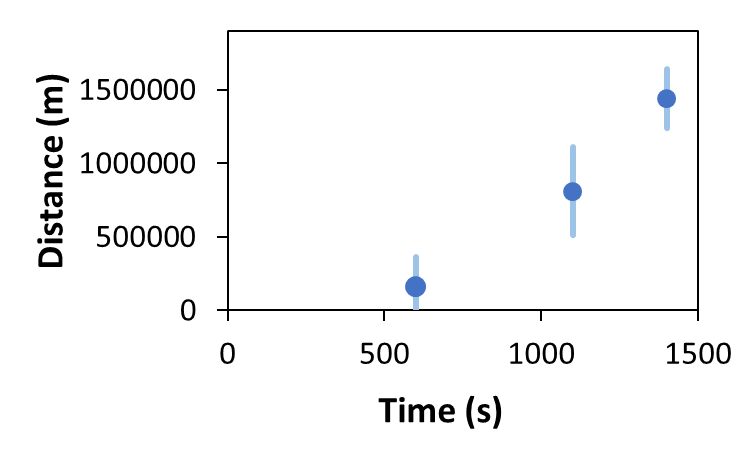
Why should you use CheKiPEUQ to get parameters from observed data? A few lines of code will give you realistic estimates and some graphs.

Consider a situation where we have three observed experimental data points with uncertainties:



Their values, including uncertainties, are:

160500 +/- 200000

810500 +/- 300000

1440500 +/- 200000

Consider that this situation is known to be described the following equation:

y=(x-a)^2 + b

Where we know that the physically realistic values of “a” and “b” are:

a is expected to be 200 +/- 100 (this is the 1 sigma confidence interval)

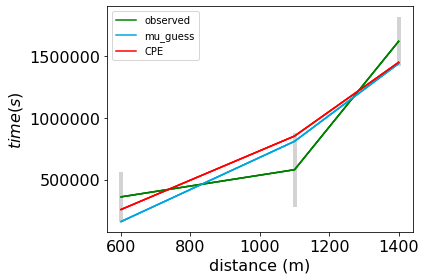
b is expected to be 500 +/- 200 (this is the 1 sigma confidence interval)

If one tries to do a sum of squares fitting (conventional parameter estimation, CPE), we will not get realistic values for “a” and “b”. We get **a = 255, b = 139153**. The value for “a” is fine, but the value for “b” is not realistic.

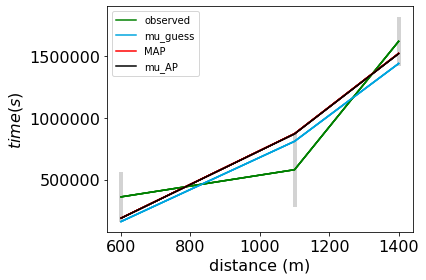
However, if we do a Bayesian Parameter Estimation (BPE), what CheKiPEUQ is designed for, then we get the following answers: **a = 166 +/- 57, b= 509 +/- 198**. Where these errors are the 1 sigma credible intervals. Notice that now ***both*** of the parameters have physically realistic values. We even have error bars that took into account the uncertainty! The covariance matrix for the parameters is also provided, so that the correlated uncertainties of estimated parameters is not lost.

**How good is the match in this example?**

The fitting (CPE) gives the red line below:



The Bayesian Parameter Estimation gives the black line below (and the red, not explained here):



We see that for this example, the CPE result from fitting and the BPE results do not look very different from each other. Both parameter estimation methods manage to stay in the error bars, yet the BPE result has a far more physically realistic pair of parameters! This is the main purpose using CheKiPEUQ BPE: it will tend to give more realistic parameter estimates, and can even give a type of uncertainty (called credible intervals) on the final estimates.

Here is the code that was required after making the model equation:

import CheKiPEUQ as CKPQ

import CheKiPEUQ.UserInput as UserInput

UserInput.model['InputParameterPriorValues'] = [200, 500] #prior expected values for a and b

UserInput.model['InputParametersPriorValuesUncertainties'] = [100, 200] #1 sigma, in this example not correlated, but a covariance matrix can be used instead.

UserInput.model['simulateByInputParametersOnlyFunction'] = simulation\_model\_00.simulation\_function\_wrapper #This just points to the User created model equation.

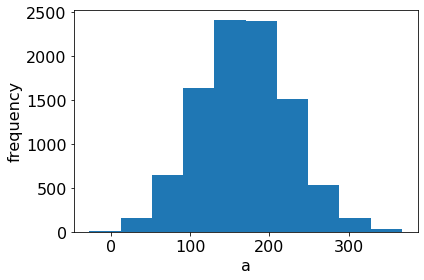
PE\_object = CKPQ.parameter\_estimation(UserInput)

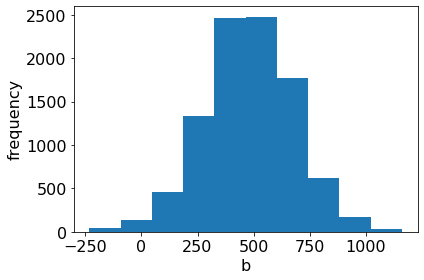
PE\_object.doEnsembleSliceSampling()

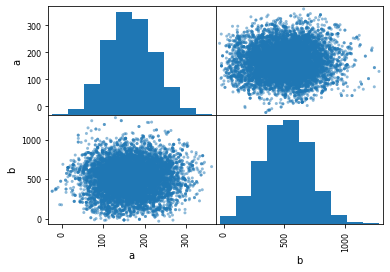
PE\_object.createAllPlots()

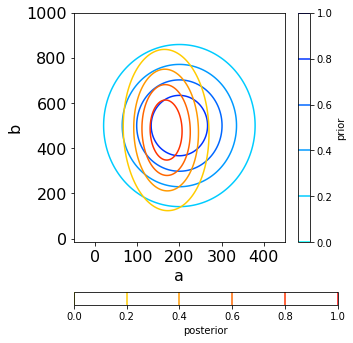
There is a logfile generated called mcmc\_log\_file.txt (along with other files in the directory).

You will also get the following plots, some of which can be further customized, such as removing the bars from the contour plots.









We can see that in this example the position and uncertainty in “a” narrowed more than that of “b”.

Additional Information on The two types of mcmc

There are two types of mcmc samplings possible in CheKiPEUQ. The EnsembleSliceSampling will be faster for many higher dimensional problems by needing fewer (but more sophisticated) samplings: there are normally almost no rejections. The MetrpolisHastings routine is what CheKiPEUQ was originally built with, but is no longer recommended except for cases when EnsembleSliceSampling is not working well.

If we compare the outputs and performance from 00a1 and 00a2, we see that the outputs are similar but that the ESS is a bit slower than the MH.

However, if we go to the harder case of 00c1 and 00c2 (which is uniform distributions for each sample), then we find that MH requires very long sampling to get a reasonable result while ESS can do a good job easily.