学号：22111076　姓名：卢常建

```python
if __name__ == '__main__':
    # 定义初始点和精度误差
    value = np.array([[3, -1, 0, 1]])
    err = 0.0001
    # 定义搜索函数，这里直接定义所用变量和题目的函数
    x_1 = sympy.symbols('x_1')
    x_2 = sympy.symbols('x_2')
    x_3 = sympy.symbols('x_3')
    x_4 = sympy.symbols('x_4')
    optiFun = (x_1 + 10 * x_2) ** 2 + 5 * (x_3 - x_4) ** 2 + (x_2 - 2 * x_3) ** 4 + 10 * (x_1 - x_4) ** 4
    #Steepestdes(optiFun, err, value)
    #Zunewton(optiFun, err, value)
    Quasinewton(optiFun, err, value, 0)
    #Conjugategra(optiFun, err, value)
```

```python
# 非精确搜索，使用Wolfe准则
def Inexactsearch(optiFun,d,value):
    c_1 = random.uniform(0, 1)
    c_2 = random.uniform(c_1, 1)
    alpha = 1
    n = -9999999999999
    m = 0
    x1 = value
    x_symbol = sympy.symbols('x_1:' + str(value.shape[1] + 1))
    x1_value = dict(zip(x_symbol, x1[0]))
    x2 = x1 + alpha * d
    x2_value = dict(zip(x_symbol, x2[0]))
    # 非精确搜索的两个条件
    conditon_1 = (optiFun.subs(x2_value) - optiFun.subs(x1_value)) <= c_1 * alpha * np.dot(Gradient(optiFun, x1), d.T)
    conditon_2 = np.dot(Gradient(optiFun, x2), d.T) >= c_2 * np.dot(Gradient(optiFun, x1), d.T)
    # 在不满足条件的情况下继续搜索步长
    while not (conditon_1 and conditon_2):
        if conditon_1:
            m = alpha
            alpha = min(2 * alpha, (alpha + n) / 2)
        else:
            n = alpha
            alpha = (alpha + m) / 2
        x2 = x1 + alpha * d
        x2_value = dict(zip(x_symbol, x2[0]))
        conditon_1 = (optiFun.subs(x2_value) - optiFun.subs(x1_value)) <= c_1 * alpha * np.dot(Gradient(optiFun, x1),
                                                                                                 d.T)
        conditon_2 = np.dot(Gradient(optiFun, x2), d.T) >= c_2 * np.dot(Gradient(optiFun, x1), d.T)
    return alpha
```

1、 非精确搜索的最速下降法

```python
# 使用非精确搜索的最速下降法，传入参数为求最优值的函数，精度误差，初始点
def Steepestdes(optiFun,err,value):
    x1 = value
    step_gra = Gradient(optiFun,x1)
    err_k = norm(step_gra)
    iter_num = 1
    while err_k > err:
        print('第'+str(iter_num)+'次迭代的误差为:',err_k)
        # 确定下降方向，进行非精确搜索，确定步长alpha的值
        d_k = -1 * step_gra
        alpha_k = Inexactsearch(optiFun,d_k,x1)
        # 更新新的x值
        x1 = x1 - alpha_k * step_gra
        step_gra = Gradient(optiFun,x1)
        err_k = norm(step_gra)
        iter_num = iter_num + 1
    print('第'+str(iter_num)+'次迭代的误差为:',err_k)
    return x1
```

```
In [7]: Steepestdes(optiFun,0.0001,value)
```

```
第6722次迭代的误差为: 0.00012899830
第6723次迭代的误差为: 0.00020421998
第6724次迭代的误差为: 0.00010577965
第6725次迭代的误差为: 0.00012795023
第6726次迭代的误差为: 0.00020119207
第6727次迭代的误差为: 0.00010549054
第6728次迭代的误差为: 0.00044090912
第6729次迭代的误差为: 0.0002782758
第6730次迭代的误差为: 0.00023100937
第6731次迭代的误差为: 0.00010811435
第6732次迭代的误差为: 0.0002539716
第6733次迭代的误差为: 0.00021219664
第6734次迭代的误差为: 0.0001467548
第6735次迭代的误差为: 0.00010131586
第6736次迭代的误差为: 0.00028397492
第6737次迭代的误差为: 0.00025950378
第6738次迭代的误差为: 0.000110860536
第6739次迭代的误差为: 9.919983e-05

Out[7]: array([[ 0.0244244 , -0.00244204,  0.01216624,  0.01217759]])
```

2、一维精确搜索的阻尼牛顿法

```python
def Zunewton(optiFun,err,value):
    x1 = value
    err_k = norm(Gradient(optiFun,x1))
    iter_num = 1
    while err_k > err:
        # 确定下降方向，进行精确搜索，确定步长alpha的值
        print('第' + str(iter_num) + '次迭代的误差为:', err_k)
        gra_k = Gradient(optiFun,x1)
        hess_k = Hessian(optiFun,x1)
        d_k = -1*gra_k.dot(inv(hess_k))
        alpha_k = Inexactsearch(optiFun,d_k,x1)
        # 更新新的x值和误差
        x1 = x1 + alpha_k * d_k
        err_k = norm(Gradient(optiFun,x1))
        iter_num = iter_num + 1
    print('第' + str(iter_num) + '次迭代的误差为:', err_k)
    return x1
```

```
In [20]: Zunewton(optiFun,0.0001,value)
```

```
第1次迭代的误差为: 458.77664
第2次迭代的误差为: 134.11421
第3次迭代的误差为: 39.73758
第4次迭代的误差为: 11.77404
第5次迭代的误差为: 3.488602
第6次迭代的误差为: 1.0336624
第7次迭代的误差为: 1.0016959
第8次迭代的误差为: 0.29679793
第9次迭代的误差为: 0.28761926
第10次迭代的误差为: 0.08522057
第11次迭代的误差为: 0.025250599
第12次迭代的误差为: 0.007481636
第13次迭代的误差为: 0.007452449
第14次迭代的误差为: 0.005740314
第15次迭代的误差为: 0.0017006486
第16次迭代的误差为: 0.0009841806
第17次迭代的误差为: 0.00029157358
第18次迭代的误差为: 0.00022458886
第19次迭代的误差为: 6.6570705e-05

Out[20]: array([[ 0.01256714, -0.00125671,  0.00201079,  0.00201078]])
```

3、 一维精确搜索的拟牛顿法

```python
# 传入参数为求最优值的函数, 精度误差, 初始点及决定是DFP还是BFGS的rho,其意义见书上BFGS部分的公式
def Quasinewton(optiFun,err,value,rho=0):
    # 先以单位矩阵作为第一次迭代的H进行计算
    x1 = value
    gra_k = Gradient(optiFun,x1)
    H = np.eye(value.shape[1])
    d_k = -1*gra_k.dot(H)
    # 根据下降方向确定步长, 并得到更新后的x值及误差
    alpha_k = Inexactsearch(optiFun,d_k,x1)
    x2 = x1 + alpha_k*d_k
    err_k = norm(Gradient(optiFun,x2))
    iter_num = 1
    # 开始进行迭代, 从而更新对称正定矩阵H和x值
    while err_k > err:
        print('第' + str(iter_num) + '次迭代的误差为:', err_k)
        del_x = x2 - x1
        del_y = Gradient(optiFun,x2) - Gradient(optiFun,x1)
        v = del_x.T/(del_x.dot(del_y.T)) - H.dot(del_y.T)/(del_y.dot(H)).dot(del_y.T)
        part_1 = del_x.T.dot(del_x)/del_x.dot(del_y.T)
        part_2 = (H.dot(del_y.T)).dot(del_y.dot(H.T))/(del_y.dot(H)).dot(del_y.T)
        H = H +  part_1 - part_2 + rho*((del_y.dot(H)).dot(del_y.T))*v.dot(v.T)
        x1 = x2
        gra_k = Gradient(optiFun,x1)
        d_k = -1 * gra_k.dot(H)
        alpha_k = Inexactsearch(optiFun,d_k,x1)
        x2 = x1 + alpha_k*d_k
        err_k = norm(Gradient(optiFun,x2))
        iter_num = iter_num + 1
    print('第' + str(iter_num) + '次迭代的误差为:', err_k)
    return x2
```

```
In  [8]: Quasinewton(optiFun,0.0001,value,0)

第1次迭代的误差为: 291.9626
第2次迭代的误差为: 290.84967
第3次迭代的误差为: 297.6821
第4次迭代的误差为: 233.22322
第5次迭代的误差为: 182.06262
第6次迭代的误差为: 59.76996
第7次迭代的误差为: 44.05109
第8次迭代的误差为: 15.223175
第9次迭代的误差为: 7.034249
第10次迭代的误差为: 2.996958
第11次迭代的误差为: 1.3103073
第12次迭代的误差为: 1.0804081
第13次迭代的误差为: 0.6452001
第14次迭代的误差为: 0.24059835
第15次迭代的误差为: 0.110256836
第16次迭代的误差为: 0.046240594
第17次迭代的误差为: 0.03127424
第18次迭代的误差为: 0.030428844
第19次迭代的误差为: 0.009186413
第20次迭代的误差为: 0.0045240545
第21次迭代的误差为: 0.0029943308
第22次迭代的误差为: 0.0010824675
第23次迭代的误差为: 0.0005023006
第24次迭代的误差为: 0.00021225355
第25次迭代的误差为: 9.2675225e-05

Out[8]: array([[ 0.01384411, -0.00138441,  0.00206054,  0.00206053]])
```

4、 一维精确搜索的共轭梯度法

```python
def Conjugategra(optiFun,err,value):
    # 先以初始点的梯度反方向为下降方向进行计算
    x1 = value
    d_k = -1*Gradient(optiFun,x1)
    # 根据下降方向确定步长，并得到更新后的x值及误差
    alpha_k = Inexactsearch(optiFun,d_k,x1)
    x2 = x1 + alpha_k*d_k
    err_k = norm(Gradient(optiFun,x2))
    # 开始进行迭代，从而更新下降方向和x值
    while err_k > err:
        gra_1 = Gradient(optiFun,x1)
        gra_2 = Gradient(optiFun,x2)
        beta = gra_2.dot(gra_2.T)/gra_1.dot(gra_1.T)
        d_k = -1*gra_2+beta*d_k
        x1 = x2
        alpha_k = Inexactsearch(optiFun,d_k,x1)
        x2 = x1 + alpha_k*d_k
        err_k = norm(Gradient(optiFun,x2))
    return x2
```

```
In  [14]: Conjugategra(optiFun,0.0001,value)
第281次迭代的误差为：0.00016092852
第282次迭代的误差为：0.00018898165
第283次迭代的误差为：0.00021105367
第284次迭代的误差为：0.0006551608
第285次迭代的误差为：0.0006585924
第286次迭代的误差为：0.0005627291
第287次迭代的误差为：0.00047845507
第288次迭代的误差为：0.0001992084
第289次迭代的误差为：0.00021346605
第290次迭代的误差为：0.00023782701
第291次迭代的误差为：0.00053317286
第292次迭代的误差为：0.00040954608
第293次迭代的误差为：0.00028671682
第294次迭代的误差为：0.00024187179
第295次迭代的误差为：0.00032580495
第296次迭代的误差为：0.00033522662
第297次迭代的误差为：0.00035978295
第298次迭代的误差为：6.713368e-05

Out[14]: array([[-0.0187086 ,  0.00187077, -0.0101166 , -0.01011966]])
```