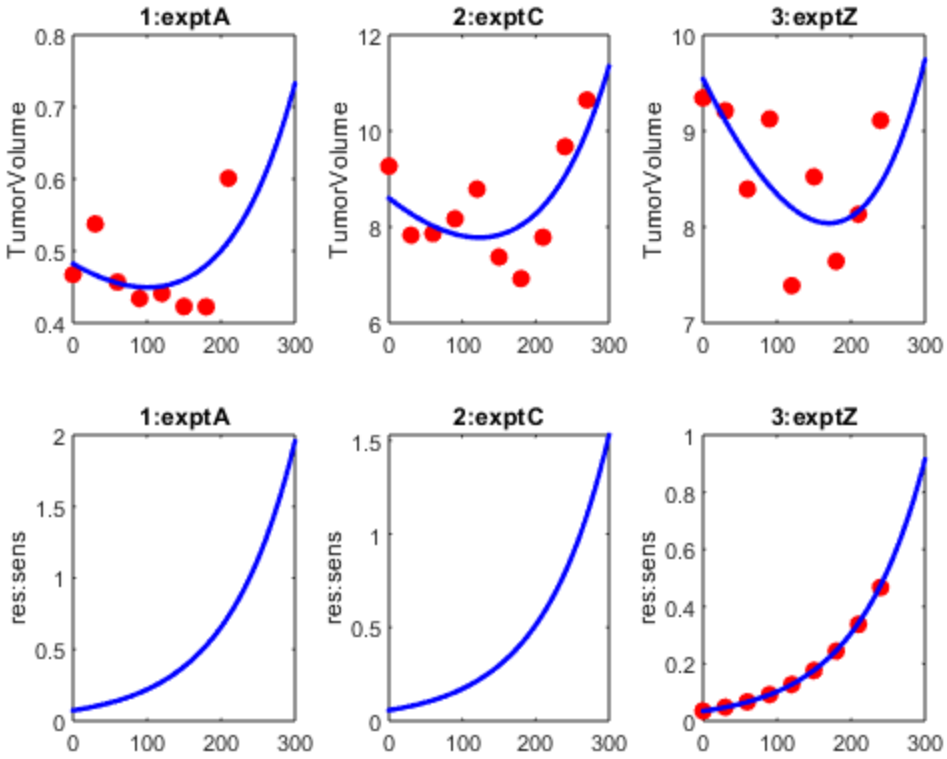
```
disp('*************** PARAMETER SETTINGS *****************');
disp(par)
disp(' ');
disp('****************** TRUE PARAMETERS ******************')
disp(tmat_true);
disp(' ');
disp('*************** ESTIMATED PARAMETERS ***************')
disp(tmat_best)
```

```
*************** PARAMETER SETTINGS *****************
            fit    value     xform
            ___    _____    _______

    Y0      -1        10    'log'
    phiR    -1       0.1    'logit'
    gR       1     0.009    'log'
    kS       1     0.002    'log'


***************** TRUE PARAMETERS ******************
              exptA        exptC        exptZ
            _________    _________    _________

    Y0        0.49654       9.0272       9.4479
    phiR     0.049086     0.048925     0.033772
    gR          0.009        0.009        0.009
    kS      0.0018001    0.0018001    0.0018001


*************** ESTIMATED PARAMETERS ***************
              exptA        exptC        exptZ
            _________    _________    _________

    Y0        0.48239       8.6003       9.5432
    phiR     0.068669     0.054512     0.033323
    gR      0.0089446    0.0089446    0.0089446
    kS      0.0019833    0.0019833    0.0019833
```

plot the final results using the best-fit parameters

that's it!

```
figure; plot_expt(expt,pmat_best,[0:300]');
```



Published with MATLAB® R2018b