Basic Instructions for Using the Matlab Toolbox `empirical_intervals`

Paul Miles[1]

[1]Department of Mathematics, North Carolina State University,
Email: prmiles@ncsu.edu

March 20, 2018

Contents

1 Toolbox Download

The `empirical_intervals` toolbox can be downloaded via the Github repository:
https://github.com/prmiles/empirical_intervals

2 Using the Toolbox

Unzip the folder downloaded from Github. Inside you should find the following contents:

- `empirical_intervals/`
- `examples/`
- `LICENSE.txt`
- `README.md`

You need simply add the `empirical_intervals` directory to your path for use with other Matlab scripts. In the `examples/` directory there is a Matlab file that shows how to access the toolbox via the relative path. It is left the the user's discretion as to the best location to store the toolbox on their computer.

3 Description

The purpose of the `empirical_intervals` toolbox is to generate different statistical intervals based on a set of model responses. It is most logically understood with respect to credible and prediction intervals, in which uncertainty in model parameters as well as observation errors have been propagated through the model. For a full description of credible and prediction intervals, the user is encouraged to review [1]. The Matlab functions that make up this toolbox were adapted from the routines developed by Marko Laine, which can be found at http://helios.fmi.fi/~lainema/mcmc/.

4 Expected Data Structure

The `empirical_intervals` toolbox expects to receive a certain form of data. Assume you have a model response of the form $f(x_i;Q)$, where x_i is the independent variable(s), and Q represents the distribution of the model parameter(s). Then one can evaluate the model using different realizations of Q in order to generate an interval estimation of the model response. This can be pictured in matrix form as

$$f(x_i;Q) \approx \begin{bmatrix} f(x_1;q_1) & f(x_2;q_1) & \cdots & f(x_N;q_1) \\ f(x_1;q_2) & f(x_2;q_2) & \cdots & f(x_N;q_2) \\ \vdots & \vdots & \ddots & \vdots \\ f(x_1;q_M) & f(x_2;q_M) & \cdots & f(x_N;q_M) \end{bmatrix} \tag{1}$$

where N is the number of independent points, and M is the number of realizations drawn from Q. The code expects an $N \times M$ matrix of this form. The form is important as the procedure for identifying interval bounds requires sorting in a column wise manner, i.e., the model response at each evaluation points will be sorted with respect to the realizations.

5 Basic Interval Generation Features in `empirical_intervals`

```
function y=generate_empirical_intervals(f,limits)
% Generates empirical intervals based on a set of model responses (f).  The
% intervals limits require a specific form:
%   limits = [left1, left2, ..., center, ..., right2, right1]
% There should be an odd number of elements in 'limits'.  If the center is
% 0.5, this will correspond to the middle row (or interpolated middle row)
% of the sorted model responses.  Note, the model responses will be sorted
% in a column wise fashion.
```

The intervals are generated by calling the `generate_empirical_intervals` routine. The model response described in Section 4 is the first input argument, `f`. The regions that you want plotted as intervals must be bounded in some sense, and they are defined in the `limits` variable. To generate a symmetric 95% response interval, you would define `limits = [0.025,0.5,0.975]`. The median/mean model response is denoted by 0.5 (center). The general form of the `limits` variable should follow the form: `[left1, left2, ..., center, ..., right2, right1]`.

6 Basic Plotting Features in `empirical_intervals`

```
function [medianhandle, fillhandle, settings] = ...
    plot_empirical_intervals(time, intervals, inputsettings)
% Settings for plotting emprical intervals
% Options:
% - colorscheme: 'gradient' or 'spectrum';
%   - gradient:
%       * initialcolor: [r,g,b] - dimmest (lightest) color
%   - spectrum:
%       * colormap: jet, parula, hsv, etc. (see MathWorks)
% - figID: specify figure handle - leave empty to generate new figure;
% - transparency_level: # number between 0 (translucent) and 1 (opaque);
% - meanmodelstyle: plot style/color for mean model response
```

The user has several options for displaying the empirical intervals. First, the user can choose a scheme/method for coloring the intervals.

- `colorscheme`: '`gradient`' or '`spectrum`', `default = 'gradient'`
  - '`gradient`': color changes from light to dark as you approach the mean model response
    - `initialcolor`: set the RGB level for lightest interval color, `default = [0.9, 0.9, 0.9]`
  - '`spectrum`': spans breadth of colormap by dividing into steps corresponding to the number of intervals
    - `colormap`: specify Matlab colormap from which to generate spectrum, `default = parula`
- `figID`: plot intervals on specific figure, `default = []`
- `transparency_level`: # between 0 (translucent) and 1 (opaque), `default = 1`
- `meanmodelstyle`: plot style for mean model response, `default = '-k'`

The user's settings are sent to the plotting routine in the `inputsettings` variable. How to define settings and send them to the function will be best explained via the example script in Section 7.

7 Model Response in Example File

The example file is a simple polynomial function often used in describing Landau energy surfaces. The form of the model is

$$\psi = \alpha_1 P^2 + \alpha_{11} P^4. \tag{2}$$

The general form of this model response is shown in Figure 1. This model is extremely convenient for calculating the sensitivity and subsequently the covariance matrices

$$X = \begin{bmatrix} \frac{d\psi}{d\alpha_1} & \frac{d\psi}{d\alpha_{11}} \end{bmatrix} = \begin{bmatrix} P^2 & P^4 \end{bmatrix} \tag{3}$$

$$V = \sigma^2 [X X^T]^{-1} \tag{4}$$

This information is subsequently used in generating responses of a statistical model and calculating the corresponding realization of the random parameter values. The generation of the statistical model response is outlined in Listing 1.

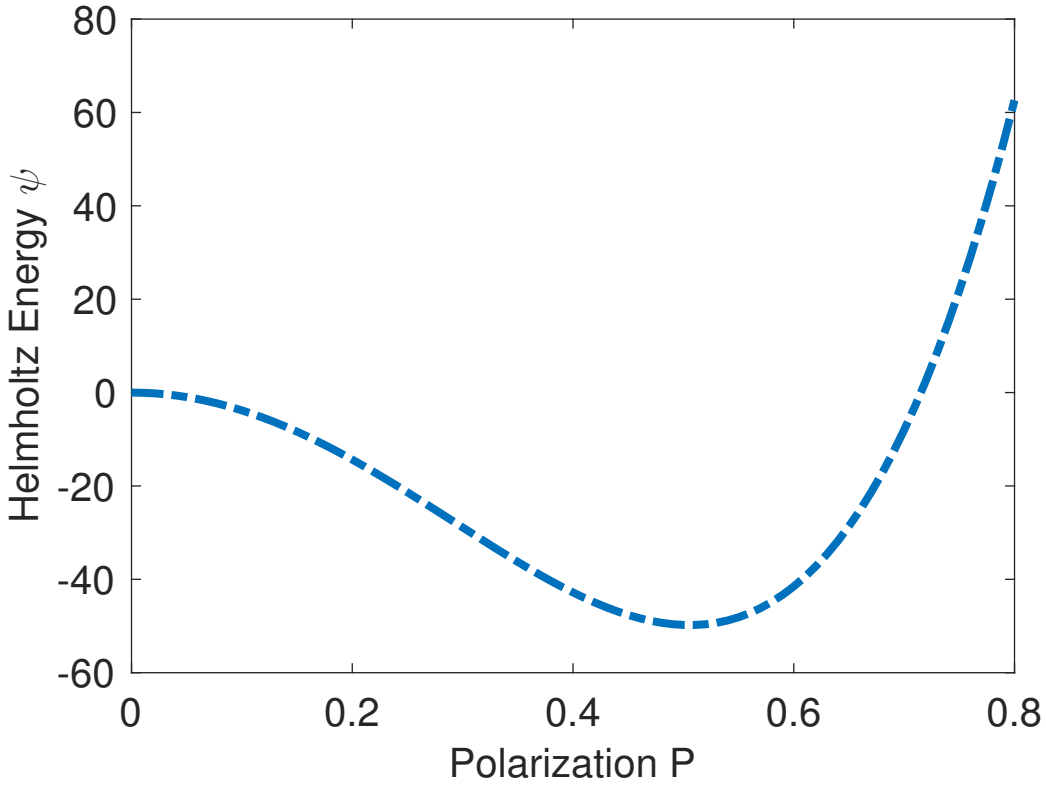

Figure 1: Basic form of example model.

Listing 1: Generation of model observations.

```
% Define polarization grid
Pf = 0.8;
P = 0:.01:Pf;

% Define mean parameter values and variances
alpha_1 = -389.4;
alpha_11 = 761.3;
n = length(P);

sigma = 2.2;
var = sigma^2;

% Compute Helmholtz energy
psi = alpha_1*P.^2 + alpha_11*P.^4;

% Compute the derivatives with respect to alpha_1 and alpha_11 and compute
    the
% sensitivity matrix X, Fisher information matrix F, and covariance matrix V
    .
psi_alpha_1 = P.^2;
psi_alpha_11 = P.^4;
X = [psi_alpha_1; psi_alpha_11];
F = X*X';
Finv = inv(F);
V = var*Finv;

% For N = 1000 iterates, compute alpha_1, alpha_11 and psi and plot.
N = 1e+3;
for j = 1:N
    error = sigma*randn(size(P));
    obs = psi + error;
    q(:,j) = Finv*X*obs';
    Y(j,:) = obs;
end
```

8 Examples

8.1 Example 1:

Listing 2: Code for Example 1.

```
%% Example 1
% Test:
% - default settings
[mh, fillh] = plot_empirical_intervals(P, Y_empirical_intervals);
axis([0 Pf -60 80])
set(gca,'Fontsize',[20]);
xlabel('Polarization P')
ylabel('Helmholtz Energy \psi')
legend([mh, fillh(1:4)], {'Mean', '99%', '95%','90%','50%'}, 'Location', '
    NorthWest')
```

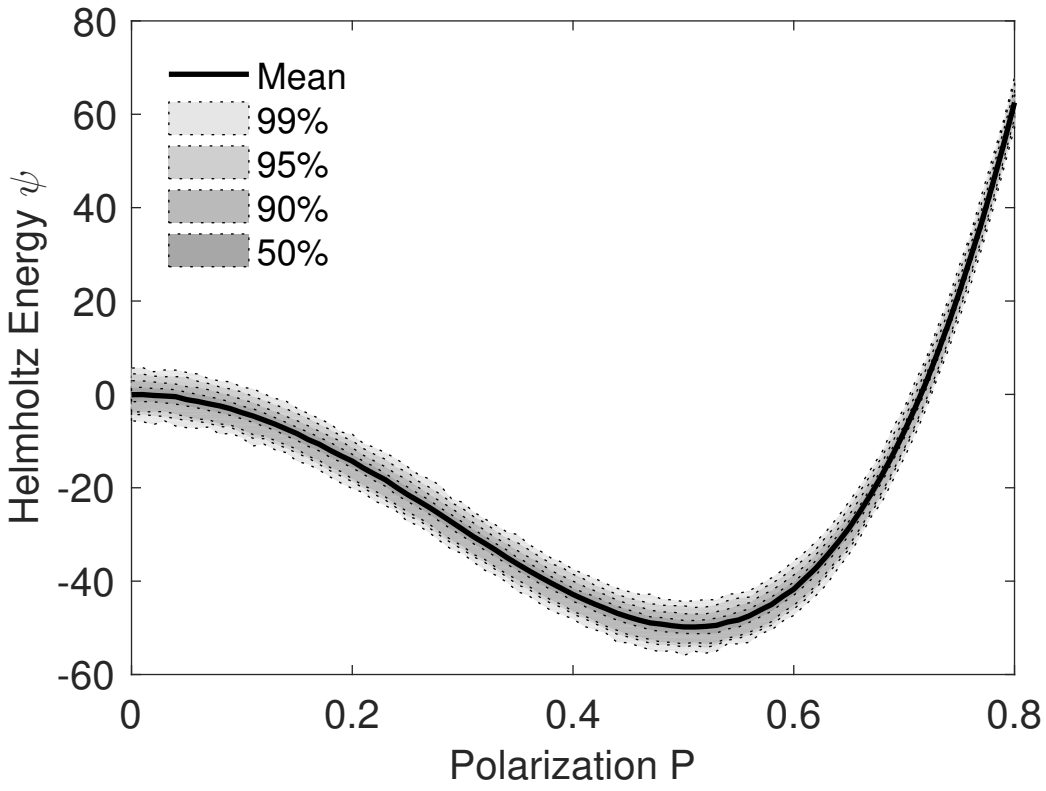

Figure 2: Default interval appearance. Intervals get darker the closer they get to the center response.

8.2 Example 2:

Listing 3: Code for Example 2.

```
%% Example 2
% Test:
% - colorscheme - 'gradient'
clear settings
settings.colorscheme = 'gradient';

[mh, fillh] = plot_empirical_intervals(P, Y_empirical_intervals, settings);
axis([0 Pf -60 80])
set(gca,'Fontsize',[20]);
xlabel('Polarization P')
ylabel('Helmholtz Energy \psi')
legend([mh, fillh(1:4)], {'Mean', '99%', '95%','90%','50%'}, 'Location', '
    NorthWest')
```

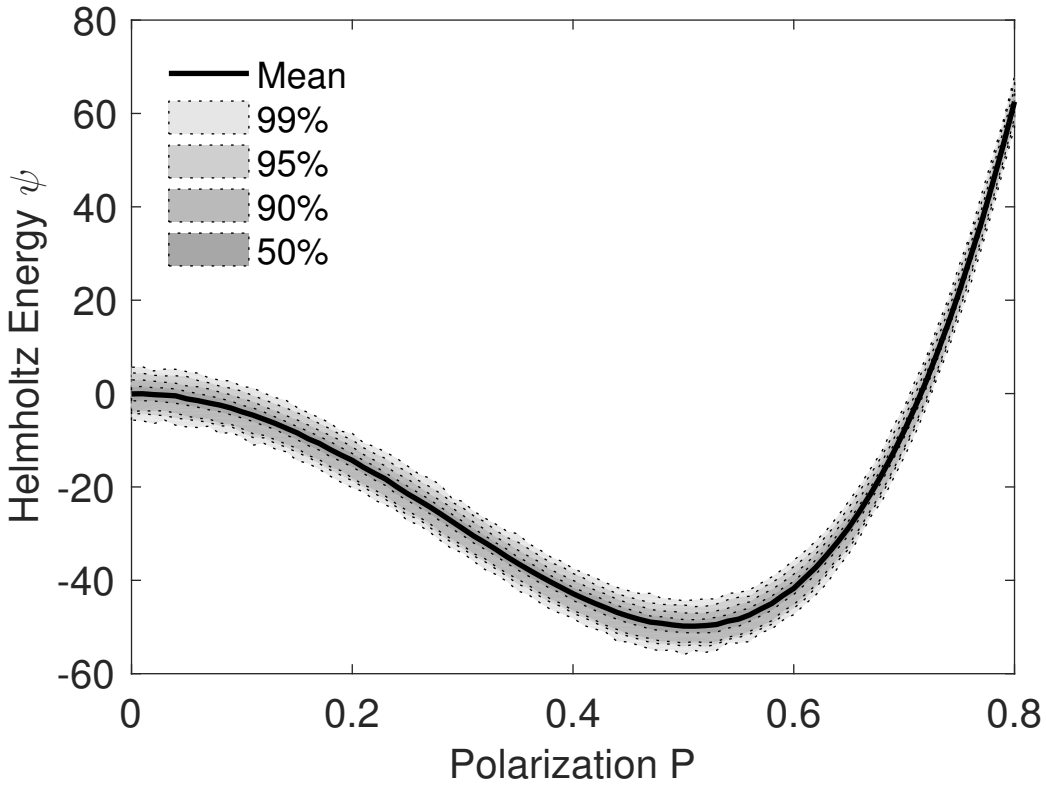

Figure 3: User specified to use the gradient method - should match the default appears from Figure 2.

8.3 Example 3:

Listing 4: Code for Example 3.

```
%% Example 3
% Test:
% - colorscheme = 'gradient'
% - initial_color = [0.5, 0.9, 0.7];
clear settings
settings.colorscheme = 'gradient';
settings.initial_color = [0.5, 0.9, 0.7];

[mh, fillh] = plot_empirical_intervals(P, Y_empirical_intervals, settings);
axis([0 Pf -60 80])
set(gca,'Fontsize',[20]);
xlabel('Polarization P')
ylabel('Helmholtz Energy \psi')
legend([mh, fillh(1:4)], {'Mean', '99%', '95%','90%','50%'}, 'Location', '
    NorthWest')
```

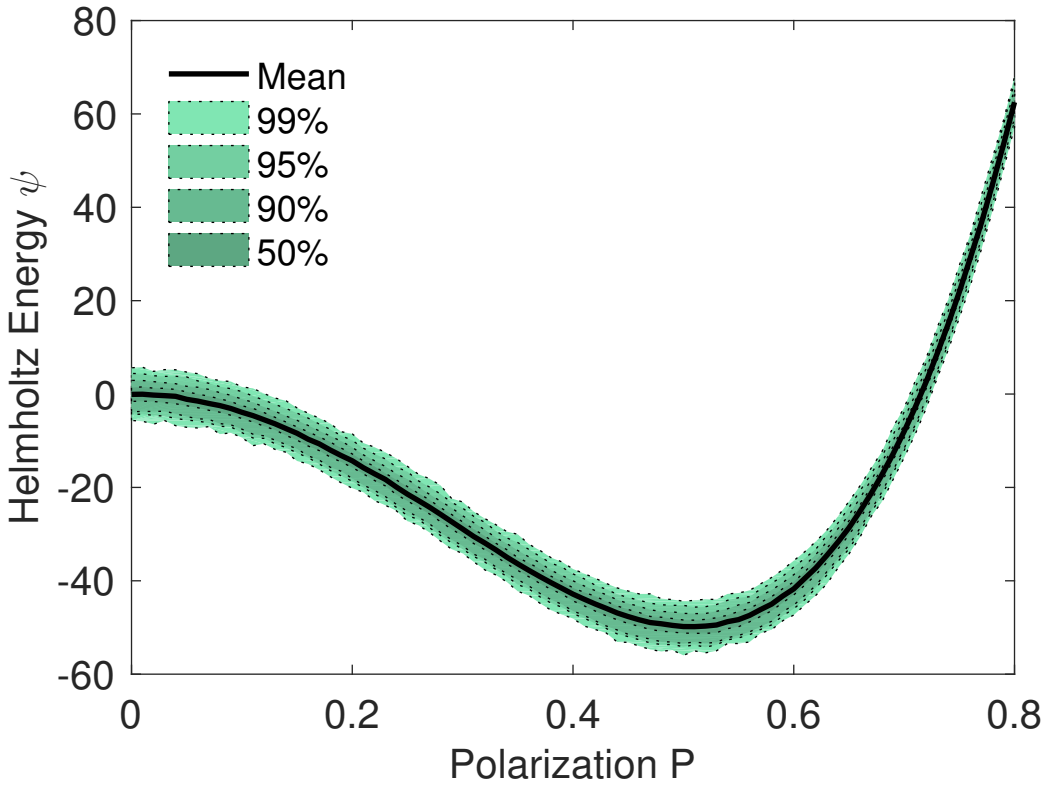

Figure 4: User specified a different initial color vector for the gradient method.

8.4 Example 4:

Listing 5: Code for Example 4.

```
%% Example 2
% Test:
% - colorscheme - 'gradient'
clear settings
settings.colorscheme = 'gradient';

[mh, fillh] = plot_empirical_intervals(P, Y_empirical_intervals, settings);
axis([0 Pf -60 80])
set(gca,'Fontsize',[20]);
xlabel('Polarization P')
ylabel('Helmholtz Energy \psi')
legend([mh, fillh(1:4)], {'Mean', '99%', '95%','90%','50%'}, 'Location', '
    NorthWest')
```

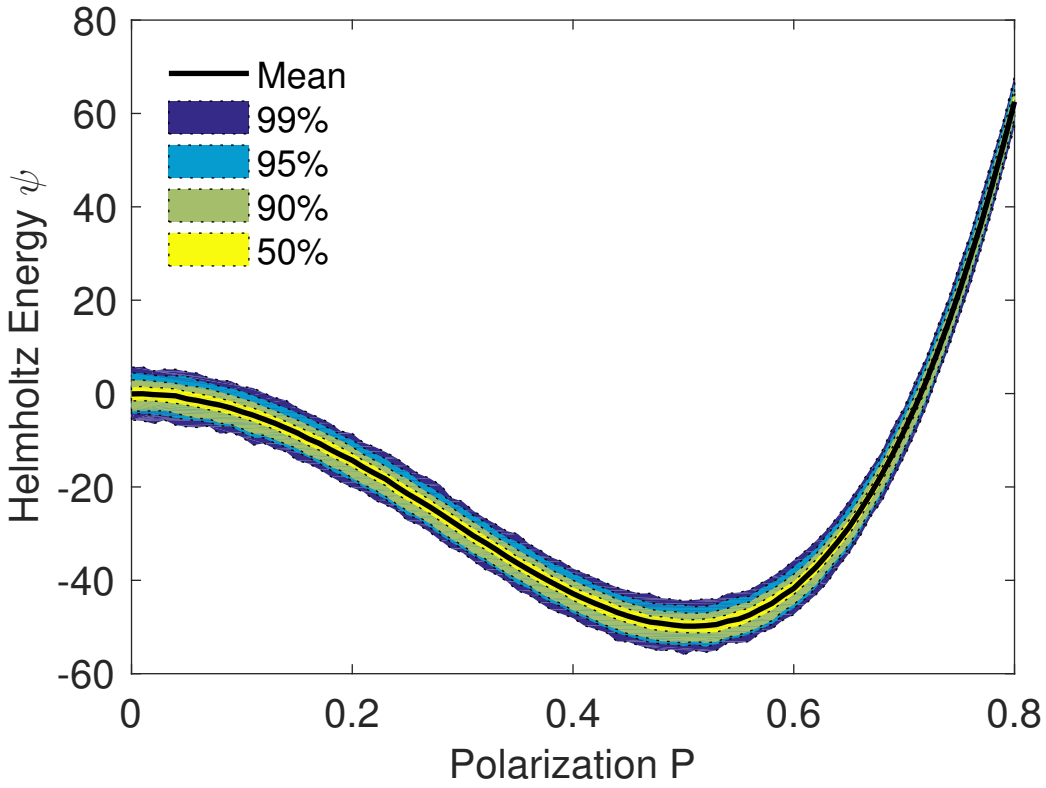

Figure 5: User specified spectrum method - uses default color map.

8.5 Example 5:

Listing 6: Code for Example 5.

```
% Test:
% - figID = gcf
clear settings
settings.figID = figure(10);
settings.transparency_level = 0.5;

% Test adding empirical intervals to existing plot with transparency
figure(10)
pd = plot(P,Y,':k',P,0*P,'k','linewidth',1.0);

[mh, fillh] = plot_empirical_intervals(P, Y_empirical_intervals, settings);
axis([0 Pf -60 80])
set(gca,'Fontsize',[20]);
xlabel('Polarization P')
ylabel('Helmholtz Energy \psi')
legend([pd(1), mh, fillh(1:4)], {'Raw Data', 'Mean', '99%', '95%','90%','50%
    '}, 'Location', 'NorthWest')
```

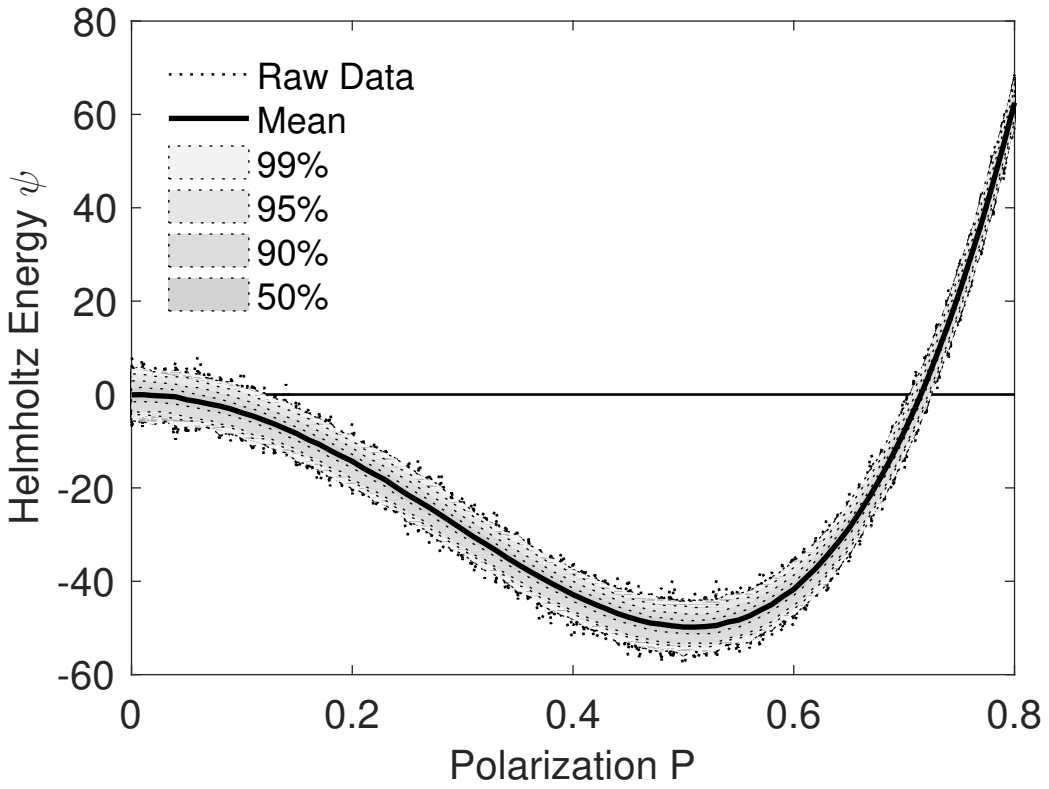

Figure 6: User specified transparency and specific figure on which to add interval plots. Raw data is included - roughly 1% of data can be seen outside the 99% interval.

8.6 Example 6:

Listing 7: Code for Example 6.

```
%% Example 6
% Test:
% - colormap = hot
clear settings
settings.colorscheme = 'spectrum';
settings.colormap = hot;

[mh, fillh, settings] = plot_empirical_intervals(P, Y_empirical_intervals,
    settings);
axis([0 Pf -60 80])
set(gca,'Fontsize',[20]);
xlabel('Polarization P')
ylabel('Helmholtz Energy \psi')
legend([mh, fillh(1:4)], {'Mean', '99%', '95%','90%','50%'}, 'Location', '
    NorthWest')
```

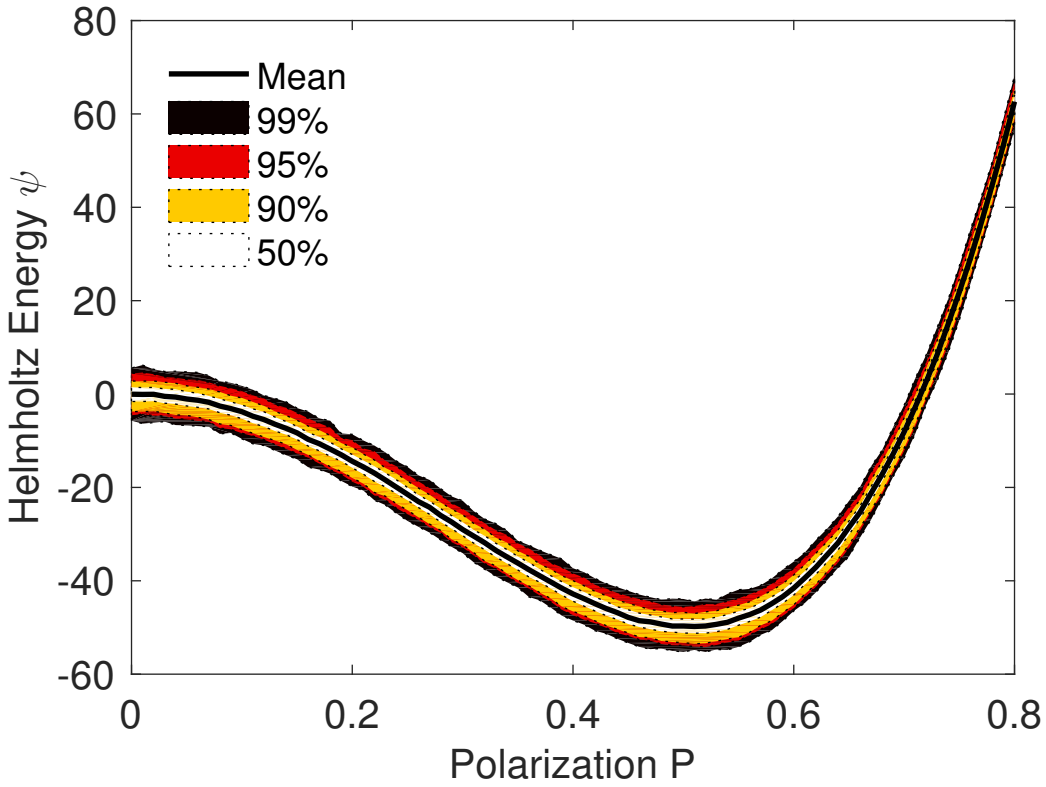

Figure 7: User specified spectrum method using the `hot` color map in Matlab.

References

[1] Ralph C Smith. *Uncertainty quantification: theory, implementation, and applications*, volume 12. Siam, 2013.