# Weekly Report
# June 18 - July 10

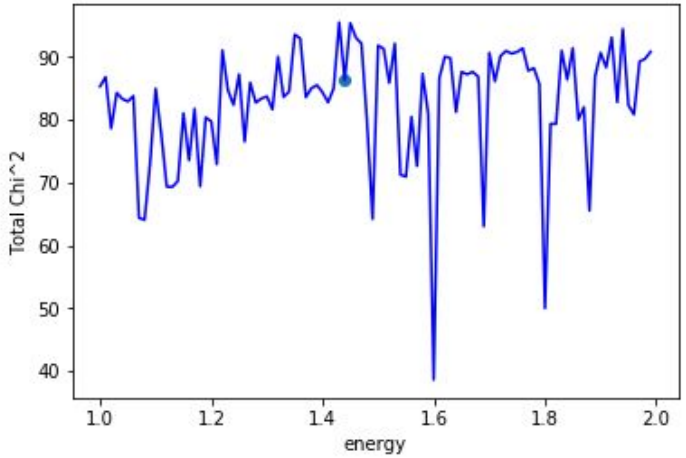
Christina

# 2D Plots of the chi^2 values - chi^2 vs. energy

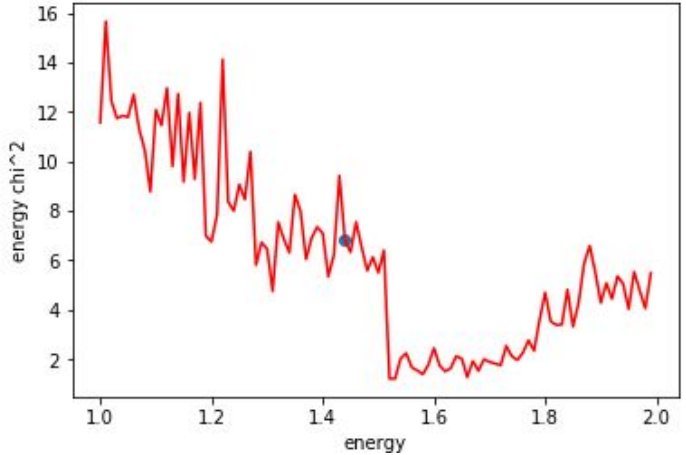
The energy chi^2 only counts towards a small part of the total chi^2 value.

Could not plot chi^2 with position because the code gives error message for some positions (because they are not physically possible?).

The distance between each minimum value is approximately 0.01 - used for Basinhopping later




```
Text(0,0.5,'energy chi^2')
```



# Looking at the cleaned data

The last column of the real data is how close each data point is to the unraveled spiral - if we can cut this distance even more we are able to clean out the noisy data points

The cleaned set of data has the name “cut_” in front of it

Right now the data points are cut off at distance=150mm; this number works well

```
#delete the points which are farther away from the center of the spiral
del_list = []

for i in range(len(xyzs)):
    if (xyzs[i,6]) > 150.0:
        del_list.append(i)
cut_xyzs = np.delete(xyzs,del_list,axis=0)
```

# Event 504

**Data with noise**

Monte-Carlo: 90.04

Conjugate Gradient: 101.06

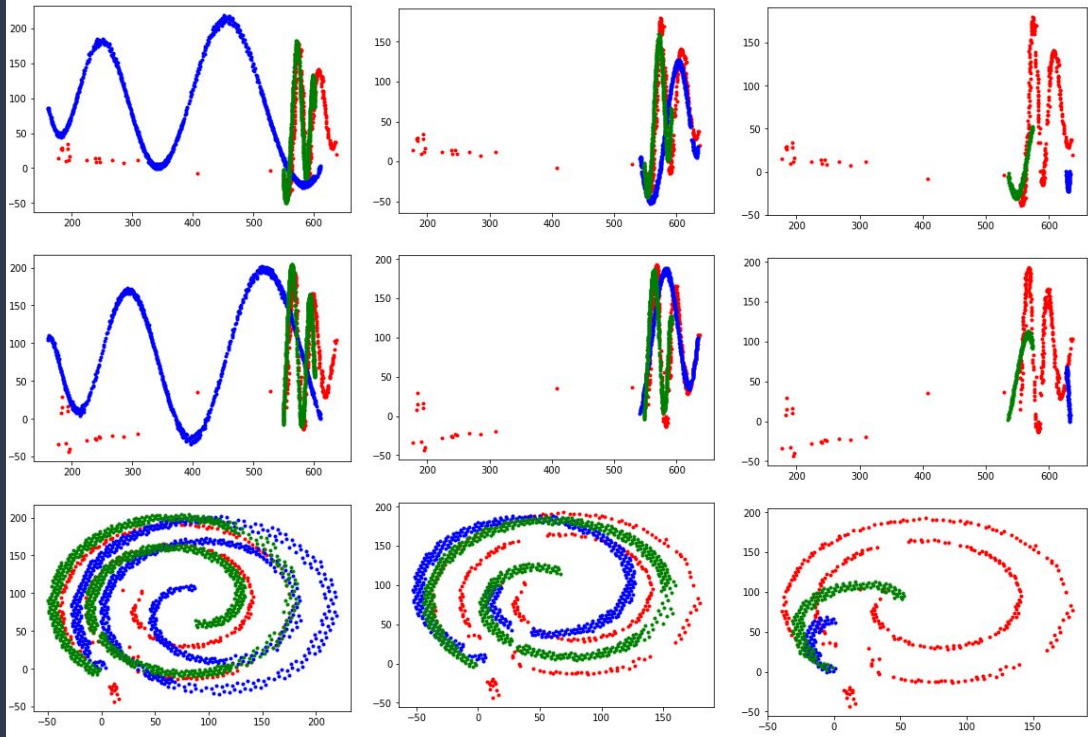
Differential Evolution: 68.06

**Data with reduced noise**

Monte-Carlo: 45.0

Conjugate Gradient: 93.57

Differential Evolution: 57.61



Monte Carlo | Diff Evolution | Conj Gradient

Red: real data (with noise)
Blue: fittings of data with noise
Green: fitting of data without noise

# Event 156

**Data with noise**

Monte-Carlo: 99.53

Conjugate Gradient: 106.07

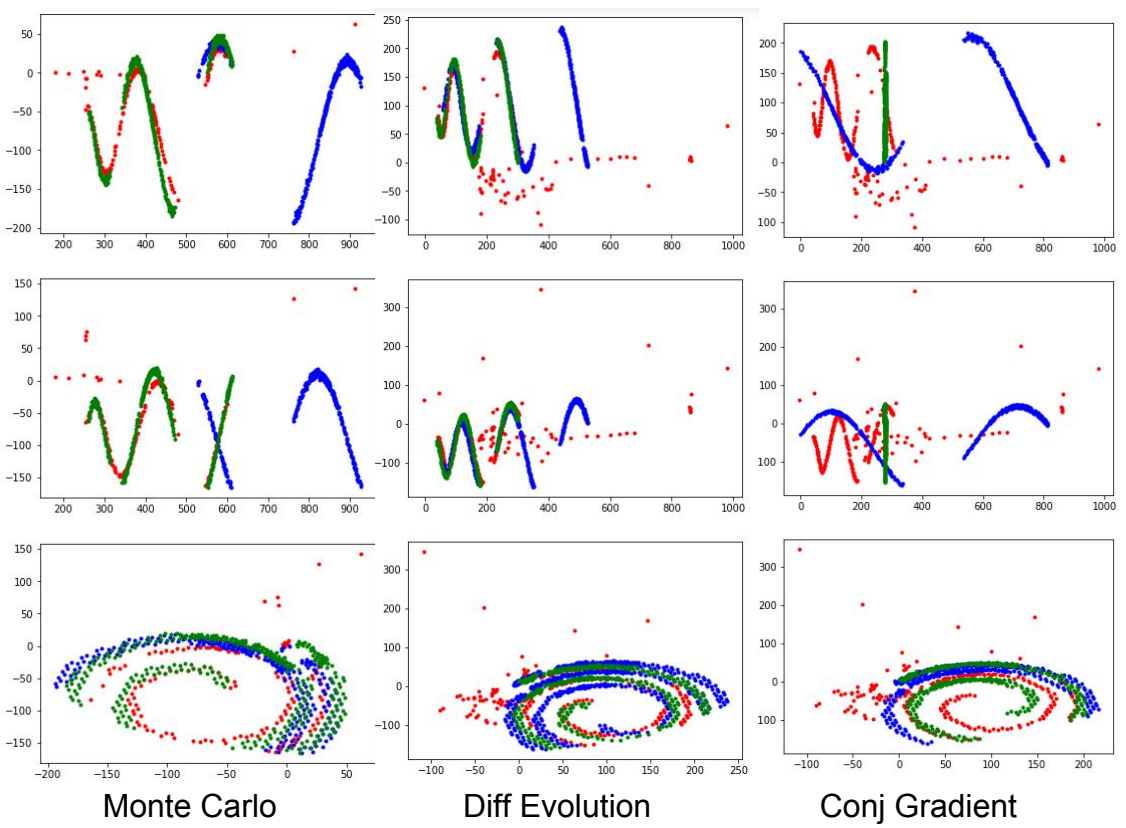
Differential Evolution: 55.12

**Data with reduced noise**

Monte-Carlo: 27.49

Conjugate Gradient: 57.74

Differential Evolution: 37.74




Red: real data (with noise)
Blue: fittings of data with noise
Green: fitting of data without noise

# Event 765

**Data with noise**

Monte-Carlo: 100.05

Conjugate Gradient: 106.98

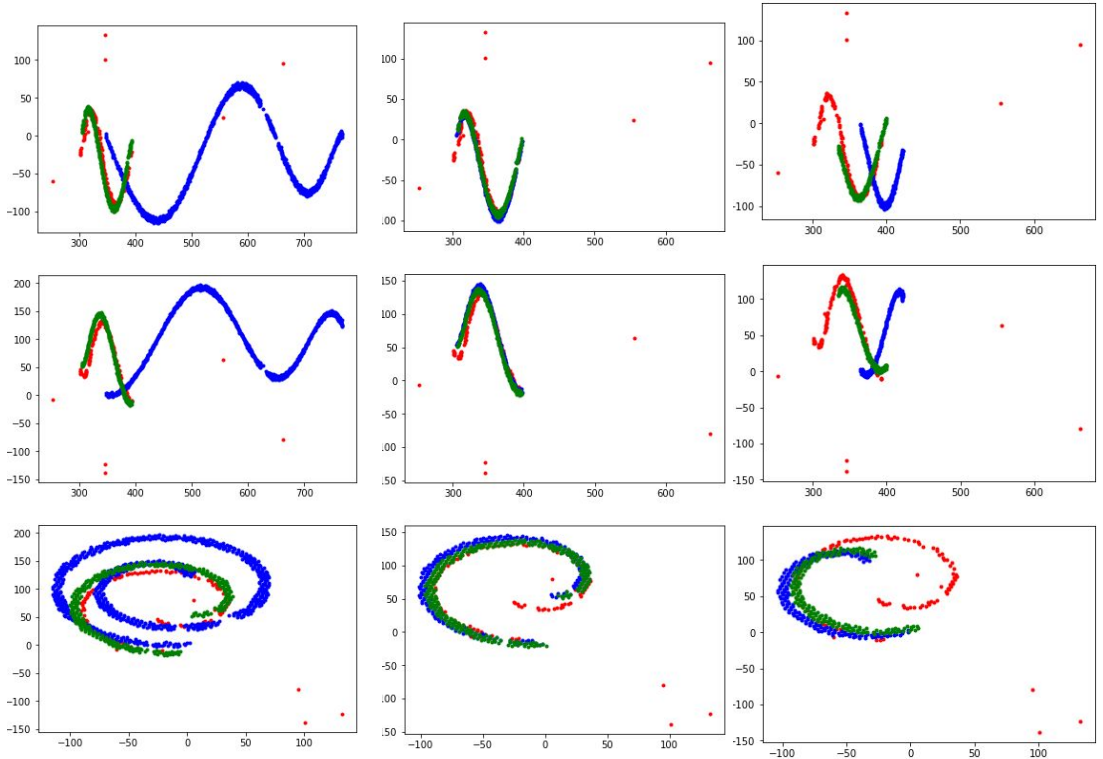
Differential Evolution: 35.45

**Data with reduced noise**

Monte-Carlo: 33.53

Conjugate Gradient: 65.87

Differential Evolution: 34.27



Monte Carlo | Diff Evolution | Conj Gradient

Red: real data (with noise)
Blue: fittings of data with noise
Green: fitting of data without noise

# Results so far

Plotted 2D and 3D graphs for each event (total of 20)

Monte Carlo's performance improves when the noise is cleaned; DE remains robust to noise.

DE still takes a very long time to run - eventually we use popsize=15, recombination=0.7 and random mutation within the range of (0.5, 1.5) to reduce the time to around 30 seconds per event (while not sacrificing the accuracy too much).

The histogram gives some information but we need more events to see a pattern (discussed later)

# Local Methods

| With Noise | 689 | 765 | 896 | 305 | 504 | 575 | 456 | 299 | 399 |
|---|---|---|---|---|---|---|---|---|---|
| Nelder-Mead | 103.4 | 104.9 | 99.4 | 24.7 | 100.16 | 105.5 | 39.06 | 71.26 | 53.67 |
| Powell | 93.4 | 102.6 | 87.8 | 17.2 | 76.4 | 91.8 | 32.6 | 85.3 | 108.1 |
| CG | 104.6 | 106.7 | 102.09 | 66.8 | 101.1 | 105.5 | 33.37 | 86.6 | 110.8 |
| BFGS | 104.6 | 106.9 | 99 | 21.4 | 101.05 | 105.5 | 35.8 | 86.5 | 110.4 |
| Newton | 104.6 | 106.5 | 100.3 | 67.7 | 101.05 | 105.5 | 33.6 | 77.79 | 110.8 |
| L-BFGS-B | 104.36 | 106.5 | 100.6 | 16.02 | 100.95 | 105.5 | 33.38 | 75.89 | 110.8 |
| TNC | 104.6 | 106.4 | 106.4 | 66.3 | 100.96 | 105.5 | 33.9 | 80.78 | 110.8 |
| COBYLA | 103.7 | 103.4 | 88.8 | 20.35 | 92.13 | 85.7 | 34.2 | 78.14 | 109.5 |
| SLSQP | 102.5 | 100.8 | 99.7 | 20.00 | 93.18 | 105.5 | 32.15 | 86.9 | 110.1 |
| DE | 59.3 | 35.01 | 15.59 | 14.2 | 68.21 | 40.57 | 31.09 | 84 | 60.0 |
| MC | 101.71 | 100.02 | 99.56 | 14.27 | 90.04 | 95.27 | 31.05 | 36.59 | 19.0 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Without Noise** | 689 | 765 | 896 | 305 | 504 | 575 | 456 | 299 |
| Nelder-Mead | 99.3 | 48.15 | 20.99 | 23.7 | 86.8 | 104.7 | 30.4 | 73.3 |
| Powell | 39.6 | 105.97 | 22.04 | 17.8 | 65.7 | 104.4 | 36.8 | 53.9 |
| CG | 100.7 | 101.9 | 23.5 | 95.1 | 87.5 | 105.8 | 104.18 | 73.8 |
| BFGS | 54.8 | 101.9 | 31.21 | 94.58 | 65.7 | 105.7 | 31.4 | 94.4 |
| Newton | 100.9 | 101.9 | 77.23 | 63.9 | 87.2 | 105.7 | 88.15 | 74.6 |
| L-BFGS-B | 100.7 | 101.9 | 23.4 | 95 | 86.8 | 105.7 | 29.15 | 69.5 |
| TNC | 100.35 | 101.9 | 23.4 | 44.8 | 72.2 | 105.7 | 26.8 | 73.4 |
| COBYLA | 96.5 | 103.4 | 35.8 | 20.3 | 87.8 | 91.1 | 98.8 | 76.8 |
| SLSQP | 52.4 | 88.8 | 21.3 | 17.72 | 100.8 | 105.5 | 23.5 | 95.4 |
| DE | 52.36 | 32.75 | 14.48 | 14.22 | 82.3 | 38.5 | 23.2 | 83.6 |
| MC | 33.17 | 33.53 | 14.45 | 14.29 | 45 | 39.3 | 23.0 | 23.01 |

# Some observations

Plotted all methods' fitting for event 299, 745 - different local methods give very different performances for each event, therefore we need to examine a lot of events as a whole

Local methods are less robust to noise but give good performance for cleaned data

They do not work as well as global methods, and some of them occasionally fail (when chi^2 = the chi^2 at the initial condition)

However, by examining how well the local methods work can help us improve the accuracy of the Basinhopping method (discussed later)

# Basinhopping

# A Markov Chain Monte Carlo Method

Choose a starting point (ctr0)

Compute a local minimum (with a user-defined local method)

Apply a random perturbation to the coordinates of the local minimum

accept/reject the new coordinates based on Metropolis criteria of Monte Carlo algorithms (random walk Monte Carlo)

Compute the next local minimum

Update the global minimum to the lowest value

# Function parameters

scipy.optimize.basinhopping(f,ctr0, niter=10, T=0.01, stepsize=0.05, minimizer_kwargs={"method": "CG":})

T is set to be the distance between local minima, in our case 0.01

The step is chosen uniformly in the region from x0-stepsize to x0+stepsize, in each dimension. If the step size is not specified, the function will calculate the optimized step size, but will only do so at a large number of iteration. So far a step size of 0.05 is giving promising results, but there is still chance to improve.

“Method” allows us to choose the best local method (experiments required)

# Global Methods.ipynb

Evaluates all three global methods over 28 events

Parameters used for each function:

scipy.optimize.differential_evolution(f, bounds, maxiter=1000, strategy='best1bin', recombination=0.7, popsize=15, mutation=(0.5,1.5))

scipy.optimize.basinhopping(f,ctr0, niter=10, T=0.01, stepsize=0.05, minimizer_kwargs={"method": method})

# Finally... Results!

**Monte Carlo**

```
average time: 0.48798972368240356 seconds
average chi2 value (with noise): 61.584411066315134
average chi2 value (without noise): 31.393359085475915
```

**Differential Evolution**

```
average time: 33.156591317483354 seconds
average chi2 value (with noise): 42.02732836605447
average chi2 value (without noise): 42.00646858265847
```

**Basinhopping**
**niter = 10**

```
average time for Nelder-Mead: 7.100889333656856 seconds
chi2 value (with noise) for Nelder-Mead: 56.602099562368004
chi2 value (without noise) for Nelder-Mead: 36.739391372534904
average time for Powell: 15.809054085186549 seconds
chi2 value (with noise) for Powell: 72.39768213081769
chi2 value (without noise) for Powell: 48.439384824847956
average time for CG: 5.473999002150127 seconds
chi2 value (with noise) for CG: 85.06981919655972
chi2 value (without noise) for CG: 68.7291566328876
average time for BFGS: 6.69640759059361 seconds
chi2 value (with noise) for BFGS: 71.78550302526396
chi2 value (without noise) for BFGS: 57.42162601972965
average time for L-BFGS-B: 5.841121426650456 seconds
chi2 value (with noise) for L-BFGS-B: 81.43751391791639
chi2 value (without noise) for L-BFGS-B: 66.45097907673568
average time for TNC: 6.563905213560377 seconds
chi2 value (with noise) for TNC: 74.96609741576006
chi2 value (without noise) for TNC: 58.198438073015446
average time for COBYLA: 1.666680829865592 seconds
chi2 value (with noise) for COBYLA: 78.49874464606329
chi2 value (without noise) for COBYLA: 66.19309393961564
average time for SLSQP: 2.873416449342455 seconds
chi2 value (with noise) for SLSQP: 62.756947144902846
chi2 value (without noise) for SLSQP: 37.88723205549305
```

# Basinhopping (ninter = 25)

```
average time for Nelder-Mead: 20.268136343785695 seconds
chi2 value (with noise) for Nelder-Mead: 49.02851080971168
chi2 value (without noise) for Nelder-Mead: 30.06558285392754
average time for Powell: 36.70845403841564 seconds
chi2 value (with noise) for Powell: 59.04416700450565
chi2 value (without noise) for Powell: 47.49919979627662
average time for BFGS: 16.16957257475172 seconds
chi2 value (with noise) for BFGS: 68.93500843710063
chi2 value (without noise) for BFGS: 46.31901726906612
average time for SLSQP: 6.993133834430149 seconds
chi2 value (with noise) for SLSQP: 63.31442266173478
chi2 value (without noise) for SLSQP: 33.89991591501348
```

## Monte Carlo

```
average time: 0.48798972368240356 seconds
average chi2 value (with noise): 61.584411066315134
average chi2 value (without noise): 31.393359085475915
```

## Differential Evolution

```
average time: 33.156591317483354 seconds
average chi2 value (with noise): 42.02732836605447
average chi2 value (without noise): 42.00646858265847
```

# Nelder-Mead

does not require any derivative information - suitable for problems with non-smooth functions.

Simplex-based method: a simplex of n dimension has n+1 vertices

The method then performs a sequence of transformations of the working simplex S, aimed at decreasing the function values at its vertices

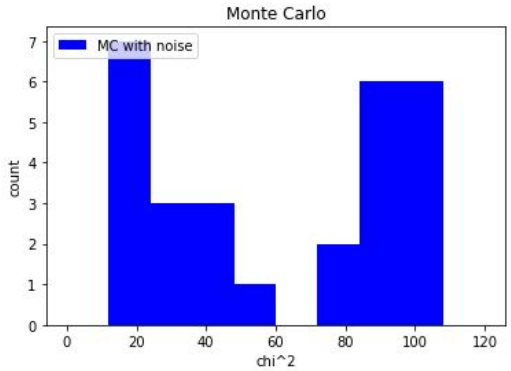
Only requires one of two function evaluations at each step

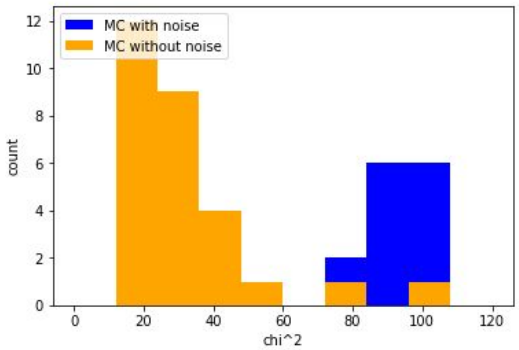
# sequential quadratic programming algorithm

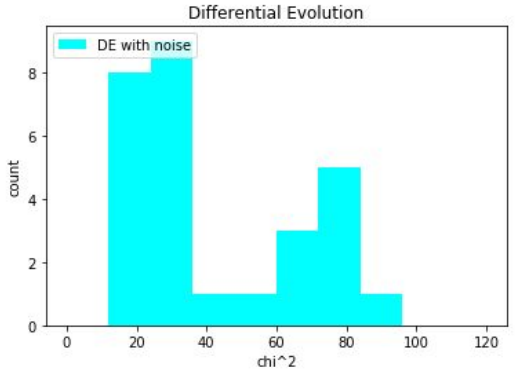
Uses the Han–Powell quasi–Newton method with a BFGS update of the B–matrix and an L1–test function in the step–length algorithm

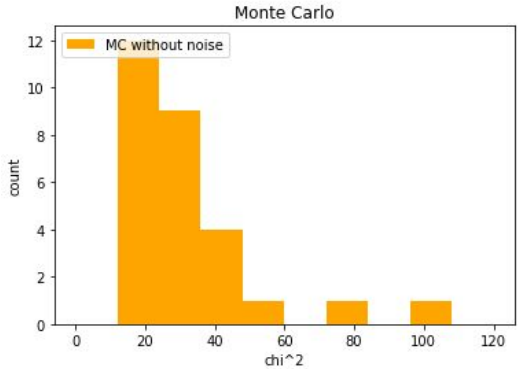
“SQP methods solve a sequence of optimization subproblems, each of which optimizes a quadratic model of the objective subject to a linearization of the constraints. If the problem is unconstrained, then the method reduces to Newton's method for finding a point where the gradient of the objective vanishes.”

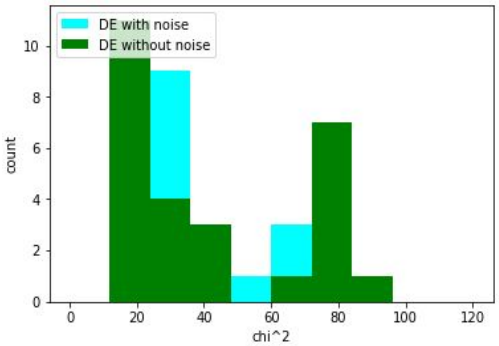
# Histograms

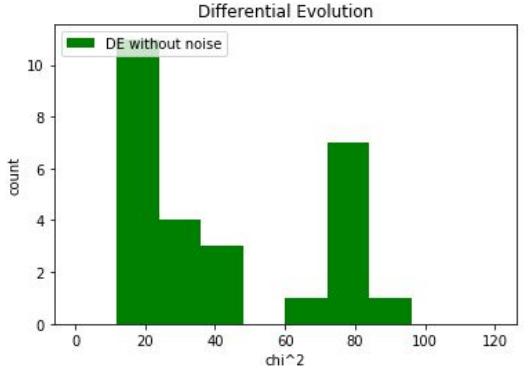

Monte Carlo
MC with noise
count
chi^2

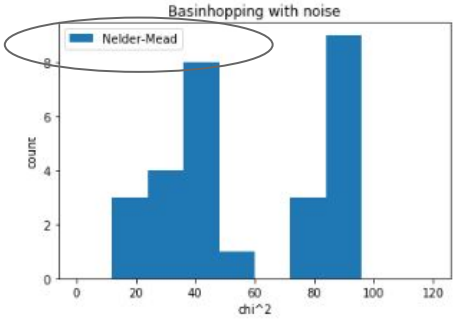

MC with noise
MC without noise
count
chi^2

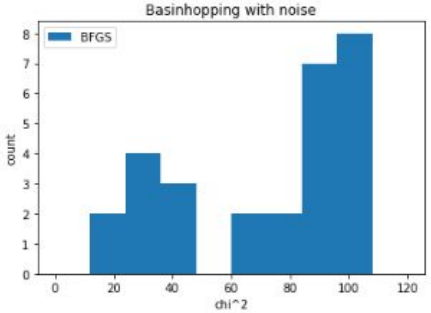

Differential Evolution
DE with noise
count
chi^2

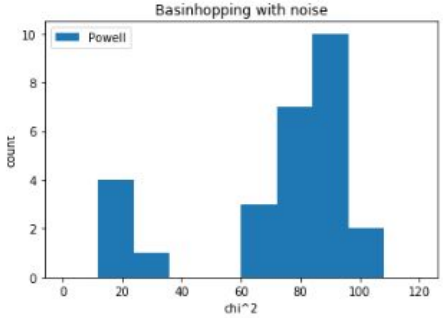

Monte Carlo
MC without noise
count
chi^2

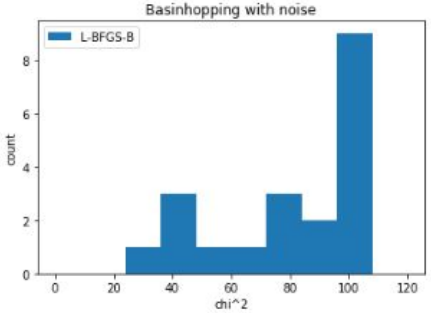

DE with noise
DE without noise
count
chi^2

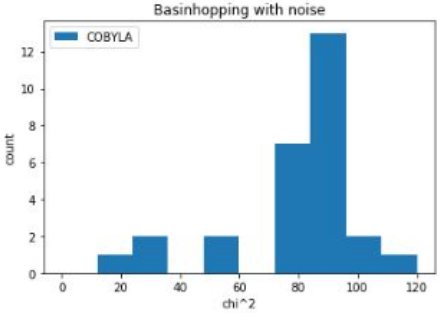

Differential Evolution
DE without noise
count
chi^2

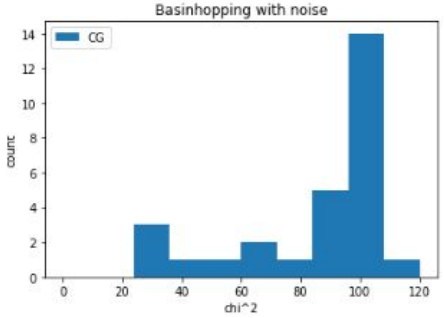

Basinhopping with noise
Nelder-Mead
count
chi^2

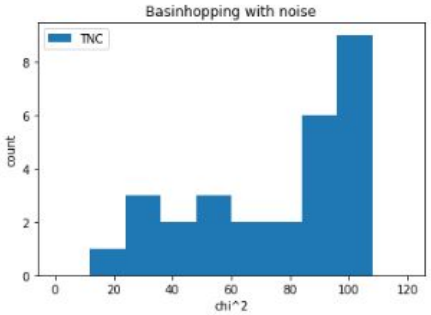

Basinhopping with noise
BFGS
count
chi^2

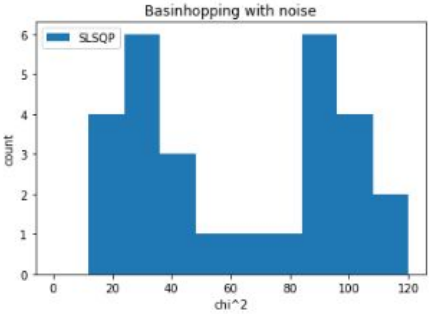

Basinhopping with noise
Powell
count
chi^2

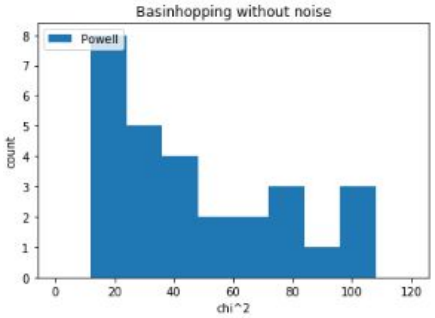

Basinhopping with noise
L-BFGS-B
count
chi^2

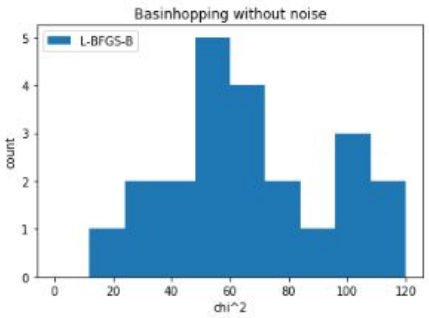

Basinhopping with noise
COBYLA
count
chi^2

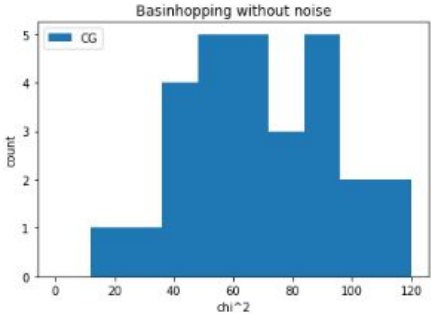

Basinhopping with noise
CG
count
chi^2

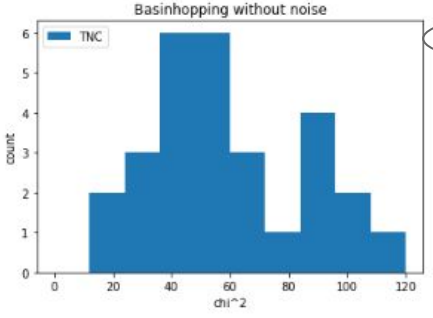

Basinhopping with noise
TNC
count
chi^2

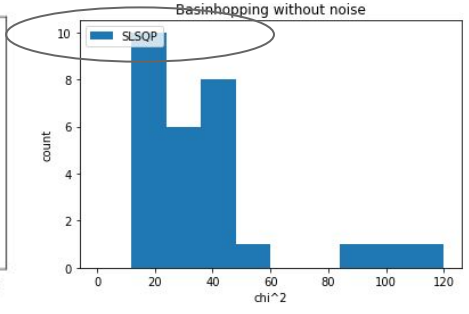

Basinhopping with noise
SLSQP
count
chi^2

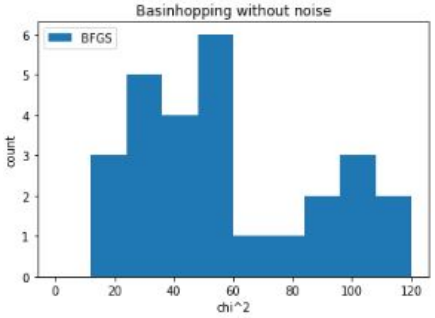

Basinhopping without noise
Powell
count
chi^2

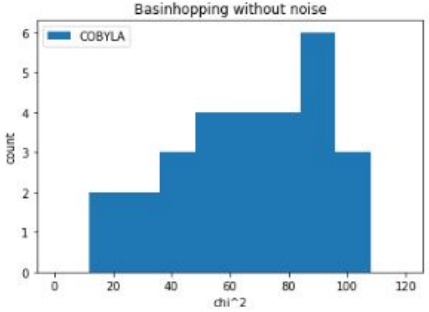

Basinhopping without noise
L-BFGS-B
count
chi^2

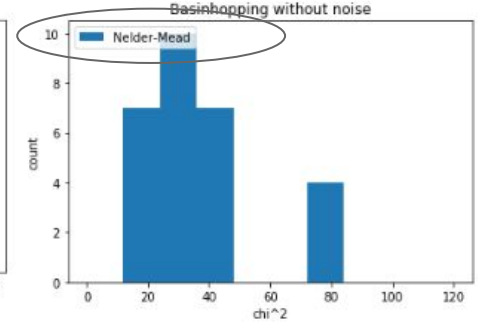

Basinhopping without noise
CG
count
chi^2


Basinhopping without noise
TNC
count
chi^2


Basinhopping without noise
SLSQP
count
chi^2


Basinhopping without noise
BFGS
count
chi^2


Basinhopping without noise
COBYLA
count
chi^2


Basinhopping without noise
Nelder-Mead
count
chi^2