学号：22111076　姓名：卢常建

```python
if __name__ == '__main__':
    n = 276
    a = np.random.randint(1,10,[n,1])
    G = np.dot(a,a.T) + random.randint(1,2)*np.eye(n)
    b = 0.5*np.dot(G,np.ones([n,1]))
    x_array = np.array(sympy.symbols('x_1:'+str(n+1)))
    fun_array = 0.5*(x_array.dot(G)).dot(x_array.T)+np.dot(b.T,x_array.T)
    fun=fun_array[0]
    Steepestdes(fun,0.01,np.zeros([1,n]),G,b)
    Zunewton(0.01, value, G, b)
    Quasinewton(0.01, value, G, b, 1)
    Conjugategra(0.01, value, G, b)
```

```python
def Accuracysearch(optiFun, searchErr):
    # 精确搜索——进退算法确定区间加0.618确定搜索值,传入的函数以alpha为变量
    # 虽然进退算法并不能一定找到单峰区间, 但对于全是凸函数的作业题足够了
    alpha_zero = 0
    alpha = sympy.symbols('alpha')
    alpha1 = alpha_zero
    h = 10
    alpha2 = alpha_zero + h
    if optiFun.subs(alpha, alpha2) > optiFun.subs(alpha, alpha_zero):
        h = -1 * h
    alpha1 = alpha_zero + h
    while optiFun.subs(alpha, alpha1) <= optiFun.subs(alpha, alpha_zero):
        h = 2 * h
        alpha2 = alpha_zero
        alpha_zero = alpha1
        alpha1 = alpha_zero + h
    left = min(alpha1, alpha2)
    right = max(alpha1, alpha2)
    # 0.618算法（可以替换成其他的, 不推荐抛物线法, 此法需要找到合适的三个点）
    lam = left + 0.382 * (right - left)
    mu = left + 0.618 * (right - left)
    while (right - left) > searchErr:
        if optiFun.subs(alpha, left) > optiFun.subs(alpha, right):
            left = lam
            lam = mu
            mu = left + 0.618 * (right - left)
        else:
            right = mu
            mu = lam
            lam = left + 0.382 * (right - left)
    return (left + right) / 2
```

```python
import sympy
from sympy import diff
import numpy as np

# 求任意函数某点的梯度, 默认传入的函数是x_1,x_2这种格式变量构成的函数
# value代表传入的该点处的x值
def Gradient(fun,value):
    x_symbol = sympy.symbols('x_1:'+str(value.shape[1]+1))
    x_value = dict(zip(x_symbol,value[0]))
    gra = np.array([[diff(fun,i).subs(x_value) for i in x_symbol]]).astype(np.float16)
    return gra

# 求任意函数某点的HESSIAN矩阵, 传入参数特点同上述求梯度的函数
def Hessian(fun,value):
    x_symbol = sympy.symbols('x_1:'+str(value.shape[1]+1))
    x_value = dict(zip(x_symbol,value[0]))
    return np.array(sympy.hessian(fun,x_symbol).subs(x_value)).astype(np.float16)
```

1、 一维精确搜索的最速下降法

```python
# 此为一维精确搜索的最速下降法，传入的参数为求最优值的函数，精度误差，初始点
def Steepestdes(err,value,G,b):
    x1 = value
    x_symbol = sympy.symbols('x_1:'+str(value.shape[1]+1))
    step_gra = G.dot(x1.T) + b
    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，进行一维精确搜索，确定alpha的值
        d = -1*step_gra
        alpha_k = -1*step_gra.T.dot(d)/(d.T.dot(G.dot(d)))
        alpha_k = alpha_k[0][0]
        # 更新新的x值
        x1 = x1 + alpha_k * d.T
        step_gra = G.dot(x1.T) + b
        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
```

```
第1次迭代的误差为: 71504.0759957081
第2次迭代的误差为: 7.160114372907483
第3次迭代的误差为: 3.419847275672264

Out[70]: array([[-0.4999871 , -0.4999871 , -0.49999811, -0.50000545, -0.4999871 ,
        -0.50000912, -0.50000178, -0.49997976, -0.49999444, -0.49999444,
        -0.49999077, -0.50000178, -0.49999444, -0.49999077, -0.50000545,
        -0.49999444, -0.49999444, -0.50000912, -0.49999077, -0.50000545,
        -0.49999077, -0.50000545, -0.50000545, -0.49999077, -0.50000912,
        -0.50000545, -0.49997976, -0.49999811, -0.50000178, -0.50000178,
        -0.49998343, -0.50000178, -0.49997976, -0.49999811, -0.50000545,
        -0.50000912, -0.50000912, -0.49998343, -0.50000912, -0.50000545,
        -0.49999444, -0.4999871 , -0.49997976, -0.50000912, -0.50000545,
        -0.50000178, -0.49999811, -0.50000912, -0.49997976, -0.49999077,
        -0.49999811, -0.49998343, -0.50000178, -0.4999871 , -0.50000178,
        -0.49999444, -0.50000912, -0.50000545, -0.49999444, -0.49999811,
        -0.50000178, -0.49999811, -0.49999811, -0.4999871 , -0.50000545,
        -0.49999077, -0.50000178, -0.49999811, -0.50000545, -0.49999077,
        -0.49999077, -0.49997976, -0.50000545, -0.49997976, -0.4999871 ,
        -0.50000912, -0.49997976, -0.50000178, -0.49997976, -0.50000545,
        -0.4999871 , -0.49997976, -0.50000545, -0.50000178, -0.49997976,
        -0.49997976, -0.50000545, -0.49997976, -0.49999811, -0.49998343,
        -0.50000178, -0.49999444, -0.50000178, -0.49999444, -0.50000545,
        -0.49999077, -0.49997976, -0.49997976, -0.50000545, -0.49999444,
        -0.4999871 , -0.50000912, -0.50000912, -0.50000912, -0.50000178,
        -0.49998343, -0.49999444, -0.49999811, -0.50000545, -0.50000545,
        -0.49997976, -0.4999871 , -0.4999871 , -0.50000545, -0.50000545,
        -0.50000912, -0.49999811, -0.49997976, -0.50000178, -0.49999444,
        -0.50000912, -0.49999444, -0.4999871 , -0.50000178, -0.49997976,
        -0.49999077, -0.49999811, -0.49999077, -0.49997976, -0.49998343,
        -0.50000912, -0.50000178, -0.49999811, -0.49998343, -0.49999077,
        -0.49999811, -0.50000912, -0.50000545, -0.49999811, -0.49999811,
        -0.49999811, -0.49999444, -0.49999811, -0.49998343, -0.49999444,
        -0.50000178, -0.50000912, -0.50000545, -0.49999077, -0.49999077,
        -0.49999811, -0.50000545, -0.49999811, -0.49999077, -0.49999444,
        -0.4999871 , -0.50000178, -0.49999077, -0.50000178, -0.49999077,
        -0.50000912, -0.49999444, -0.49999811, -0.49999444, -0.50000178,
        -0.49997976, -0.49999811, -0.49998343, -0.50000912, -0.50000912,
        -0.50000545, -0.49998343, -0.50000178, -0.49999444, -0.49999811,
        -0.49999811, -0.4999871 , -0.49999444, -0.49999811, -0.49999811,
        -0.4999871 , -0.49998343, -0.49999444, -0.49999811, -0.49999444,
        -0.50000545, -0.49999444, -0.49999077, -0.49998343, -0.49997976,
        -0.50000912, -0.50000912, -0.50000912, -0.4999871 , -0.50000912,
        -0.50000178, -0.50000178, -0.50000912, -0.50000545, -0.4999871 ,
        -0.49998343, -0.49999444, -0.49997976, -0.4999871 , -0.50000912,
        -0.49999811, -0.50000912, -0.49999444, -0.49999444, -0.49997976,
        -0.50000545, -0.49998343, -0.49999444, -0.50000545, -0.49999077,
        -0.49999077, -0.4999871 , -0.49999811, -0.50000178, -0.49999077,
        -0.49999811, -0.49999077, -0.49999077, -0.50000178, -0.49998343,
        -0.50000912, -0.49999811, -0.4999871 , -0.49999077, -0.50000912,
        -0.50000912, -0.50000178, -0.50000912, -0.4999871 , -0.50000545,
        -0.49998343, -0.49997976, -0.49999444, -0.50000912, -0.50000545,
        -0.49999444, -0.49997976, -0.49997976, -0.49997976, -0.49999077,
        -0.49999077, -0.50000178, -0.49997976, -0.4999871 , -0.50000178,
        -0.4999871 , -0.49999811, -0.49999811, -0.49999811, -0.4999871 ,
        -0.49997976, -0.50000178, -0.49999811, -0.50000912, -0.49998343,
        -0.50000178, -0.50000178, -0.50000545, -0.49998343, -0.49998343,
        -0.49999444, -0.4999871 , -0.49999444, -0.4999871 , -0.49999077,
        -0.50000178, -0.4999871 , -0.50000178, -0.49998343, -0.49999444,
        -0.49999811]])
```

2、一维精确搜索的阻尼牛顿法

```python
# 此为一维精确搜索的阻尼牛顿法, 传入参数为求最优值的函数, 精度误差, 初始点
def Zunewton(err,value,G,b):
    x1 = value
    step_gra = G.dot(x1.T) + b
    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)
        # 进行一维精确搜索, 确定alpha的值
        hess_k = G
        d = -1*inv(hess_k).dot(step_gra)
        alpha_k = -1*step_gra.T.dot(d)/(d.T.dot(G.dot(d)))
        alpha_k = alpha_k[0][0]
        # 更新新的x值
        x1 = x1 + alpha_k * d.T
        step_gra = G.dot(x1.T) + b
        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
```

```
第1次迭代的误差为: 61885.08597190441
第2次迭代的误差为: 9.195712210462411e-11
```

```
Out[101]: array([[-0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5]])
```

3、 一维精确搜索的拟牛顿法

```python
# 一维精确搜索的拟牛顿法
# 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(err,value,G,b,rho):
    # rho决定是DFP还是BFGS
    x1 = value
    gra_k = G.dot(x1.T) + b
    H = np.eye(value.shape[1])
    d= -1*H.dot(gra_k)
    alpha_k = -1*gra_k.T.dot(d)/(d.T.dot(G.dot(d)))
    alpha_k = alpha_k[0][0]
    x2 = x1 + alpha_k*d.T
    err_k = norm(G.dot(x2.T) + b)
    iter_num = 1
    while err_k > err:
        print('第'+str(iter_num)+'次迭代的误差为:',err_k)
        del_x = x2 - x1
        del_y = (G.dot(x2.T) - G.dot(x1.T)).T
        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 = G.dot(x1.T) + b
        d= -1*H.dot(gra_k)
        alpha_k = -1*gra_k.T.dot(d)/(d.T.dot(G.dot(d)))
        x2 = x1 + alpha_k*d.T
        err_k = norm(G.dot(x2.T) + b)
        iter_num = iter_num + 1
    print('第'+str(iter_num)+'次迭代的误差为:',err_k)
    return x2
```

```
In [145]: Quasinewton(0.01,value,G,b,1)

第1次迭代的误差为: 3.760949669009528
第2次迭代的误差为: 3.1805302477212774e-11

Out[145]: array([[-0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5]])
```

4、一维精确搜索的共轭梯度法

```python
# 此为一维精确搜索的共轭梯度法，传入的参数为最优值的函数，精度误差，初始点
def Conjugategra(err,value,G,b):
    x1 = value
    gra_k = G.dot(x1.T) + b
    d = -1*gra_k
    alpha_k = -1*gra_k.T.dot(d)/(d.T.dot(G.dot(d)))
    alpha_k = alpha_k[0][0]
    x2 = x1 + alpha_k*d.T
    err_k = norm(G.dot(x2.T) + b)
    iter_num = 1
    while err_k > err:
        print('第'+str(iter_num)+'次迭代的误差为:',err_k)
        gra_1 = G.dot(x1.T) + b
        gra_2 = G.dot(x2.T) + b
        beta = gra_2.T.dot(gra_2)/gra_1.T.dot(gra_1)
        d = -1*gra_2+beta*d
        x1 = x2
        alpha_k = -1*gra_k.T.dot(d)/(d.T.dot(G.dot(d)))
        alpha_k = alpha_k[0][0]
        x2 = x1 + alpha_k*d.T
        err_k = norm(G.dot(x2.T) + b)
    print('第'+str(iter_num)+'次迭代的误差为:',err_k)
    return x2
```

```
In  [147]: Conjugategra(0.01,value,G,b)

第1次迭代的误差为: 3.760949669009528
第1次迭代的误差为: 4.074381133185618e-07

Out[147]: array([[-0.50000001, -0.49999998, -0.50000003, -0.50000002, -0.50000001,
        -0.49999999, -0.5       , -0.49999998, -0.50000004, -0.50000004,
        -0.50000004, -0.50000002, -0.50000003, -0.49999999, -0.49999998,
        -0.5       , -0.5       , -0.50000003, -0.50000002, -0.5       ,
        -0.50000003, -0.5       , -0.50000003, -0.50000004, -0.49999999,
        -0.50000002, -0.50000003, -0.50000003, -0.50000001, -0.49999998,
        -0.50000001, -0.49999998, -0.49999998, -0.5       , -0.50000003,
        -0.50000003, -0.5       , -0.5       , -0.50000001, -0.50000003,
        -0.50000003, -0.5       , -0.50000002, -0.50000002, -0.50000004,
        -0.50000001, -0.50000001, -0.50000003, -0.50000003, -0.50000003,
        -0.5       , -0.50000002, -0.49999998, -0.50000003, -0.50000001,
        -0.49999999, -0.5       , -0.5       , -0.50000004, -0.5       ,
        -0.50000001, -0.49999999, -0.5       , -0.50000003, -0.49999998,
        -0.49999998, -0.50000004, -0.5       , -0.5       , -0.50000003,
        -0.5       , -0.49999998, -0.50000001, -0.5       , -0.5       ,
        -0.5       , -0.50000001, -0.50000004, -0.50000004, -0.5       ,
        -0.49999999, -0.50000003, -0.50000003, -0.5       , -0.50000003,
        -0.50000003, -0.49999998, -0.50000001, -0.5       , -0.50000001,
        -0.50000001, -0.49999998, -0.50000001, -0.50000003, -0.5       ,
        -0.50000004, -0.49999998, -0.50000001, -0.5       , -0.50000001,
        -0.50000002, -0.50000002, -0.50000002, -0.50000004, -0.50000002,
        -0.49999999, -0.49999999, -0.49999998, -0.50000002, -0.49999999,
        -0.49999999, -0.50000004, -0.50000003, -0.49999998, -0.50000003,
        -0.49999998, -0.50000001, -0.5       , -0.5       , -0.49999998,
        -0.49999998, -0.49999999, -0.5       , -0.5       , -0.5       ,
        -0.49999998, -0.50000003, -0.50000002, -0.5       , -0.5       ,
        -0.5       , -0.50000003, -0.50000003, -0.50000001, -0.50000004,
        -0.50000003, -0.50000003, -0.5       , -0.50000002, -0.50000003,
        -0.49999998, -0.50000003, -0.50000001, -0.5       , -0.50000003,
        -0.50000001, -0.50000003, -0.49999999, -0.49999999, -0.50000002,
        -0.50000003, -0.5       , -0.49999998, -0.50000003, -0.50000004,
        -0.50000003, -0.50000003, -0.49999998, -0.5       , -0.50000003,
        -0.50000003, -0.50000003, -0.50000003, -0.49999999, -0.5       ,
        -0.49999999, -0.50000002, -0.5       , -0.50000004, -0.50000002,
        -0.49999999, -0.5       , -0.49999999, -0.49999999, -0.50000003,
        -0.50000001, -0.50000003, -0.5       , -0.49999999, -0.50000002,
        -0.50000003, -0.5       , -0.49999999, -0.49999999, -0.50000001,
        -0.5       , -0.49999998, -0.49999998, -0.50000003, -0.50000002,
        -0.50000003, -0.5       , -0.5       , -0.49999998, -0.49999999,
        -0.49999999, -0.5       , -0.50000002, -0.50000004, -0.5       ,
        -0.50000002, -0.50000001, -0.49999999, -0.5       , -0.5       ,
        -0.50000003, -0.50000002, -0.50000004, -0.50000003, -0.50000003,
        -0.50000003, -0.50000004, -0.50000003, -0.5       , -0.50000003,
        -0.5       , -0.5       , -0.5       , -0.50000001, -0.5       ,
        -0.50000002, -0.49999998, -0.50000002, -0.49999998, -0.5       ,
        -0.50000003, -0.50000003, -0.49999999, -0.50000004, -0.50000004,
        -0.50000002, -0.50000003, -0.50000003, -0.49999998, -0.50000002,
        -0.50000001, -0.50000004, -0.49999998, -0.50000004, -0.49999998,
        -0.50000004, -0.49999998, -0.49999998, -0.50000002, -0.49999999,
        -0.5       , -0.5       , -0.50000003, -0.50000002, -0.50000003,
        -0.49999999, -0.49999998, -0.5       , -0.5       , -0.5       ,
        -0.49999998, -0.49999999, -0.5       , -0.49999999, -0.49999998,
        -0.50000001, -0.50000003, -0.49999999, -0.5       , -0.50000003,
        -0.49999998, -0.50000003, -0.49999999, -0.5       , -0.50000003,
        -0.50000003, -0.5       , -0.50000004, -0.50000003, -0.49999999,
        -0.50000003]])
```