Numerical Optimization HW06

December 1, 2021

20211108 Donghyuk Jung

1 Least Square Method

Implement Gauss-Newton's and LM(Levenberg-Marquardt) for the following models and the given observations:

- Model 1 : $\phi_1(a, b, c, d; x, y, z) = ax + by + cz + d$
- Model 2 : $\phi_2(a, b, c, d; x, y, z) = \exp -\frac{(x-a)^2+(y-b)^2+(z-c)^2}{d^2}$

Determine unknown parameters a, b, c and d in the least square sence

2 Common funtion implementation

[13]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.tri as tri
import math as m
import time as t
from tabulate import tabulate

def printVal(param, loss, iter):
    print("loss : %.8f , parameter (a, b, c, d) = (%.4f, %.4f, %.4f, %.4f) ␣
 ↪iteration : %d"% (loss,param[0][0],param[0][1],param[0][2],param[0][3],␣
 ↪iter))
```

3 Implementation

Read observation data from csv file

[4]:
```
obs = np.genfromtxt('NOHW06.csv', delimiter=',', skip_header=1)
N=len(obs)
```

Define residue $r_k = \phi(p; t) - y$ where $p = (a, b, c, d),\ t = (x, y, z)$

[5]:
```
h = 1e-3
eps=1e-15
```

```
def r1(param):
    a, b, c, d = param
    res=np.array([])
    for x1, x2, x3,y in obs:
        res=np.append(res,a*x1+b*x2+c*x3+d-y)
    return res

def r2(param):
    a, b, c, d = param
    res=np.array([])
    for x1, x2, x3,y in obs:
        res=np.append(res,np.exp(-((x1-a)**2+(x2-b)**2+(x3-c)**2)/d**2)-y)
    return res
```

Define Jaccobian

[8]:
```
def grad(param, r,i):
    p1=param.copy()
    p2=param.copy()
    p1[i][0]+=h
    p2[i][0]-=h
    dr=(r(p1)-r(p2))/(2*h)
    # print(dr)
    return np.reshape(dr,(N,1))

def jaccobian(param,r):
    J=grad(param,r,0).T
    J=np.concatenate((J,grad(param,r,1).T))
    J=np.concatenate((J,grad(param,r,2).T))
    J=np.concatenate((J,grad(param,r,3).T))
    return J.T

def ls(pk,r):
    rk=r(pk)
    sum=0
    for i in range(len(rk)):
        sum+=(rk[i]**2)
    return sum/2
```

4 Gauss-Newton Method

search direction p_k^{GN} : $J(x_k)^T J(x_k) p_k^{GN} = -J(x_k)^T r(x_k)$

[17]:
```
def LS_GN(p0, r):
    pk=p0
    iter=0
    loss=0
```

```
    while True:
        loss=ls(pk,r)
        iter += 1
        J=jaccobian(pk,r)
        JTJinv=np.linalg.inv(np.matmul(J.T,J))
        JTJinvJT=np.matmul(JTJinv,J.T)
        pk1=pk-np.matmul(JTJinvJT,r(pk)).reshape(4,1)
        pk=pk1
        if np.abs(loss-ls(pk,r))<eps:
            break

    return pk.T, loss, iter
```

5 Levenberg-Marquardt Method

search direction p_k^{LM} : $(J(x_k)^T J(x_k) + \lambda_k I)p_k^{LM} = -J(x_k)^T r(x_k)$ for some λ_k

when $\lambda_k \to 0$, it may reach $J(x_k)^T J(x_k)p_k = -J(x_k)^T r(x_k)$

- $p_k \to p_k^{GN}$: a Gauss-Newton direction
- It comes to Gauss-Newton method

when $\lambda_k \to \infty$, it may reach p_k : $\lambda_k p_k = -J(x_k)^T r(x_k)$

- $p_k \to -k\nabla f(x_k), k$: const
- It comes to the mothod of steepest descent

[40]:
```
def LS_LM(p0, r):
    pk=p0
    iter=0
    lk=1
    a=2
    loss=0
    while True:
        lk=10
        loss=ls(pk,r)
        iter += 1
        print(iter, loss)
        J=jaccobian(pk,r)
        JTJinv=np.linalg.inv(np.matmul(J.T,J))
        JTJinvJT=np.matmul(JTJinv+np.identity(4)*lk,J.T)
        pk1=pk-np.matmul(JTJinvJT,r(pk)).reshape(4,1)
        if loss-ls(pk1,r)<0:
            temp=pk1
            while True:
                temp=pk1
                JTJinvJT=np.matmul(JTJinv+np.identity(4)*lk,J.T)
                pk1=pk-np.matmul(JTJinvJT,r(pk)).reshape(4,1)
```

```
                lk = lk / a
                print(iter, loss-ls(pk1,r))
                if loss-ls(pk1,r)>0:
                    pk1=temp
                    lk=lk*a
                    break
        else :
            while True:
                JTJinvJT=np.matmul(JTJinv+np.identity(4)*lk,J.T)
                pk1=pk-np.matmul(JTJinvJT,r(pk)).reshape(4,1)
                lk = lk * a
                print(lk)
                if loss-ls(pk1,r)>0:
                    break
        pk=pk1
        print(loss)
        pk1=pk-np.matmul(JTJinvJT,r(pk)).reshape(4,1)
        if np.abs(loss-ls(pk,r))<eps:
            break
    return pk.T, loss, iter
```

6 Result

ϕ_2 is better than ϕ_1. but ϕ_2 include exponential term, ϕ_2 works quite sensitively to the initial value.

Implementation of LM method doesn't not work well

[41]:
```
th1=np.array([[10.],[10.],[10.],[10.]])

print("Gauss-Newton method using model 1 (linear model)")
printVal(*LS_GN(th1,r1))

print("Gauss-Newton method using model 2 (Gaussian model)")
printVal(*LS_GN(th1,r2))



print("Levenberg-Marquardt doesn't not work well")
```

```
Gauss-Newton method using model 1 (linear model)
loss : 1.87512104 , parameter (a, b, c, d) = (0.0017, 0.0007, -0.0035, 0.2535)
iteration : 2
Gauss-Newton method using model 2 (Gaussian model)
loss : 0.06488771 , parameter (a, b, c, d) = (5.3313, 5.6874, 5.6071, 9.9244)
iteration : 9
Levenberg-Marquardt doesn't not work well
```