1. linear regression function by gradient descent
2. method:

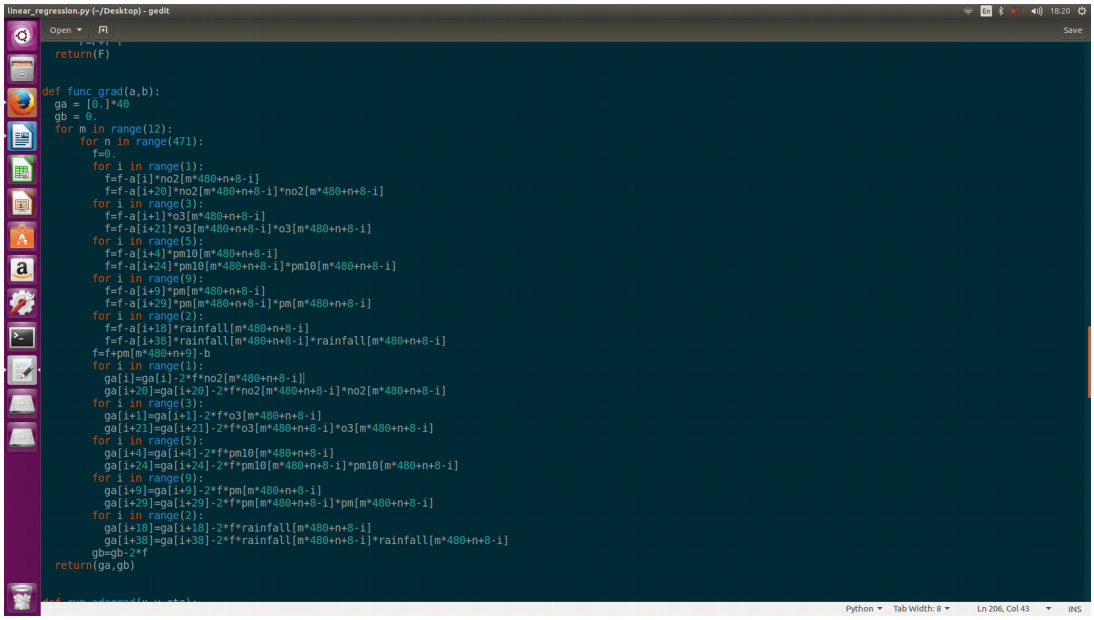
After searching for some aerological knowledge and calculating the correlation coefficient, I chose to use previous 1 hour of NO2 as feature and training data.

previous 3 hours of O3
previous 5 hours of PM10
previous 9 hours of PM2.5
previous 2 hours of RAINFALL

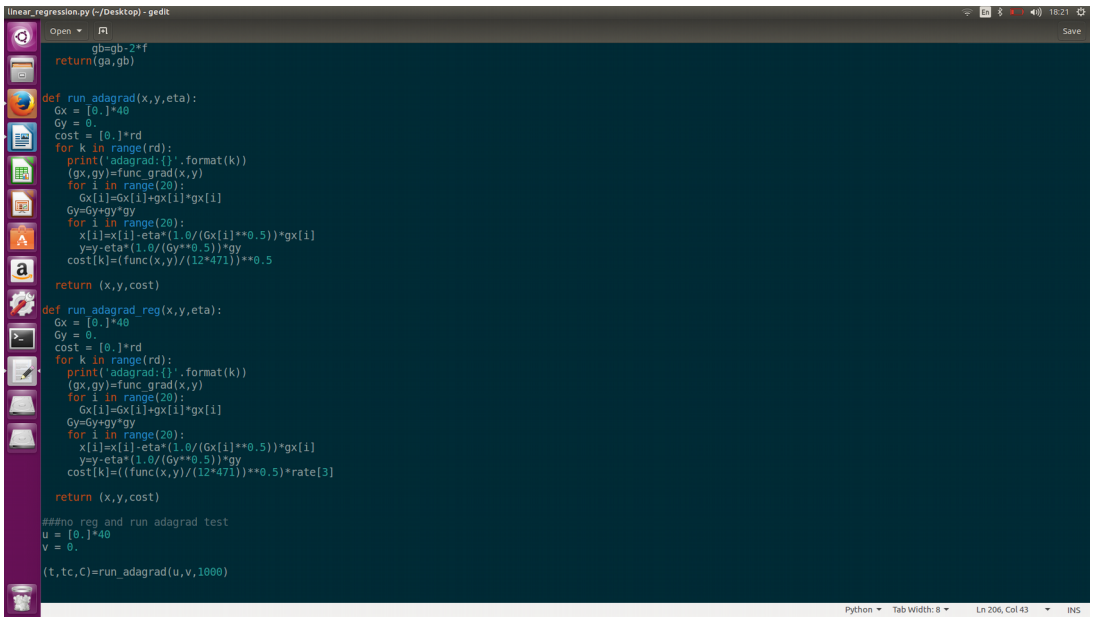
First, I applied regulization on the data.

In order to achieve better precision, I added both linear term and quadratic term to the approximation.

some main funcitons are :



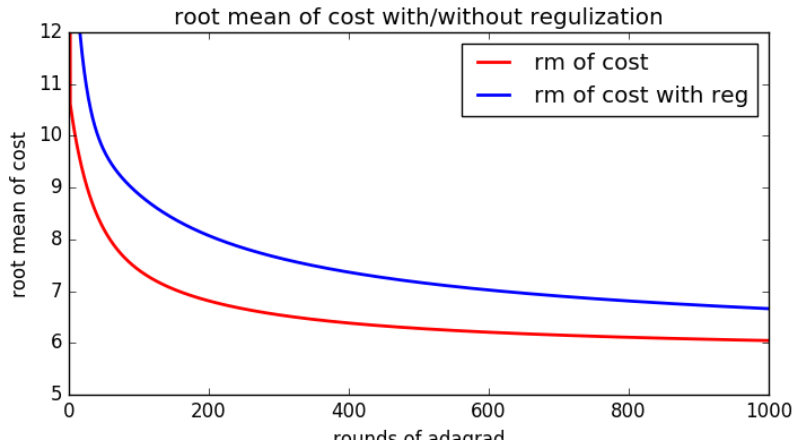
```
  return(F)


def func_grad(a,b):
  ga = [0.]*40
  gb = 0.
  for m in range(12):
      for n in range(471):
        f=0.
        for i in range(1):
          f=f-a[i]*no2[m*480+n+8-i]
          f=f-a[i+20]*no2[m*480+n+8-i]*no2[m*480+n+8-i]
        for i in range(3):
          f=f-a[i+1]*o3[m*480+n+8-i]
          f=f-a[i+21]*o3[m*480+n+8-i]*o3[m*480+n+8-i]
        for i in range(5):
          f=f-a[i+4]*pm10[m*480+n+8-i]
          f=f-a[i+24]*pm10[m*480+n+8-i]*pm10[m*480+n+8-i]
        for i in range(9):
          f=f-a[i+9]*pm[m*480+n+8-i]
          f=f-a[i+29]*pm[m*480+n+8-i]*pm[m*480+n+8-i]
        for i in range(2):
          f=f-a[i+18]*rainfall[m*480+n+8-i]
          f=f-a[i+38]*rainfall[m*480+n+8-i]*rainfall[m*480+n+8-i]
        f=f+pm[m*480+n+9]-b
        for i in range(1):
          ga[i]=ga[i]-2*f*no2[m*480+n+8-i]
          ga[i+20]=ga[i+20]-2*f*no2[m*480+n+8-i]*no2[m*480+n+8-i]
        for i in range(3):
          ga[i+1]=ga[i+1]-2*f*o3[m*480+n+8-i]
          ga[i+21]=ga[i+21]-2*f*o3[m*480+n+8-i]*o3[m*480+n+8-i]
        for i in range(5):
          ga[i+4]=ga[i+4]-2*f*pm10[m*480+n+8-i]
          ga[i+24]=ga[i+24]-2*f*pm10[m*480+n+8-i]*pm10[m*480+n+8-i]
        for i in range(9):
          ga[i+9]=ga[i+9]-2*f*pm[m*480+n+8-i]
          ga[i+29]=ga[i+29]-2*f*pm[m*480+n+8-i]*pm[m*480+n+8-i]
        for i in range(2):
          ga[i+18]=ga[i+18]-2*f*rainfall[m*480+n+8-i]
          ga[i+38]=ga[i+38]-2*f*rainfall[m*480+n+8-i]*rainfall[m*480+n+8-i]
        gb=gb-2*f
  return(ga,gb)
```



```
        gb=gb-2*f
  return(ga,gb)


def run_adagrad(x,y,eta):
  Gx = [0.]*40
  Gy = 0.
  cost = [0.]*rd
  for k in range(rd):
    print('adagrad:{}'.format(k))
    (gx,gy)=func_grad(x,y)
    for i in range(20):
      Gx[i]=Gx[i]+gx[i]*gx[i]
    Gy=Gy+gy*gy
    for i in range(20):
      x[i]=x[i]-eta*(1.0/(Gx[i]**0.5))*gx[i]
      y=y-eta*(1.0/(Gy**0.5))*gy
    cost[k]=(func(x,y)/(12*471))**0.5

  return (x,y,cost)

def run_adagrad_reg(x,y,eta):
  Gx = [0.]*40
  Gy = 0.
  cost = [0.]*rd
  for k in range(rd):
    print('adagrad:{}'.format(k))
    (gx,gy)=func_grad(x,y)
    for i in range(20):
      Gx[i]=Gx[i]+gx[i]*gx[i]
    Gy=Gy+gy*gy
    for i in range(20):
      x[i]=x[i]-eta*(1.0/(Gx[i]**0.5))*gx[i]
      y=y-eta*(1.0/(Gy**0.5))*gy
    cost[k]=((func(x,y)/(12*471))**0.5)*rate[3]

  return (x,y,cost)

###no reg and run adagrad test
u = [0.]*40
v = 0.

(t,tc,C)=run_adagrad(u,v,1000)
```

And then apply adagrad to the cost function to seek the ideal value of coefficients x[0] to x[39].

3. applying regulization doesn't seem to cause significant improvement on the speed to stability.



4. changing eta: since Gx and Gy become very large after several rounds, small eta may be more accurate in the beginning but is not efficient enough.
(the red curve is overlapped by the blue curve)