Task 4. Algorithms for unconstrained nonlinear optimization. Stochastic and metaheuristic algorithms.

Egor Turukhanov

Group C4134

05.11.19

Data generation

Firstly, we generated noisy data according to following rules:

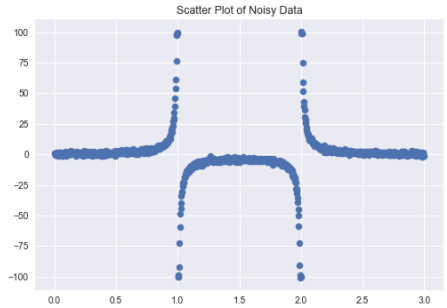
$$y_k = \begin{cases} -100 + \delta_k, & f(x_k) < -100, \\ f(x_k) + \delta_k, & -100 \le f(x_k) \le 100, \\ 100 + \delta_k, & f(x_k) > 100, \end{cases} \qquad x_k = \frac{3k}{1000}, \qquad f(x) = \frac{1}{x^2 - 3x + 2},$$

Code for Data generation:

```
def func(x):
    return 1/(x**2-3*x+2)
random.normalvariate(0,1)
def noisy(elements):
    for i in range(0, elements):
        x=(3*i)/elements
        if func(x) <-100:
            y=-100+random.normalvariate(0,1)
            noisy_dt.append([x,y])
        elif func(x) >=-100 and func(x) <=100:
            y=func(x)+random.normalvariate(0,1)
            noisy_dt.append([x,y])
        elif func(x) >100:
            y=100+random.normalvariate(0,1)
            noisy_dt.append([x,y])


noisy_dt= list()
noisy(1000)
noisy_df = pd.DataFrame(noisy_dt, columns=['x', 'y'])
```

Below you can scatter plot of our noisy data.



Nelder-Mead optimization Algorithm.

For Nelder-Mead Optimization Algorithms we have initial guess = 0.5 ,0.5, 0.5, 0.5. Maximum Iterations=1000 and Precision=0.01.

Below you can see code for functions minimization.

```
def F(x):
    return (x[0]*noisy_df.x.values+x[1])/(noisy_df.x.values**2+x[2]*noisy_df.x.values+x[3])

def lstsqr(x):
    return np.sum((F(x)-noisy_df.y.values)**2)

### Nelder Mead
%time nelder_mead = optimize.minimize(lstsqr, (0.5,0.5,0.5,0.5),
                                      method='Nelder-Mead', options={'maxiter':1000,'xtol':0.01})
a_nm = nelder_mead['x'][0]
b_nm = nelder_mead['x'][1]
c_nm = nelder_mead['x'][2]
d_nm = nelder_mead['x'][3]
it_nm =  nelder_mead['nit']
```

It takes 56.8 ms and 394 iterations to find function minimum. Function minimum is 135231.50049.

```
nelder_mead['fun']
135231.50049236306
```

```
   ...: %time nelder_mead =
optimize.minimize(lstsqr,
(0.5,0.5,0.5,0.5),method='Nelder-Mead'
options={'maxiter':1000,'xtol':0.01})
   ...: a_nm = nelder_mead['x'][0]
   ...: b_nm = nelder_mead['x'][1]
   ...: c_nm = nelder_mead['x'][2]
   ...: d_nm = nelder_mead['x'][3]
   ...: it_nm =  nelder_mead['nit']
Wall time: 56.8 ms
```

```
In [4]: it_nm
Out[4]: 394
```

Levenberg-Marquardt method

Now we applied Levenberg-Marquardt Algorithm to minimize our function.

```
def lstsqr_lma(x):
    return (F(x)-noisy_df.y.values)**2

x0=[2,2,2,2]

%time least_squares = optimize.leastsq(lstsqr_lma, x0, xtol=0.01,full_output=True)

a_lma = least_squares[0][0]
b_lma = least_squares[0][1]
c_lma = least_squares[0][2]
d_lma = least_squares[0][3]
it_lma =  least_squares[2]['nfev']
```

For Levenberg-Marquardt Algorithm we got Iterations = 86 and time = 2.99 ms. Function Minimum is 572801.0961.

```
In [10]: it_lma
Out[10]: 86
```

```
    ...: lstsqr_lma_sum(least_squares[0])
Wall time: 2.99 ms
Out[62]: 572801.0961173662
```

Next, we have applied Particle Swarm Optimization Method for function minimization.

Particle Swarm Optimization Method

Next, we have applied Particle Swarm Optimization Method for function minimization.

```
import psopy
from psopy import _minimize_pso
x0 = np.random.uniform(0, 2, (1000, 4))
def F(x):
    return (x[0]*noisy_df.x.values+x[1])/(noisy_df.x.values**2+x[2]*noisy_df.x.values+x[3])

def lstsqr(x):
    return np.sum((F(x)-noisy_df.y.values)**2)

lstsqr_ =  lambda x: np.apply_along_axis(lstsqr, 1, x)

%time swap_part = _minimize_pso(lstsqr_, x0,max_iter=1000)

a_sp = swap_part['x'][0]
b_sp = swap_part['x'][1]
c_sp = swap_part['x'][2]
d_sp = swap_part['x'][3]
it_sp =  swap_part['nit']
```

For Particle Swarm Algorithm iterations = 282 and time=57.5 sec. Function minimum is 195808.445.

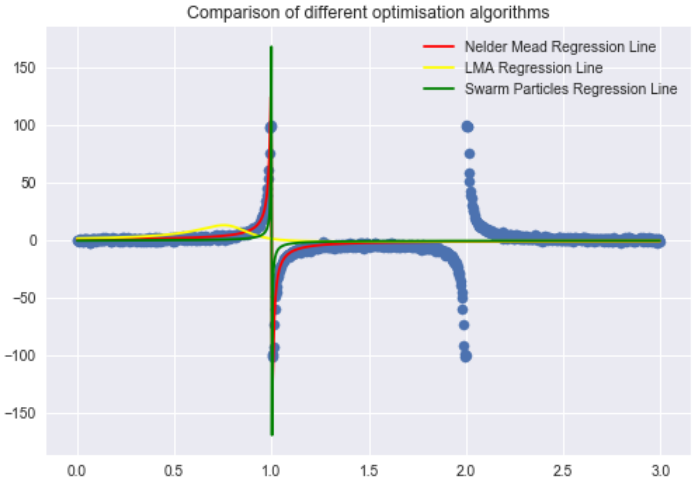
```
    ...: %time swap_part = _minimize_pso(lstsqr_, x0,max_iter=1000, ptol=0.01)
    ...:
    ...: a_sp = swap_part['x'][0]
    ...: b_sp = swap_part['x'][1]
    ...: c_sp = swap_part['x'][2]
    ...: d_sp = swap_part['x'][3]
    ...: it_sp =  swap_part['nit']
    ...: swap_part['fun']
Wall time: 57.5 s
Out[41]: 195808.44495869975
```

```
In [42]: it_sp
Out[42]: 282
```

Comparison of different optimization methods function minimization

As can be seen from graph below all optimization methods regression lines are close to each other.

Method	Time, ms	Number of iterations	Function minimum	Precision
Nelder-Mead	56.8	394	135231.50049	0.01
Levenberg-Marquardt	2.99	86	572801.0961	0.01
Particle Swarm	57500	282	195808.445	0.01



Results and conclusion.

As can be seen from results above Nelder-Mead and Particle Swarm Algorithms showed better result comparing to the Levenberg-Marquardt Method. In matter of time Particle Swarm showed the worst result and Levenberg-Marquardt showed the best one. Moreover Levenberg-Marquardt Algorithms used less iterations than Nelder-Mead and Particle Swarm. In our case we have Least Squares minimization problem. So, method created specifically for that purpose works better in matter of time and iteration. However, on graph below you can see result of minimization without 0.01 precision error. In this situation Levenberg-Marquardt Algorithm show us the best result in matter of time, iteration and function minimum.