C:\Users\fvs\Documents\GitHub\CheKiPEUQ\Examples\Example15\runfile\_Example15a\_two\_site\_ratio\_d\_initial\_guess.py

Took the initial guess from a gridsearch (d) and then did mcmc using that as an initial guess. That gridsearch had a “reasonable” value, but the mcmc ended up pushing A1 into unrealistic values, despite starting at a realistic value.

~~Now in version 16, adding in the vertical offset which is going to be 0.005 and 0.001 as the standard deviation.~~

Changed back to using original file input with 0.005 uncertainty

Using this: C:\Users\fvs\Documents\GitHub\CheKiPEUQ\Examples\Example16\runfile\_Example15d\_two\_site\_ratio.py

With this:

PE\_object.doGridSearch('getLogP', gridSamplingAbsoluteIntervalSize=np.array(UserInput.model['InputParametersPriorValuesUncertainties']), gridSamplingNumOfIntervals=[2,2,2,2,2,2,2], verbose = True)

Which takes 10 minutes. Apparently below would only take 1 hour. So could make a finer grid.

PE\_object.doGridSearch('getLogP', gridSamplingAbsoluteIntervalSize=np.array(UserInput.model['InputParametersPriorValuesUncertainties']), gridSamplingNumOfIntervals=[3,3,3,3,3,2,2], verbose = True)

Will do this….

gridSamplingAbsoluteIntervalSize = [0.50/3, 10, 10, 1,1, 0.1, 0.1]

gridSamplingNumOfIntervals=[3,5,5,5,5,2,2], which is around 10 hours.

Had this result, which is concerning because of the 18 being at he edge of the allowed range.

0.33333333333333337, 21.5, 31.5, 12.0, 18.0, -0.1, 0.30000000000000004

Doing 20,000 mcmc samplings in C:\Users\fvs\Documents\GitHub\CheKiPEUQ\Examples\Example16\runfile\_Example15a\_two\_site\_ratio\_d\_initial\_guess.py

MAP\_logP:[153.83066979]

self.map\_index:16325

self.map\_parameter\_set:[ 0.25621994 24.18377348 37.56220456 13.54975894 23.9337553 -0.23260353

0.32934818]

self.mu\_AP\_parameter\_set:[ 0.28020449 21.98859657 34.34588062 12.04392328 21.309083 -0.20579051

0.37237914]

self.stdap\_parameter\_set:[0.03286409 1.02479803 1.72291198 0.72492619 1.30577351 0.03797193

0.05300712]

Now doing 200,000 samplings to see if the pre-exponential 2 of 21 goes even higher.

Basically, yes:

MAP\_logP:[590.58207571]

self.map\_index:163006

self.map\_parameter\_set:[ 0.73219435 31.28311851 45.33287731 17.41267305 29.34240026 -0.14982982

0.30270917]

self.mu\_AP\_parameter\_set:[ 0.77918148 25.65596376 41.13716493 13.53095376 26.17482362 -0.19787239

0.3180629 ]

self.stdap\_parameter\_set:[0.05653414 2.48512324 2.5346084 1.74610975 1.89415728 0.02530495

0.04507378]

Going to 9 parameter case with runfile\_Example16a\_two\_site\_NineParameters and processing\_functions\_tpd\_odeint\_two\_site\_NineParameters

Strangely, it now was taking more time to do the grid sampling with 9 parameters, 4-11 days.

Need to figure out why.

Also, this set of parmaeters from mcmc:

self.map\_parameter\_set:[ 1.10041633e+00 -2.86936286e-03 -7.59004532e-02 2.75992148e+01

3.14641581e+01 1.63898938e+01 1.43717723e+01 -1.80551162e-01

4.13798151e-01]

Produced a curve that went negative at the end. Need to see how it got a negative rate. Was it just due the background subtraction, or something else?

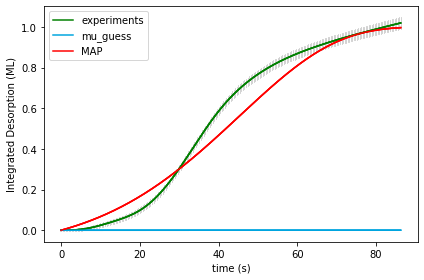
* Ok, eventually I figured out that what happened is that the ratio went to negative such that the second site has a negative concentration. This means it’s more important to have the absolute cutoffs.
* After adding a lower bound of 0, then the ratio even tried going over 1. So needed to put an upper bound of 1 also.

Now, with these bounds, using C:\Users\fvs\Documents\GitHub\CheKiPEUQ\Examples\Example16\runfile\_Example16a\_two\_site\_NineParameters\_grid\_reduced.py and PE\_object.doGridSearch('getLogP', gridSamplingAbsoluteIntervalSize=[0.03, 0.0025, 0.50/3, 10, 10, 1,1, 0.1, 0.1], gridSamplingNumOfIntervals=[0,0,2,4,4,4,4,2,2], verbose = True)

This has around 2 hours of waiting time, (apparently 800,000 points) and is reasonable for a screening.

This gave grid map of (1.0, 0.0, 0.5, 30.0, 20.0, 17.0, 11.0, 0.30000000000000004, 0.1)

It actually does not look like a very good match, but will start with it for MCMC and see what happens (below is what it looks like). It’s good that it looks like a bad match, that means it’s a local max on a prior.



Taking that as a starting point and using C:\Users\fvs\Documents\GitHub\CheKiPEUQ\Examples\Example16\runfile\_Example16a\_two\_site\_NineParameters.py

Sadly, this quickly resulted in terrible numbers. Turning off scaling uncertainties.

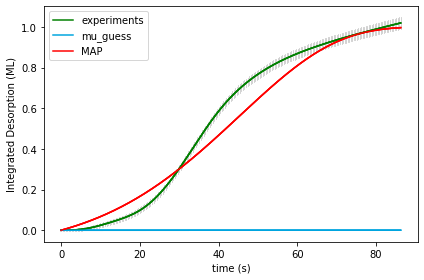
That changed the output, but the output still went into unrealistic territory.

Making the prior uncertainties like this seemed to work:

UserInput.model['InputParametersPriorValuesUncertainties'] = [ 0.10, 0.005, 0.50/3, 20, 20, 2, 2, 0.1, 0.1]

Grid search gave:

Final map results from gridsearch: (1.0, 0.0, 0.5, 30.0, 20.0, 17.0, 11.0, 0.30000000000000004, 0.1) final logP: -859.4228753252021



Unfortuantely, further sampling ended up similar to before, note that the A1 is 2.2E1 which is 22. The rest are probably okay.

MAP\_logP:[351.5613435]

self.map\_index:91337

self.map\_parameter\_set:[1.12201353e+00 1.17147534e-04 1.35970903e-01 3.63785919e+01

8.45020338e+01 2.23622925e+01 1.34879578e+01 4.49840683e-01

2.15005533e-01]

self.mu\_AP\_parameter\_set:[1.14670154e+00 7.35148777e-04 1.20185195e-01 3.50046786e+01

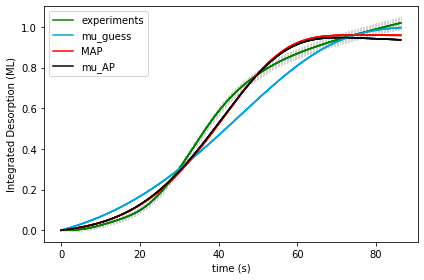
6.46914089e+01 2.13986512e+01 1.39505492e+01 4.64717038e-01

1.18772359e-01]

self.stdap\_parameter\_set:[7.48158390e-02 1.28785820e-03 7.30652837e-02 1.87451011e+00

2.04903992e+01 1.30831121e+00 9.21891142e-01 6.80434390e-02

1.09855635e-01]



The contour plots are saved as

Trying mcmc\_relative\_step\_length of 0.05 to see what happens after 100000 steps. Get a similar map but “nicer” match.

MAP\_logP:[557.03577242]

self.map\_index:95236

self.map\_parameter\_set:[1.14124173e+00 1.18804842e-03 2.73603460e-01 3.68304092e+01

2.89629004e+01 2.31287321e+01 1.56145376e+01 2.99737044e-01

1.40512992e-01]

self.mu\_AP\_parameter\_set:[1.06062424e+00 9.22119977e-04 3.28924808e-01 3.15683581e+01

2.42642984e+01 1.91201743e+01 1.32472582e+01 2.80919235e-01

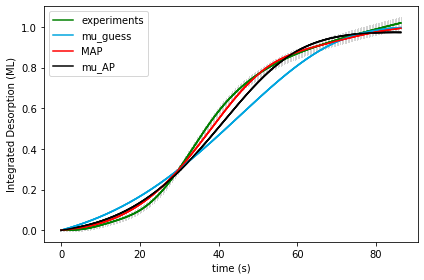
1.18815765e-01]

self.stdap\_parameter\_set:[5.01955346e-02 5.91562078e-04 9.42154520e-02 3.10817066e+00

2.33591687e+00 2.31408563e+00 1.24991695e+00 3.11942663e-02

3.11011646e-02]

self.info\_gain:-306.28470383514485



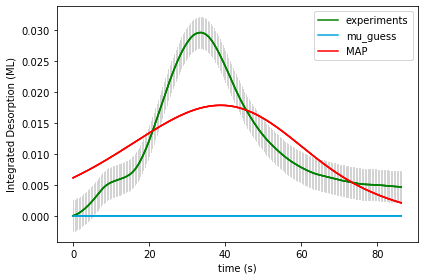
Made C:\Users\fvs\Documents\GitHub\CheKiPEUQ\Examples\Example16\runfile\_Example16a\_two\_site\_NineParameters\_optimize\_negLogP\_transformed.py

Now I am going to use the conversion to integral and back by looking at example 1d. Did that and created 16a as a result.

16a and 16d are both running.

For 16a the grid search map output is:

(1.0, 0.0, 0.6666666666666666, 21.5, 21.5, 11.0, 12.0, 0.2, 0.1) -4970.366869435983



Strange that it’s not a better match visually like the above cases.

Apparently I needed to get 0.3 for the gammas, and didn’t go that far.

With even bigger grid get this, which is not much different:

(1.0, 0.0, 0.6666666666666666, 21.5, 21.5, 11.0, 12.0, 0.30, 0.1) -4961.719559184852

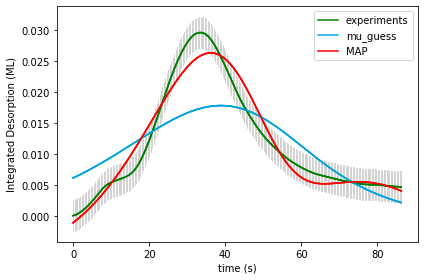
16b as planned with 16a

It looks okay visually, and the kinetic params are okay, but the scaling factor is not very good, highlighted below:

[1.89153477e+00 4.10798815e-03 4.39042344e-01 3.09097300e+01

2.31617314e+01 1.90892567e+01 1.13489468e+01 1.39278627e-10

8.08367926e-02] 652.9040035123369



Tried to make 16b double uncertainty and there was almost no change. So it really is that this local minimum seems to go towards the high scaling factor.

One idea alternatively would be to have a sequential parameter estimation.

* using SLSQP does not do things usefully.

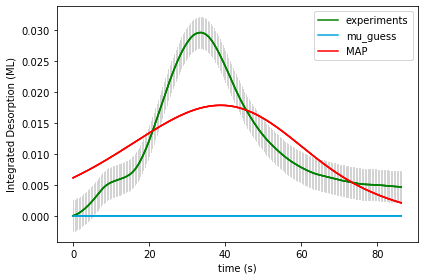
16d was similar to 16a:

Final map results from gridsearch: (1.0, 0.0, 0.6666666666666666, 21.5, 21.5, 11.0, 12.0, 0.2, 0.1) final logP: -4965.0914958436715

(1.0, 0.0, 0.6666666666666666, 21.5, 21.5, 11.0, 12.0, 0.2, 0.1) -4965.0914958436715

Bigger grid also about the same.

(1.0, 0.0, 0.6666666666666666, 21.5, 21.5, 11.0, 12.0, 0.30000000000000004, 0.1) -4954.944185592541

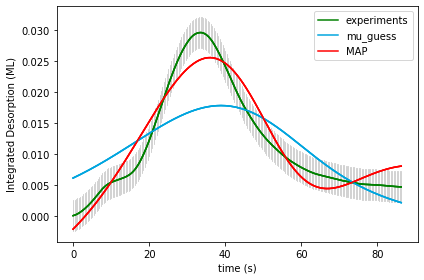


16e then had this, which is pretty similar to the 16b. Obviously it’s a local minimum.

[2.71694618e+00 4.51048101e-03 5.00771456e-01 2.64011542e+01

2.34869779e+01 1.57809111e+01 1.09706465e+01 1.99366259e-06

7.24099554e-02] 613.2649110505431



Note: tried BFGS and did not get anything much better.

C:\Users\fvs\Documents\GitHub\CheKiPEUQ\Examples\Example16\runfile\_Example16a\_opt.py

Using PE\_object.doGridSearch('doOptimizeNegLogP', gridSamplingAbsoluteIntervalSize=[ 0.10, 0.005, 0.50/3, 20, 20, 2, 2, 0.1, 0.1], gridSamplingNumOfIntervals=[1,1,1,1,1,1,1,1,1], passThroughArgs={"method":"Powell"})#, "maxiter":1000, "verbose":False})

Gave:

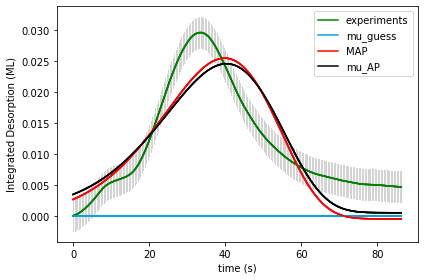
16c tried no initial guess and 100,000 mcmc length, which means input was:

UserInput.model['InputParameterPriorValues'] = [ 1.0, 0.0, 0.50, 41.5, 41.5, 13.0, 13.0, 0.1, 0.1] # Ea1\_mean, Ea2\_mean, log\_A1\_mean, log\_A2\_mean, gamma\_1\_mean, gamma\_2\_mean

Output was: [9.85030462e-01 4.93570288e-04 3.09149191e-03 3.50171900e+01

1.75795603e+02 2.14684711e+01 2.59394822e+01 4.13007251e-01

3.27442809e-01]

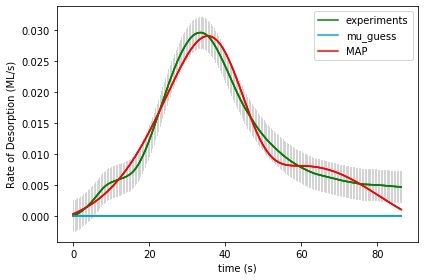


Now doing another 100,000 from there. Was not useful.

Found a way that works:

runfile\_Example16a\_BPE\_grid\_opt:

PE\_object.doGridSearch('doOptimizeNegLogP', gridSamplingAbsoluteIntervalSize=[ 0.10, 0.005, 0.50/3, 20, 20, 2, 2, 0.1, 0.1], gridSamplingNumOfIntervals=[0,0,1,1,1,1,1,0,0], passThroughArgs={"method":"Nelder-Mead", "verbose":False})#, "maxiter":1000, "verbose":False})



Doing it again with maxiter of 5000.

Changed Strategy somewhat, now doing a series of grid searches which will then be optimized:

**Example 16a\_BPE\_grid:**

Final map results from gridsearch: (1.1, 0.0, 0.16666666666666669, 21.5, 21.5, 11.0, 13.0, 0.2, 0.1) final logP: -8752.761553714412

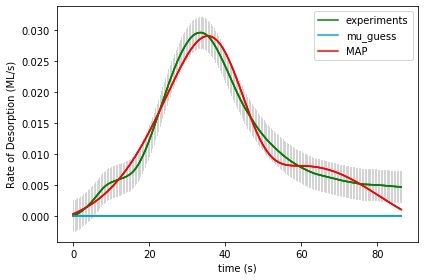


**Example 16a\_BPE\_grid\_opt: with Nelder-Mead and up to 5000 simulation iterations per point.**

Final map results from gridsearch: [1.23943326e+00 1.95055604e-03 5.62199599e-01 2.60209307e+01

4.47401296e+01 1.40086172e+01 2.92348678e+01 4.22570091e-02

2.22870485e-02] final logP: 779.8462737897203



**Example 16a\_BPE\_grid\_mcmc: 243 points with 1 burn in and 1000 samplings each.**

**Example 16a\_CPE\_grid:** 1,361,367 grid points. On the order of 5 hours to run.

Final map results from gridsearch: (1.1, 0.0, 0.16666666666666669, 21.5, 21.5, 11.0, 13.0, 0.2, 0.1) final logP: -8738.991106967724

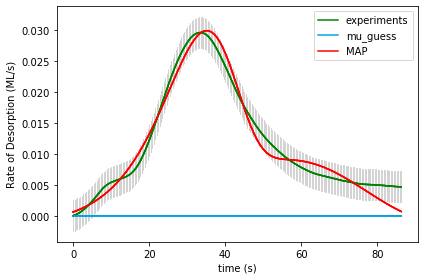


**Example 16a\_CPE\_grid\_opt: with Nelder-Mead and up to 5000 simulation iterations per point.**

Final map results from gridsearch: [1.17855879e+00 1.56599493e-03 4.89378831e-01 5.01380589e+01

2.47742710e+01 3.31859754e+01 1.33279804e+01 1.37736928e-02

1.18045551e-01] final logP: 831.2464561309017



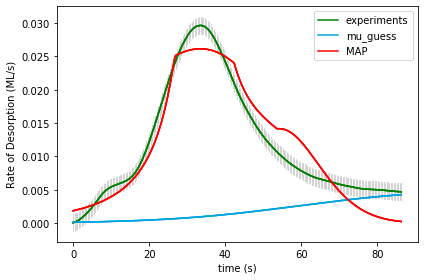
**Example 16a\_CPE\_grid\_mcmc: 243 points with 1 burn in and 1000 samplings each.**

**In example 17, I used the same things only now there is a total coverage dependence for the gamma’s effects on the activation energies of the two processes.**

**In Example 18 I used the coverage dependent gamma case.**

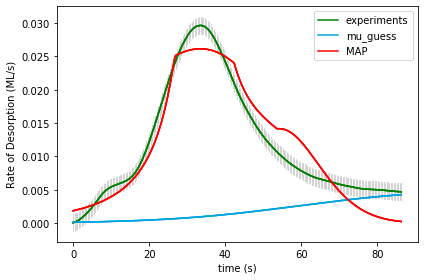
**18a\_CPE\_grid**

Final map results from gridsearch: (1.0, 0.0, 20.0, 11.0, -0.3, 0.3, -0.1, 0.1, 0.0, 0.0, -0.1) final logP: -1527.5171602395776



**18a\_BPE\_grid**

Final map results from gridsearch: (1.0, 0.0, 20.0, 11.0, -0.3, 0.3, -0.1, 0.1, 0.0, 0.0, -0.1) final logP: -1527.5828666259995



Not much better with optimization:

**18a\_CPE\_grid\_opt**

PE\_object.doGridSearch('getLogP', gridSamplingAbsoluteIntervalSize= [.1, 0.005, 20, 2, 0.3,

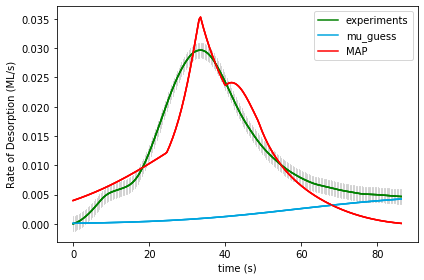
0.1, 0.1, 0.1, 0.1, 0.1, 0.1]

gridSamplingNumOfIntervals=[0,0,1,1,0, 1,0,1,0,1,0]

Final map results from gridsearch: [ 9.72403839e-01 2.45807149e-04 2.16720262e+01 1.17139968e+01

-2.39634576e-05 2.00249952e-01 6.62735737e-05 -1.06494051e-01

2.17505358e-05 -1.00161392e-01 1.44611048e-04] final logP: -3653.765327218868



**18a\_BPE\_grid\_opt**

PE\_object.doGridSearch('getLogP', gridSamplingAbsoluteIntervalSize= [.1, 0.005, 20, 2, 0.3,

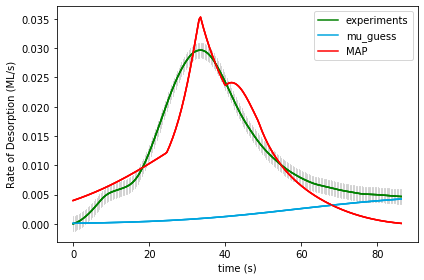
0.1, 0.1, 0.1, 0.1, 0.1, 0.1]

gridSamplingNumOfIntervals=[0,0,1,1,0, 1,0,1,0,1,0]

Final map results from gridsearch: [ 9.72403839e-01 2.45807149e-04 2.16720262e+01 1.17139968e+01

-2.39634576e-05 2.00249952e-01 6.62735737e-05 -1.06494051e-01

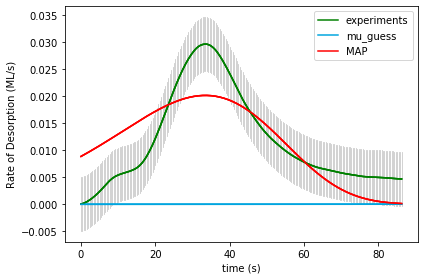
2.17505358e-05 -1.00161392e-01 1.44611048e-04] final logP: -3649.737299338728



Tried making 005 errors:

18a\_CPE\_grid\_005

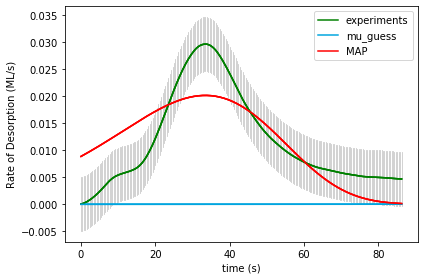
Final map results from gridsearch: (1.0, 0.0, 20.0, 11.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0) final logP: -3308.595108978507



Overall, the prior makes little difference during gridsearch. The reason is clear as we see a difference of <10 in the logP. Whereas between different gridpoints the logP can change (due to the likelihood) very easily by 10,000. So it is going to be only with optimization that there will be a difference. This makes sense, because with a gridsearch things can’t move very far away into “unlreastic” terroritory, and that’s where the prior starts to make a difference. Also, we know that the “good match” range is very sharp along a diagonal. So what we really want do is sequential parameter optimization, probably grid based. Scan a very fine grid with all of the others allowed to vary freely, that seems like the best option.

18a\_BPE\_grid\_005

Final map results from gridsearch: (1.0, 0.0, 20.0, 11.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0) final logP: -3302.634334851036



Fine grid worked

# PE\_object.doGridSearch('getLogP', gridSamplingAbsoluteIntervalSize=[0,0,2.0,1.0,0.1, 0,0,0,0,0,0], gridSamplingNumOfIntervals=[0,0,10,10,10, 0,0,0,0,0,0], passThroughArgs={"method":"Nelder-Mead", "maxiter":5000, "verbose":False})#, "maxiter":1000, "verbose":False})

It produced something that optimized to a good fit. So making an even finer grid now.

Final results from doOptimizeNegLogP: [ 1.23665574e+00 1.64772953e-03 3.32008913e+01 1.71466320e+01

2.56859423e-01 9.84680758e-02 -7.07252919e-02 -2.05013915e-03

-5.91049782e-02 -5.76832260e-02 -5.99951967e-02] final logP: 854.316750462401

