Weekly Report
June 21-June 27

Christina

Monte-Carlo Simulation

Some improvements:

1. Preprocess function: added rotate_pads=True for the simulated data only
    a. The real & simulated data are now on the same coordinates
2. Commented out the “process shifted z by 38mm due to trigger delay” command in the preprocess function
    a. Rather nice fitting for event_305
    b. However, some events who do go through trigger delays now experience a shift in z-direction

Gradient Descent - a local optimization method

Gradient Descent Principles

1. We choose a starting point within a function , and then take steps towards the direction with negative (approximate) gradient

2. Conjugate gradient: used to approximate the answer x to the equation Q**x** = **b**, where Q is a symmetrical matrix -> transformed to this form from our objective equation

    a. Let {d0,...,dn-1} be a set of nonzero Q-orthogonal vectors, the sequence $x_{k+1} = x_k + a_k * d_k$ (where a is step size) converges to a unique solution Q**x** = **b** that minimizes the function.

Chi-square function

1. The function we used to evaluate the chi-square value is minimizer.run_tracks

    a. Takes in our initial guess (in the form of a 2D array), the experimental data points (x y z) and the experimental hit pattern (pad amplitudes indexed by pad number).

    b. Returns an array consisting of the three chi-square values (position, energy, vertex components)

Scipy function construction

1. We define a function f(y) that can be accepted by Scipy.optimize functions

```
def f(y): #accept y as a vector consisting of six parameters
    ctr = np.zeros([1,6])
    ctr[0] = y #transform y to a 2D array acceptable by run_tracks function
    chi_result = minimizer.run_tracks(ctr, exp_pos, exp_hits)
    return sum(chi_result[0]) #We want to minimize the sum
```

Scipy function construction

2. We define the jacobian with a numerical derivative function scipy.optimize.approx_fprime so we can implement it into our optimization function to get a more accurate result

3. We then create a callback function (callbackF) which prints out the intermediate result of each iteration when we call the optimization function

4. Run the optimization function

    a. results = scipy.optimize.minimize(f #object function, ctr0 #initial guess, method="CG", jac=jacobian, callback=callbackF, options={'eps':1e-13} #step size)

Differential evolution - a global optimization method

Differential Evolution Principles

1. The method iteratively try to improve an existing candidate solution -> stochastic method, the best solution is not guaranteed to be found
2. Starts with a “population” of candidate solutions - points with random positions in the search space.
3. Pick two distinct candidates (a, b) from the population
4. Compute the candidate’s potentially new position (aka trial candidate):
    a. We want to “mutate” the best candidate c0
    b. C’ = c0+mutation*(a-b)
    c. For ‘best1bin’ strategy, we take a random number in [0,1) and if it is less than the recombination constant (explained later) the parameters is loaded from c’ (else it’s loaded from c0).
    d. If the trial candidate c’ is better than the original candidate c0, it replaces c0.

Scipy function parameters

1. Bounds: (min, max) pairs for each element in our input vector; defines search space.
2. Maxitr: # of iterations allowed
3. Popsize: population size. Default is 15; larger size improves chance of convergence
4. Mutation: a float or a range - larger mutation value increases the search radius (“step size”) but will slow down convergence. If mutation is given a range, the constant is randomly changed per iteration to increase speed.
5. Recombination constant: float in [0,1]. Increased RC will allow more mutants to become trial candidates, but will decrease population stability

Results

Event #305

1. Conjugate Gradient chi = 24.713

```
results = scipy.optimize.minimize(f, ctr0, method="CG", jac=jacobian, callback=callbackF, options={'gtol': 30.0, 'eps':1e-1
cg = results.x
```

```
302 0.000360-0.000141 0.734463 2.614517-3.002859 1.951079 24.713554
303 0.000360-0.000141 0.734463 2.614517-3.002859 1.951079 24.713554
304 0.000360-0.000141 0.734463 2.614517-3.002859 1.951079 24.713554
305 0.000360-0.000141 0.734463 2.614517-3.002859 1.951079 24.713554
306 0.000360-0.000141 0.734463 2.614517-3.002859 1.951079 24.713554
307 0.000360-0.000141 0.734463 2.614517-3.002859 1.951079 24.713554
308 0.000360-0.000141 0.734463 2.614517-3.002859 1.951079 24.713554
309 0.000360-0.000141 0.734462 2.614516-3.002859 1.951079 24.713481
310 0.000360-0.000141 0.734462 2.614516-3.002858 1.951079 24.713390
311 0.000360-0.000141 0.734462 2.614515-3.002858 1.951079 24.713326
312 0.000360-0.000141 0.734462 2.614515-3.002858 1.951079 24.713326
313 0.000360-0.000141 0.734462 2.614515-3.002858 1.951079 24.713326
314 0.000360-0.000141 0.734462 2.614515-3.002858 1.951079 24.713326
315 0.000360-0.000141 0.734462 2.614515-3.002858 1.951079 24.713326
316 0.000360-0.000141 0.734462 2.614515-3.002858 1.951079 24.713326
317 0.000360-0.000141 0.734462 2.614515-3.002858 1.951079 24.713326
318 0.000360-0.000141 0.734462 2.614515-3.002858 1.951079 24.713326
319 0.000360-0.000141 0.734462 2.614515-3.002858 1.951079 24.713326
320 0.000360-0.000141 0.734462 2.614515-3.002858 1.951079 24.713326
```

2. Monte-Carlo

```
uvw, (cu, cv) = mcfitter.preprocess(xyzs[:,0:5], center=(cx, cy), rotate_pads=False) # get calibrated set of data
uvw_values = uvw.values #transform pd file to arrays
print(cu,cv)
mcres, minChis, all_params, good_param_idx = mcfitter.process_event(uvw, cu, cv, return_details=True)

print(mcres)
```

```
-43.91625428724363 118.39379347393248
{'x0': 0.004871498439906168, 'y0': -0.011977943314897467, 'z0': 0.7347184090083536, 'enu0': 2.8186679462143145, 'azi0': -
2.920989043120735, 'pol0': 1.9263176741477495, 'posChi2': 3.97700712247824, 'enChi2': 7.1981754856024605, 'vertChi2': 3.3
44052462098103, 'lin_scat_ang': 1.2420273973461806, 'lin_beam_int': 720.0566082022592, 'lin_chi2': 36.34021928440234, 'ra
d_curv': 117.8081237563669, 'brho': 0.20911788771969209, 'curv_en': 2.0944258199546852, 'curv_ctr_x': -43.91625428724363,
'curv_ctr_y': 118.39379347393248}
```

Chi = 3.977+7.198+3.344 = 14.519

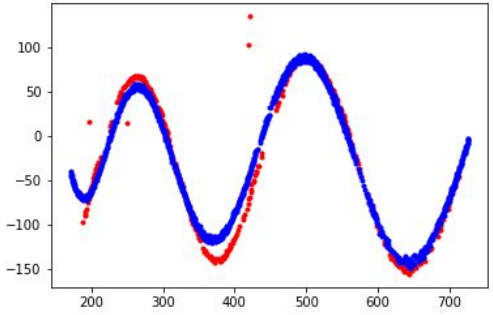
3. Differential evolution

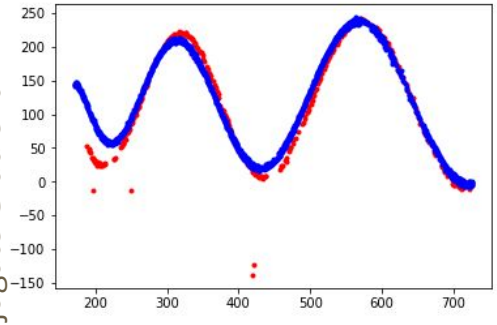
chi=14.208

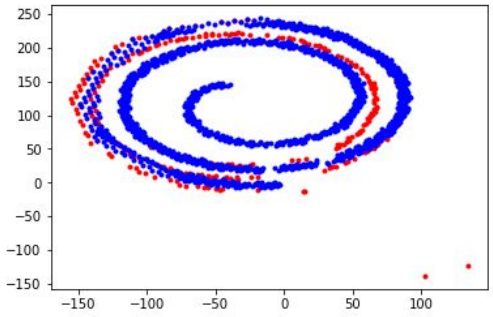
```
i = 1
def callbackF(x, convergence=10):
    global i
    print('{0:4d}{1: 3.6f}{2: 3.6f}{3: 3.6f}{4: 3.6f}{5: 3.6f}{6: 3.6f}{7: 3.6f}'.format(i,x[0],x[1],x[2],x[3],x[4],x[5],
    i += 1
bounds = [(-1,1), (-1, 1), (0, 1), (0,5), (-2 * pi, 2 * pi), (-2 * pi, 2 * pi)]
results = scipy.optimize.differential_evolution(f, bounds, callback=callbackF, maxiter=10000, strategy='best1bin',\
                                                recombination=0.8, popsize=15, mutation=(0.5, 1.0))
'''    The differential evolution strategy should be one of:
            'best1bin' - shorter time but usually needs to run multiple times to get the best result
            'best1exp'- shorter time but usually needs to run multiple times to get the best result
            'rand1exp' - takes a long time but converges
            'randtobest1exp' - takes a long time but converges
            'currenttobest1exp' - takes a long time but converges
            'best2exp'
            'rand2exp'
            'randtobest1bin'
            'currenttobest1bin'
            'best2bin'
            'rand2bin'
            'rand1bin'
    The default is 'best1bin'.
'''
```

```
 94 0.003659-0.012275 0.733378 2.809433 0.211693-1.926856 14.577127
 95 0.003659-0.012275 0.733378 2.809433 0.211693-1.926856 14.577127
 96 0.003659-0.012275 0.733378 2.809433 0.211693-1.926856 14.577127
 97 0.002564-0.011759 0.733055 2.809754 0.207638-1.926092 14.260398
 98 0.002564-0.011759 0.733055 2.809754 0.207638-1.926092 14.260398
 99 0.002564-0.011759 0.733055 2.809754 0.207638-1.926092 14.260398
100 0.002564-0.011759 0.733055 2.809754 0.207638-1.926092 14.260398
101 0.002564-0.011759 0.733055 2.809754 0.207638-1.926092 14.260398
102 0.002564-0.011759 0.733055 2.809754 0.207638-1.926092 14.260398
103 0.002564-0.011759 0.733055 2.809754 0.207638-1.926092 14.260398
104 0.002306-0.011989 0.732534 2.808862 0.204325-1.926166 14.207878
105 0.002306-0.011989 0.732534 2.808862 0.204325-1.926166 14.207878
106 0.002306-0.011989 0.732534 2.808862 0.204325-1.926166 14.207878
```

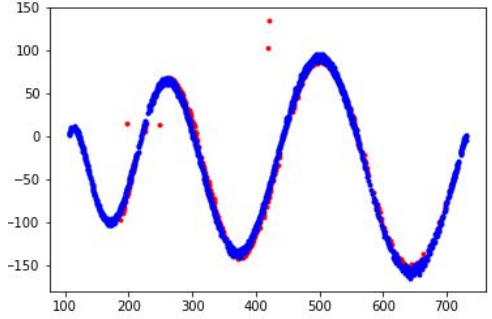
Conjugate Gradient

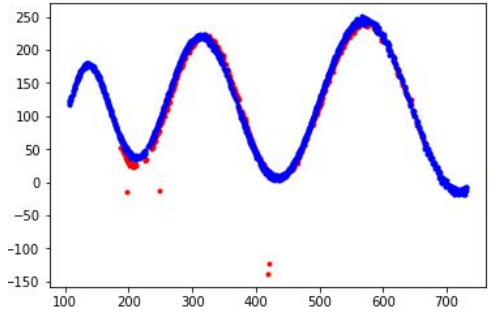


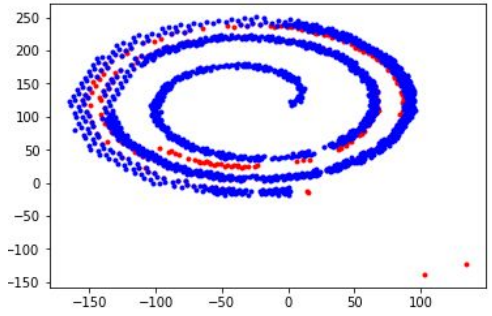




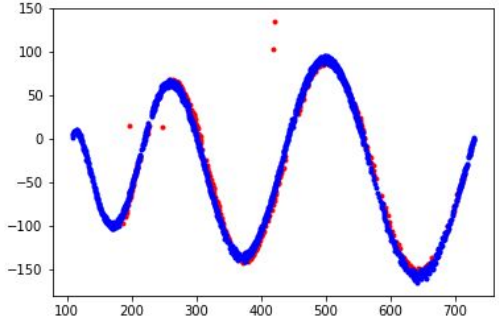
Monte Carlo

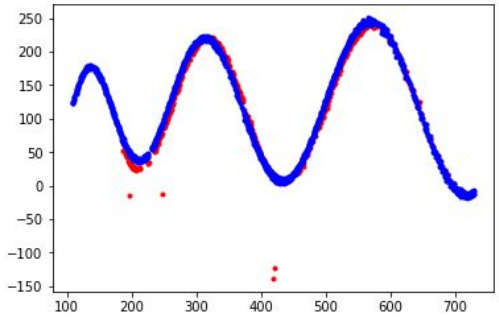


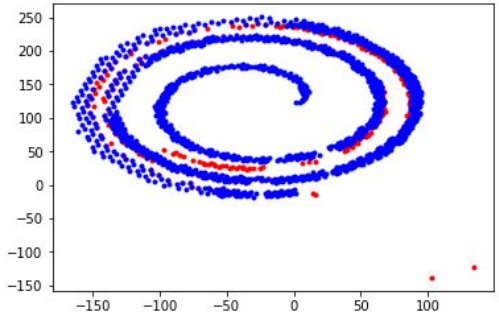




Differential Evolution







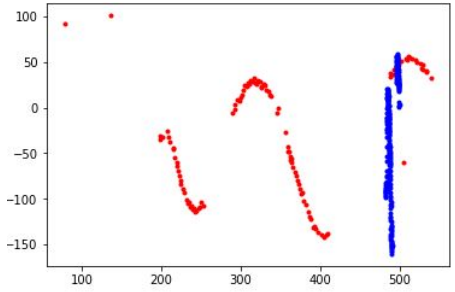
Event #456

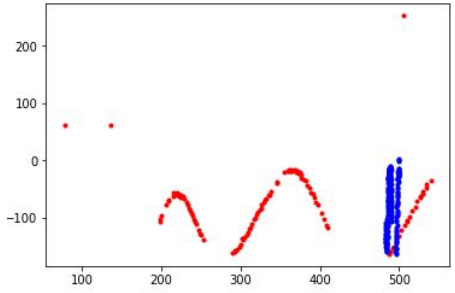
Monte Carlo: chi = 12.365+7.088+0.033=19.486

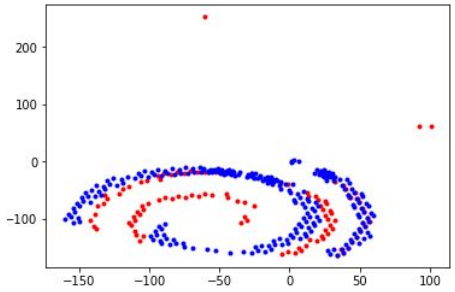
Gradient: 110.8 (fail)

Differential evolution: with population size=30, recombination constant = 0.6, chi= 27.32

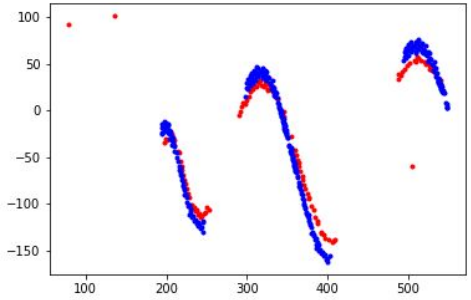
Conjugate Gradient

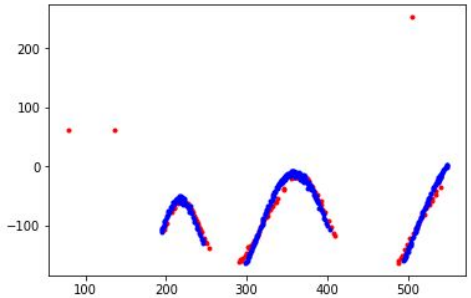


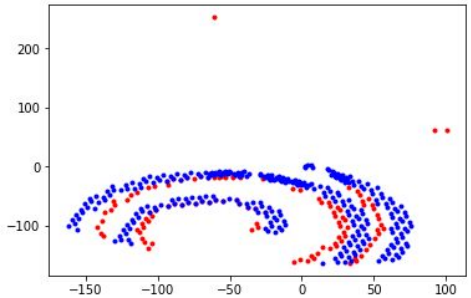




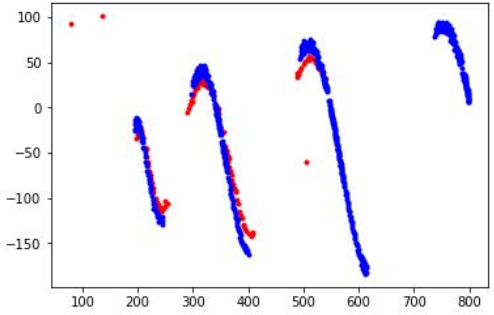
Monte Carlo

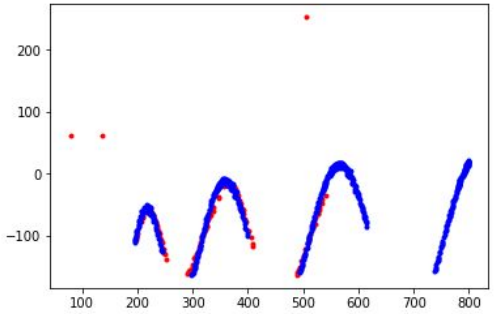


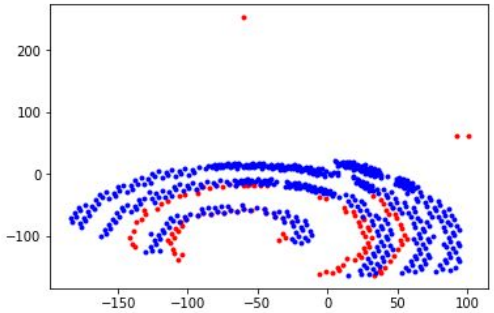




Differential Evolution







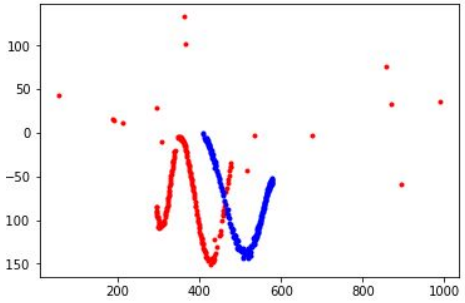
Event 689

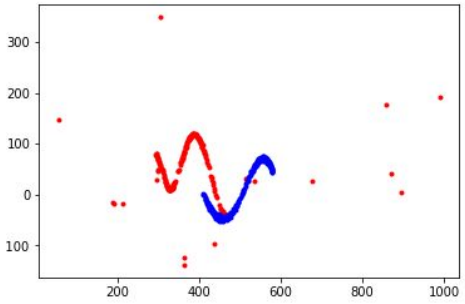
Monte-Carlo: chi = 94.97+6.71+0.03 = 101.71

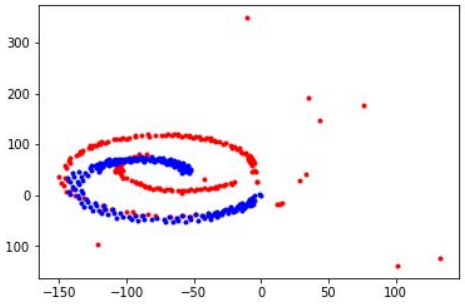
Conjugate Gradient: chi=104.62 (failed)

Differential evolution: chi=37.7

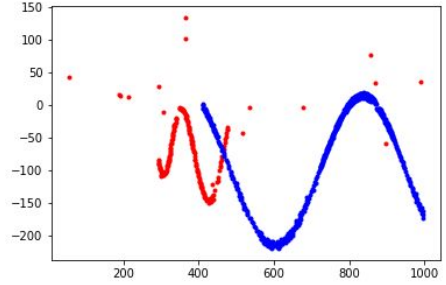
Conjugate Gradient

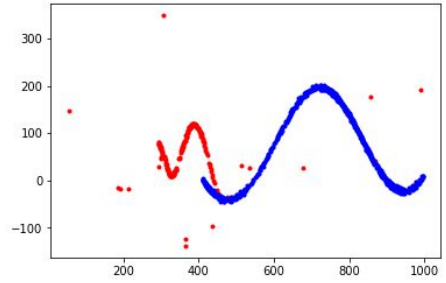


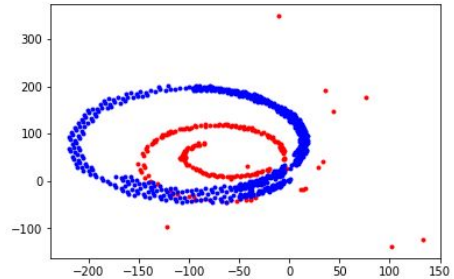




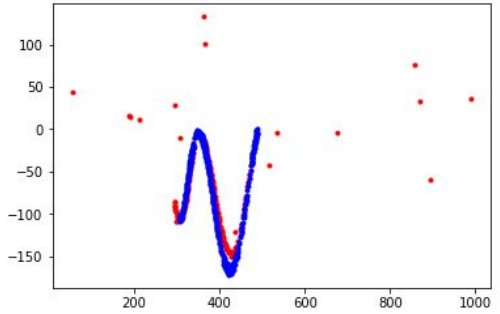
Monte Carlo

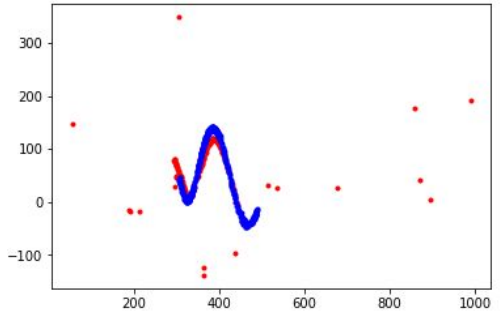


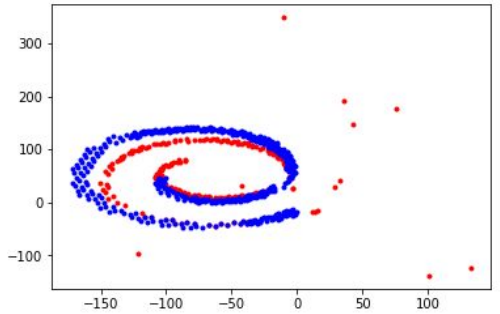




Differential Evolution







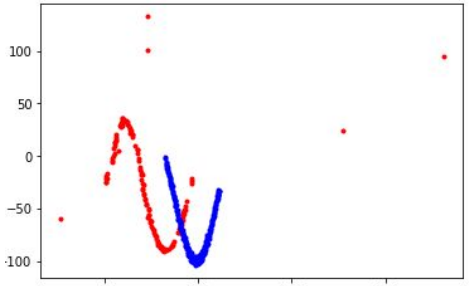
Event 765

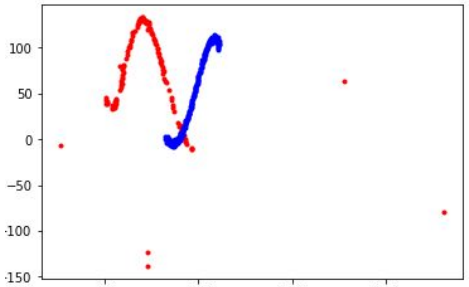
Monte-Carlo: chi = 88.75+11.09+0.21 = 100.05

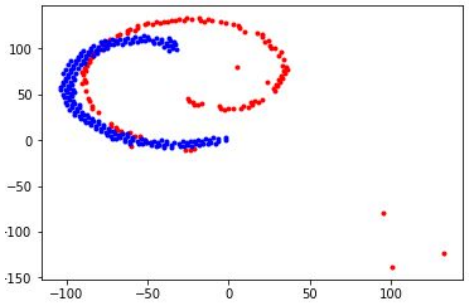
Conjugate Gradient: chi=106.98 (failed)

Differential evolution: chi=35.06

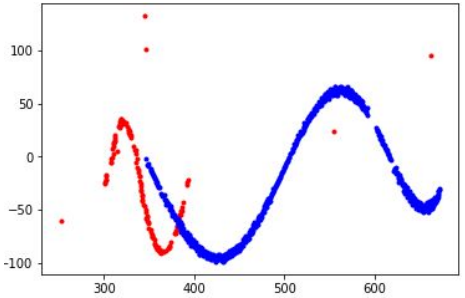
Conjugate Gradient

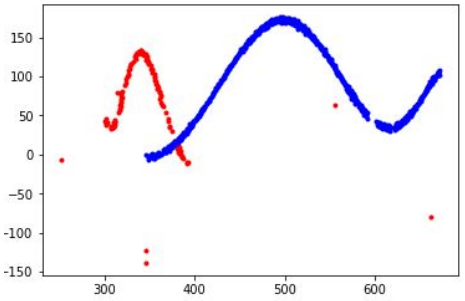


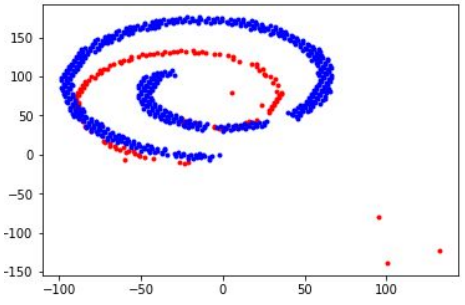




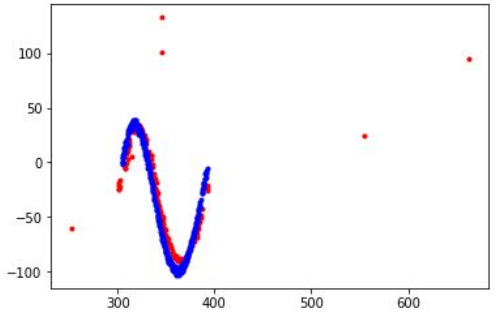
Monte Carlo

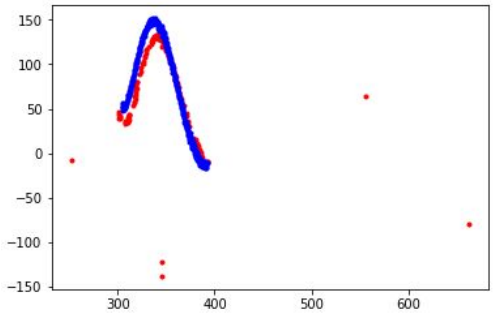


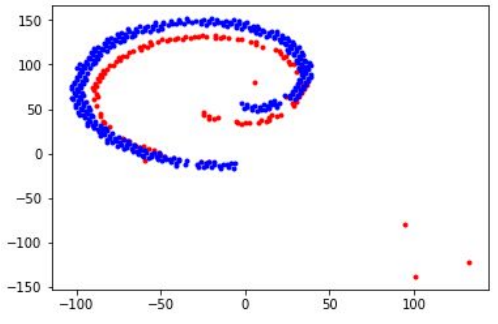




Differential Evolution







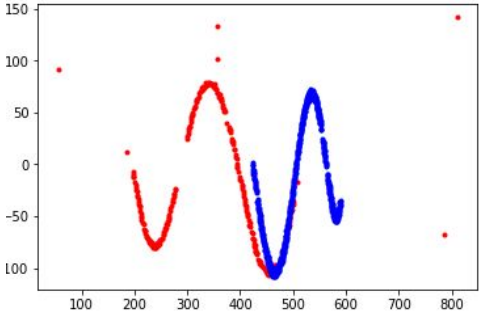
Event 896

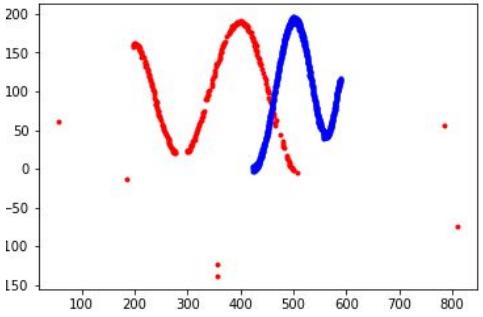
Monte-Carlo: chi = 95.92+3.626+0.04 = 99.59

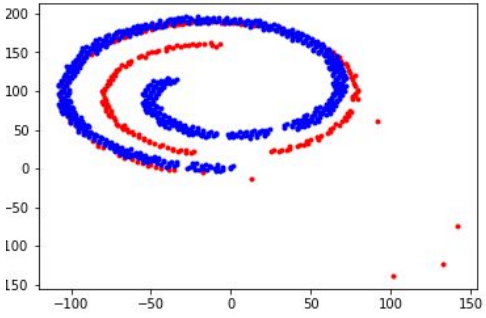
Conjugate Gradient: chi=102.09

Differential evolution: chi=15.58

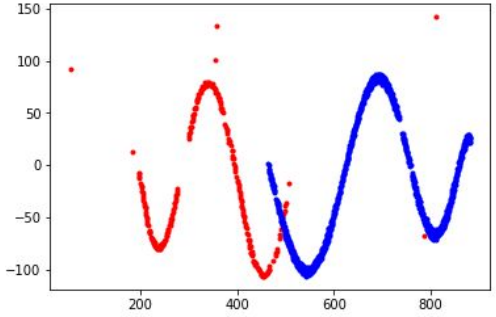
Conjugate Gradient

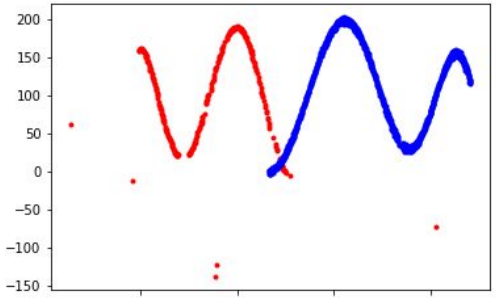


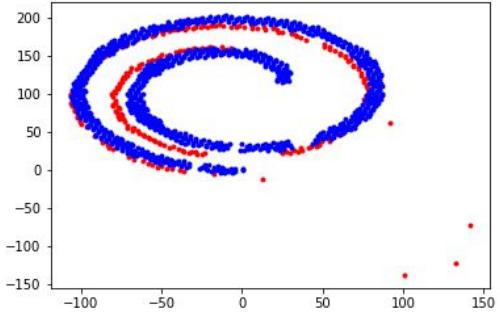




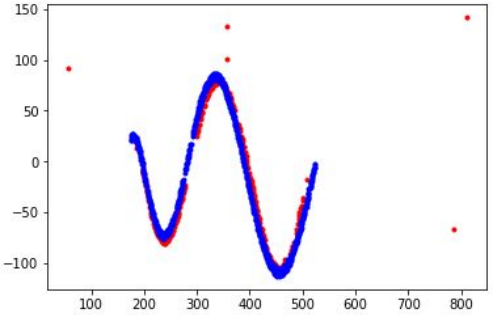
Monte Carlo

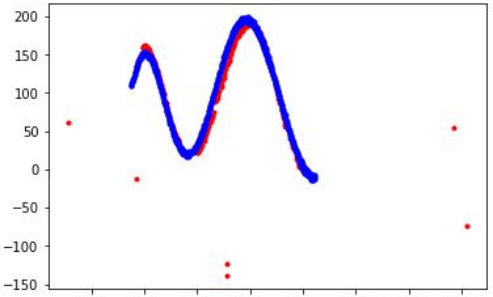


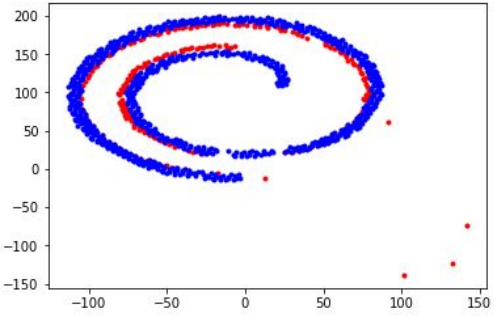




Differential Evolution







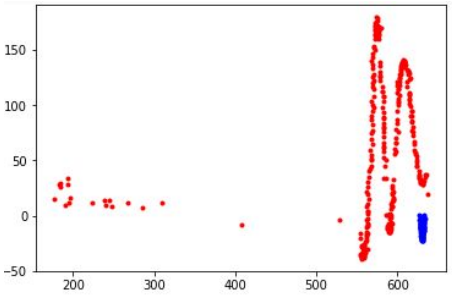
Event 504

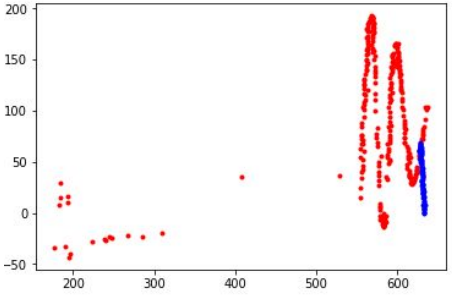
Monte-Carlo: chi = 88.79+1.211+0.04 = 90.041

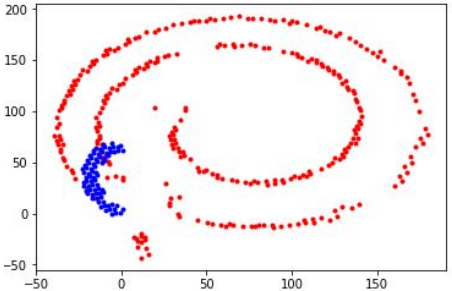
Conjugate Gradient: chi=101.06

Differential evolution: chi=53.08

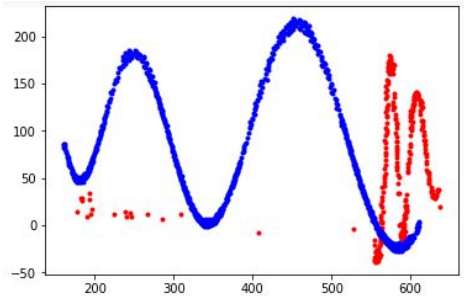
Conjugate Gradient

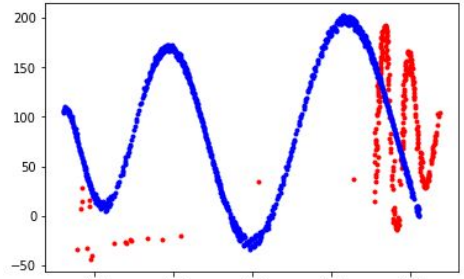


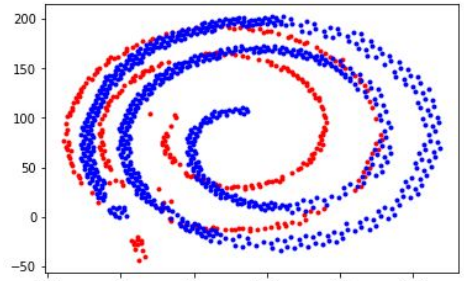




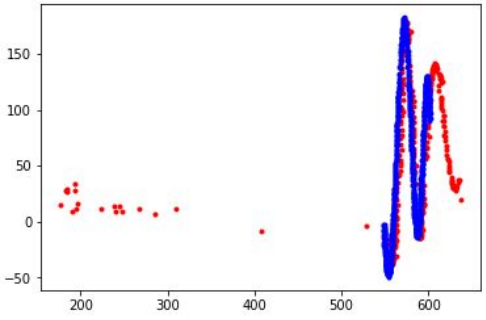
Monte Carlo

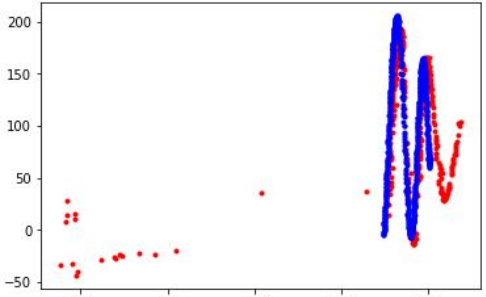


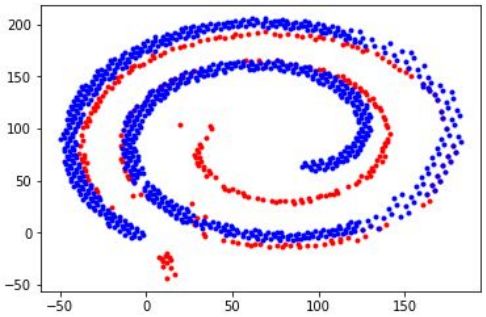




Differential Evolution







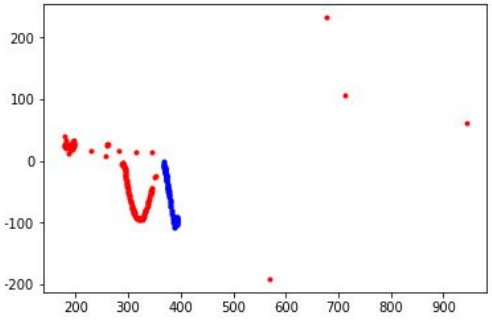
Event 575

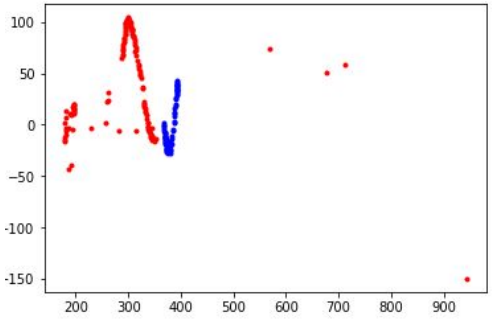
Monte-Carlo: chi = 89.9+5.044+0.037 = 94.98

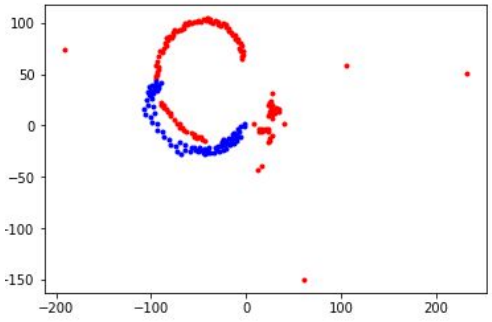
Conjugate Gradient: chi=105.539 (failed)

Differential evolution: chi=40.36

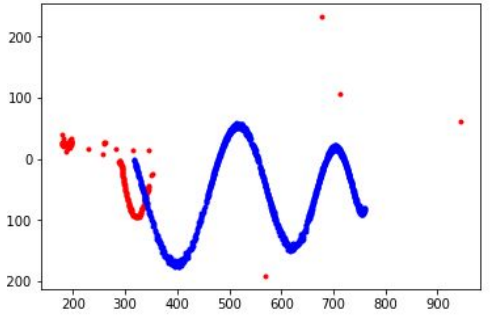
Conjugate Gradient

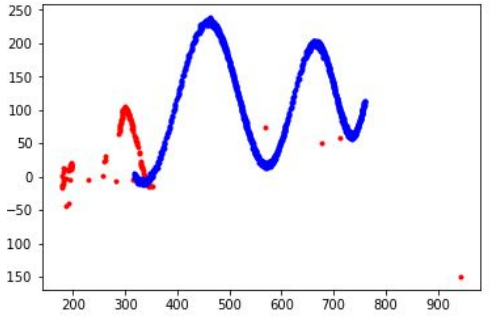


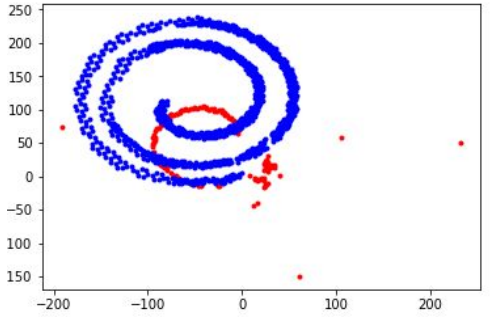




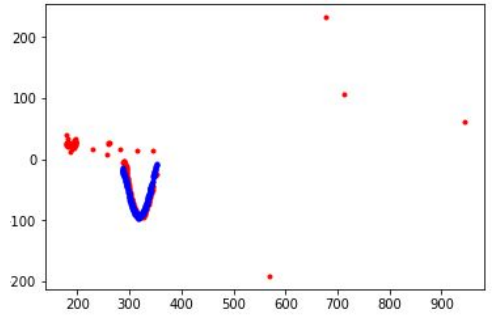
Monte Carlo

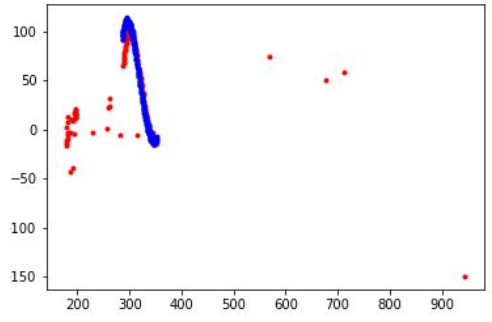


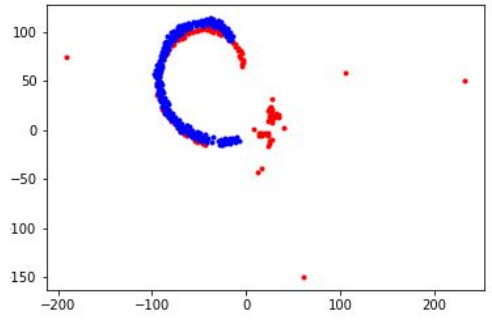




Differential Evolution







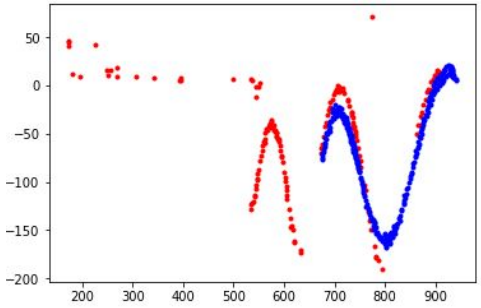
Event 299

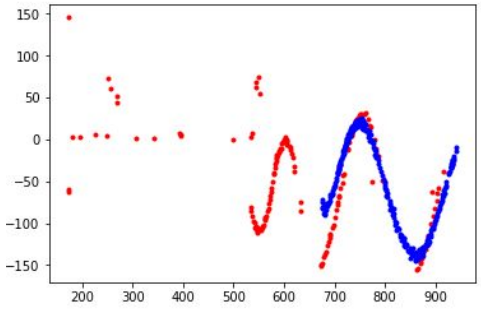
Monte-Carlo: chi = 28.9+5.96+1.68 = 36.54

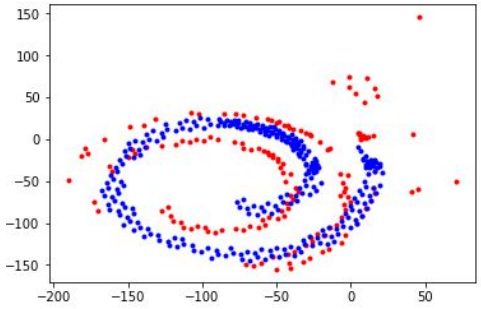
Conjugate Gradient: chi=86.62

Differential evolution: chi=36.427

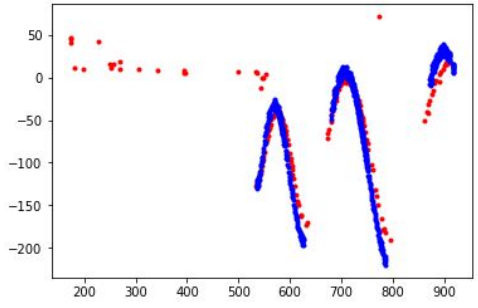
Conjugate Gradient

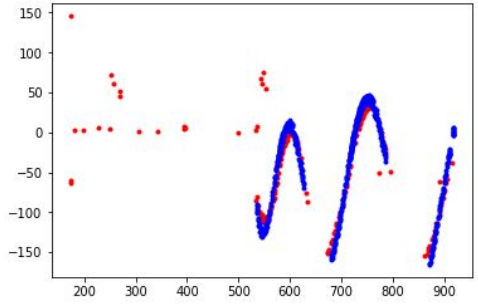


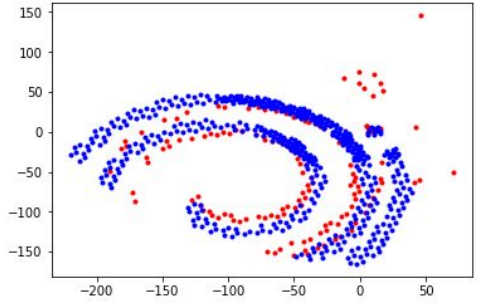




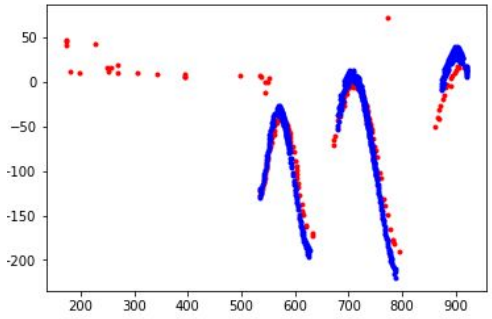
Monte Carlo

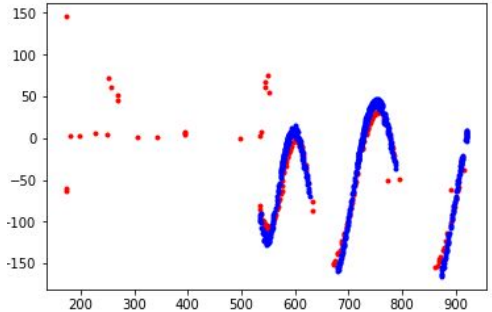


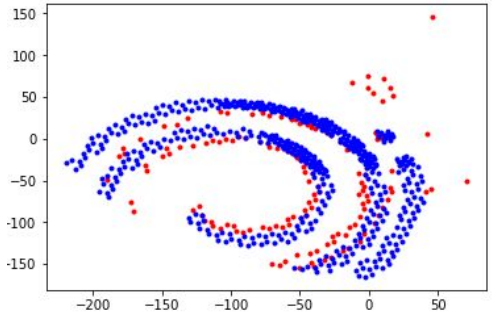




Differential Evolution







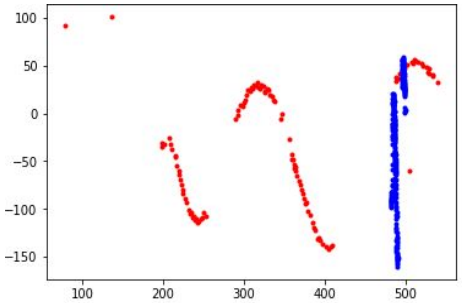
Event 399

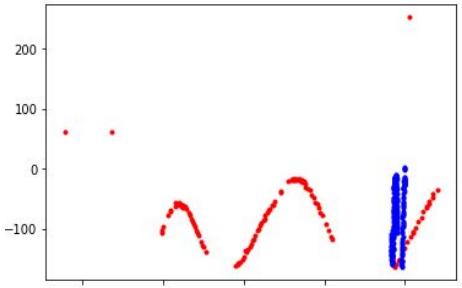
Monte-Carlo: chi = 12.22+7.26+0.09 = 19.57

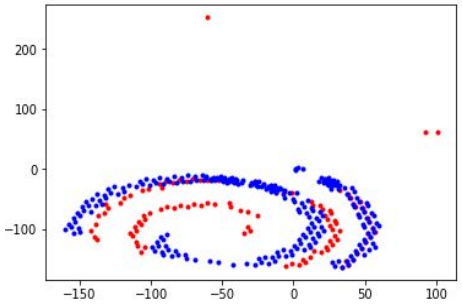
Conjugate Gradient: chi=110.822 (failed)

Differential evolution: chi=18.97

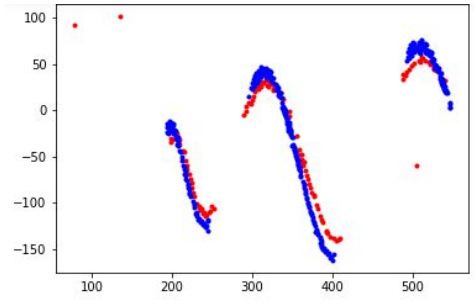
Conjugate Gradient

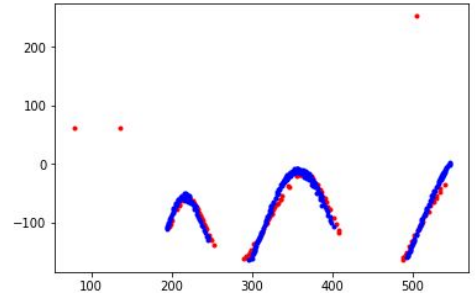


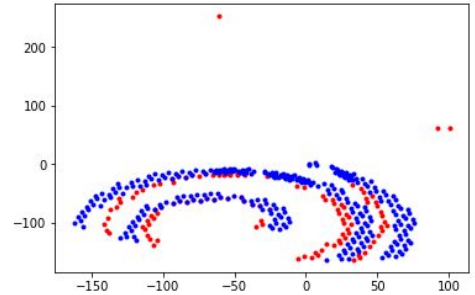




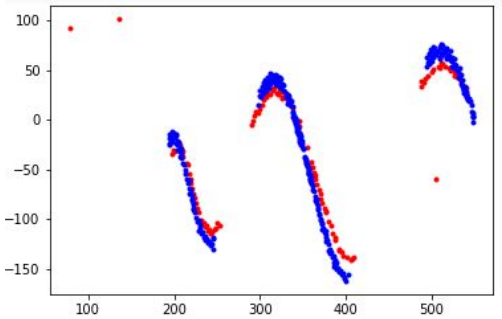
Monte Carlo

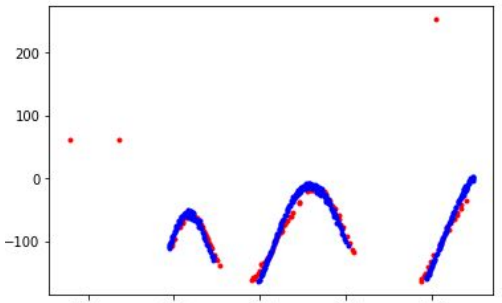


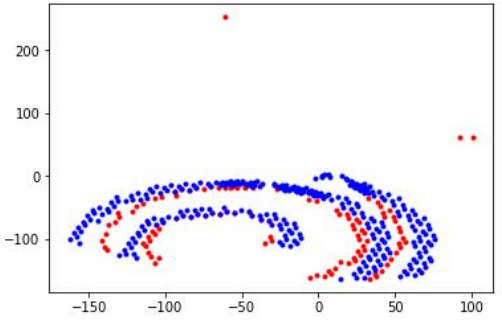




Differential Evolution







Other Methods

The Basin-Hopping Algorithm

Similar to the Simulated Annealing method (but more performant according to Scipy)

1. Chooses a starting point
2. Computes the local minimum (s) using one of the local minimum functions in scipy.optimize.minimize
3. Applies a random perturbation (candidate state) to the local minimum (s')
4. When T ~ 0, if energy at s' is larger than energy at s, the probability of transition from s to s' is approximately zero
5. However when T is large, the system is more likely to update to s' in order to have a broader region of search space
6. T becomes smaller per iteration so that the system converges

However it gave very unstable & inefficient performance on our data - could be that our function is not smooth enough; we are not sure which T or step size to choose; the method usually fails before giving a significantly lower chi-square than our initial guess