⇒ H 对称正定

3.31. $Ax \cong b$ 的最小二乘解 $x = A^{\dagger}b = V\Sigma^{\dagger}U^{T}b$.

$\Rightarrow x = V\Sigma^{\dagger}(U_1^{T}b, U_2^{T}b, \cdots, U_m^{T}b)$

$= (v_1, v_2, \cdots v_m)\left(\frac{U_1^{T}b}{\sigma_1}, \cdots \frac{U_m^{T}b}{\sigma_m}\right)$, $\sigma_i \neq 0$.

$= \sum_{\sigma_i \neq 0} \frac{U_i^{T}b}{\sigma_i} v_i$

(由于 σ_i 等0时，对应广义逆为0).

3.32. (a) $AA^{\dagger}A = A \Leftrightarrow U\Sigma\Sigma^{\dagger}\Sigma V^{T} = U\Sigma V^{T}$.

由于 $\Sigma\Sigma^{\dagger}\Sigma = \Sigma$. ⇒ 结论成立.

(b). $\Sigma^{\dagger}\Sigma\Sigma^{\dagger} = \Sigma^{\dagger}$. ⇒ $A^{\dagger}AA^{\dagger} = A^{\dagger}$.

(c) 由于 $\Sigma\Sigma^{\dagger}$ 为对角阵. ⇒ $(\Sigma\Sigma^{\dagger})^{T} = (\Sigma\Sigma^{\dagger})$

⇒ 结论成立

(d) 同上，结论成立。

Hw4.

4.13. 设 $AX_0 = \lambda_{max}X_0$, $\|X_0\| = 1$

$\Rightarrow \|A\| = \sup_{\|x\|=1} \|Ax\| \geq \lambda_{max}$. 即 $\rho(A) \leq \|A\|$

4.19 $Ax = \lambda x$, ⇒ $\lambda^2 x = A^2 x = Ax = \lambda x$

⇒ $\lambda = 1$ 或 0.

4.20. (a) $Ax = \lambda x$, $Ay = \mu y$. ⇒ $y^{H}Ax = \mu y^{H}x$

$x^{H}Ay = \lambda x^{H}y$

科目：

上海学生统一课业簿册

$\Rightarrow y^H x = 0$.

(b) $y^H A x = \mu y^H x$

另一方面，$y^H A x = y^H (Ax) = \lambda y^H x$.

(c) 考查 A 的 Jordan 标准型.

$\exists$ 可逆矩阵 P, st $P^{-1}AP = \begin{pmatrix} \lambda & \\ & J \end{pmatrix}$.

$\Rightarrow AP = P\begin{bmatrix} \lambda & \\ & J \end{bmatrix}$ $\Rightarrow$ P 的第一列为 λ 的右特征值.

$P^{-1}A = \begin{bmatrix} \lambda & \\ & J \end{bmatrix} P^{-1}$, $\Rightarrow P^{-1}$ 的第一行为 λ 的左特征值.

由于 $P^{-1}P = I$. $\Rightarrow y^H x \neq 0$

4.24

(a) $A = U\Sigma V^T = U \begin{pmatrix} \sqrt{\sigma} & 0 \\ 0 & \ddots \end{pmatrix} \begin{pmatrix} \sqrt{\sigma} & 0 \\ 0 & \ddots \end{pmatrix} V^T$

$= \sqrt{\sigma} u_1 (\sqrt{\sigma} v_1)^T$

令 $u = \sqrt{\sigma} u_1$, $v = \sqrt{\sigma} v_1$, 即可.

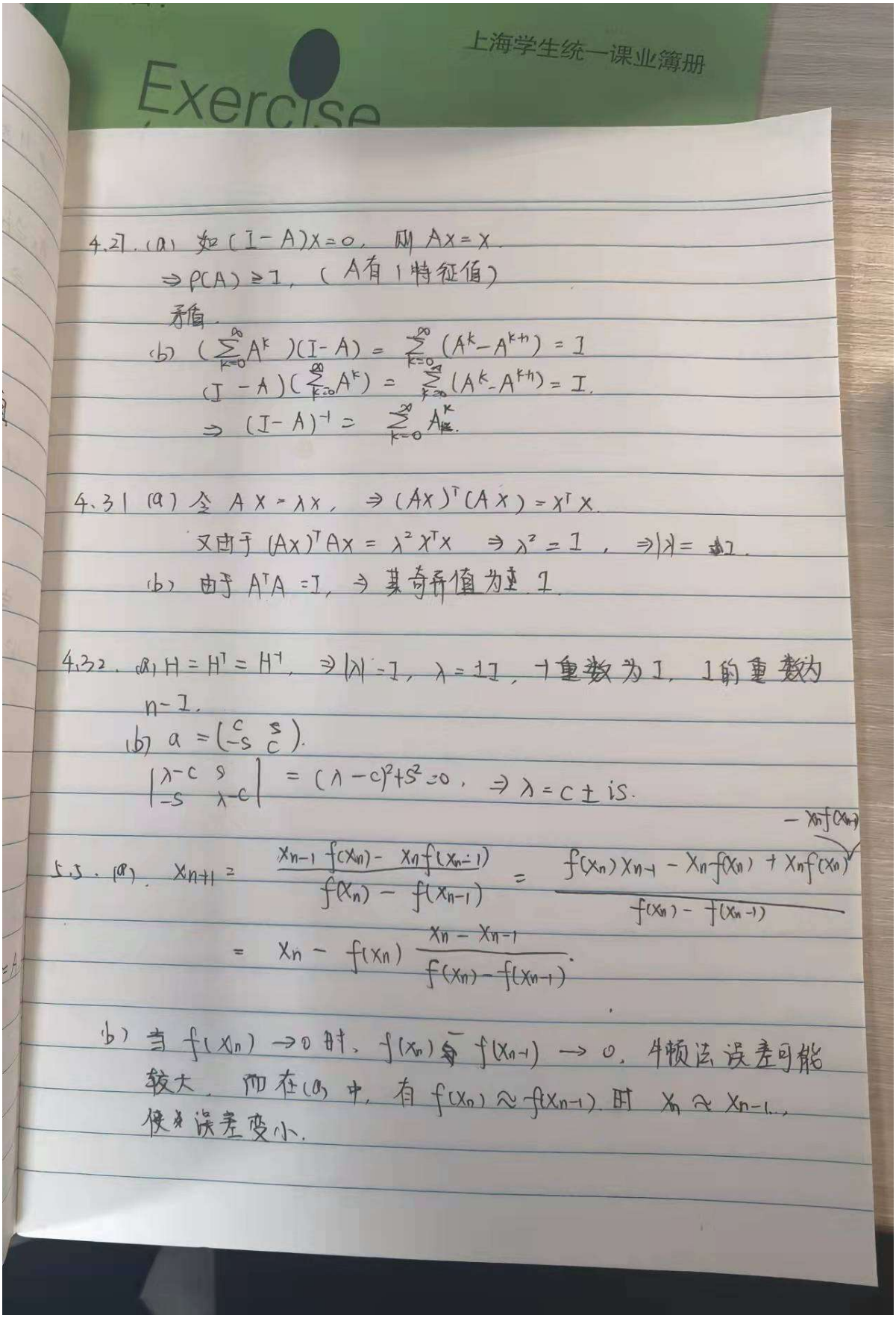
(b) $Au = uv^T u = (v^T u) \cdot u$ $\Rightarrow v^T u = u^T v$ 的特征值

(c) 0.

(d) $\forall x$, $Ax = uv^T x = u(v^T x)$, $\Rightarrow$ 一步.

4.25. 由于 $r(A) = 1$, $\Rightarrow \exists$ 正交阵 U, st $U^T \begin{pmatrix} \sigma & \\ & 0 \end{pmatrix} U = A$.

$\Rightarrow \det(I + uv^T) = 1 + \sigma = 1 + u^T v$.



T1

Computer problem (in C or C++): Write the functions to achieve (1) Arnoldi iteration and Lanczos iteration; (2) QR iteration for Hessenburg matrix to find the eigenvalue decomposition. Test these algorithms for a few matrix.

Arnoldi iteration

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import circulant,solve

def arnoldi_iteration(A, b, n: int):
    """Computes a basis of the (n + 1)-Krylov subspace of A: the space
    spanned by {b, Ab, ..., A^n b}.

    Arguments
      A: m × m array
      b: initial vector (length m)
      n: dimension of Krylov subspace, must be >= 1

    Returns
      Q: m x (n + 1) array, the columns are an orthonormal basis of the
        Krylov subspace.
      h: (n + 1) x n array, A on basis Q. It is upper Hessenberg.
    """
    eps = 1e-12
    h = np.zeros((n+1,n), dtype=np.complex)
    Q = np.zeros((A.shape[0],n+1), dtype=np.complex)
     # Normalize the input vector
    Q[:,0] =b/np.linalg.norm(b,2)    # Use it as the first Krylov vector
    for k in range(1,n+1):
        v = np.dot(A,Q[:,k-1])  # Generate a new candidate vector
        for j in range(k):  # Subtract the projections on previous vectors
            h[j,k-1] = np.dot(Q[:,j].conj().T, v)
            v = v - h[j,k-1] * Q[:,j]
        h[k,k-1] = np.linalg.norm(v,2)
        if h[k,k-1] > eps:  # Add the produced vector to the list, unless
            Q[:,k] = v/h[k,k-1]
        else:  # If that happens, stop iterating.
            return Q, h
    return Q, h
```

Lanczos iteration

In [ ]:

```
def lanczos_iteration(A, b, n: int):
    eps = 1e-12
    beta = 0
    Q = np.zeros((A.shape[0],n+2), dtype=np.complex)
    h = np.zeros((n+1,n), dtype=np.complex)
     # Normalize the input vector
    Q[:,1] =b/np.linalg.norm(b,2)    # Use it as the first Krylov vector
    for k in range(2,n+2):
        v = np.dot(A,Q[:,k-1])  # Generate a new candidate vector
        alpha = Q[:,k-1].conj().T@v
        v = v - beta*Q[:,k-2]-alpha*Q[:,k-1]
        bata = np.linalg.norm(v,2)
        h[k-2,k-2] = alpha
        h[k-1,k-2] = beta
        if k < n+1:
            h[k-2,k-3]=beta
        if bata > eps:  # Add the produced vector to the list, unless
            Q[:,k] = v/bata
        else:  # If that happens, stop iterating.
            return Q[:,1:],h
    return Q[:,1:],h
```

In [ ]:

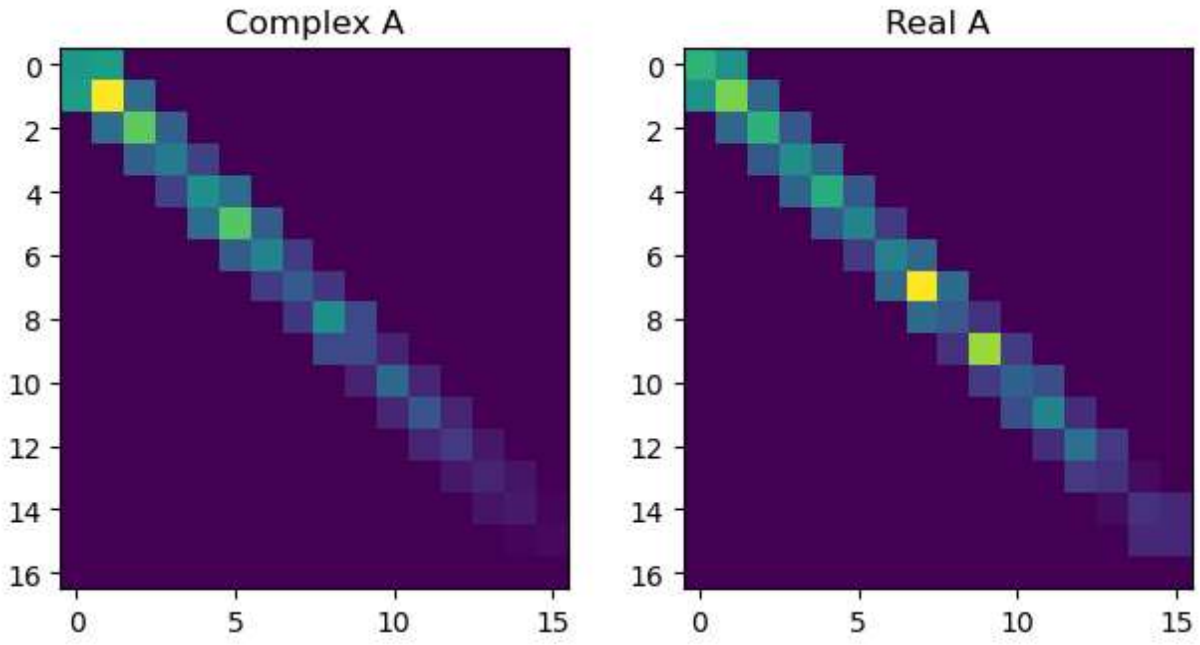
```
N = 2**4
I = np.eye(N)
k = np.fft.fftfreq(N, 1.0 / N) + 0.5
alpha = np.linspace(0.1, 1.0, N)*2e2
c = np.fft.fft(alpha) / N
C = circulant(c)
A = np.einsum("i, ij, j->ij", k, C, k)

# Show that A is Hermitian
print(np.allclose(A, A.conj().T))

# Arbitrary (random) initial vector
np.random.seed(0)
v = np.random.rand(N)
# Perform Arnoldi iteration with complex A
_, h = arnoldi_iteration(A, v, N)
# Perform Arnoldi iteration with real A
_, h2 = arnoldi_iteration(np.real(A), v, N)

# Plot results
plt.subplot(121)
plt.imshow(np.abs(h))
plt.title("Complex A")
plt.subplot(122)
plt.imshow(np.abs(h2))
plt.title("Real A")
plt.tight_layout()
plt.show()
```

```
True
<ipython-input-1-90a6c109bec1>:20: DeprecationWarning: `np.complex` is a deprecated al
ias for the builtin `complex`. To silence this warning, use `complex` by itself. Doing
this will not modify any behavior and is safe. If you specifically wanted the numpy sc
alar type, use `np.complex128` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/rel
ease/1.20.0-notes.html#deprecations
  h = np.zeros((n+1,n), dtype=np.complex)
<ipython-input-1-90a6c109bec1>:21: DeprecationWarning: `np.complex` is a deprecated al
ias for the builtin `complex`. To silence this warning, use `complex` by itself. Doing
this will not modify any behavior and is safe. If you specifically wanted the numpy sc
alar type, use `np.complex128` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/rel
ease/1.20.0-notes.html#deprecations
  Q = np.zeros((A.shape[0],n+1), dtype=np.complex)
```

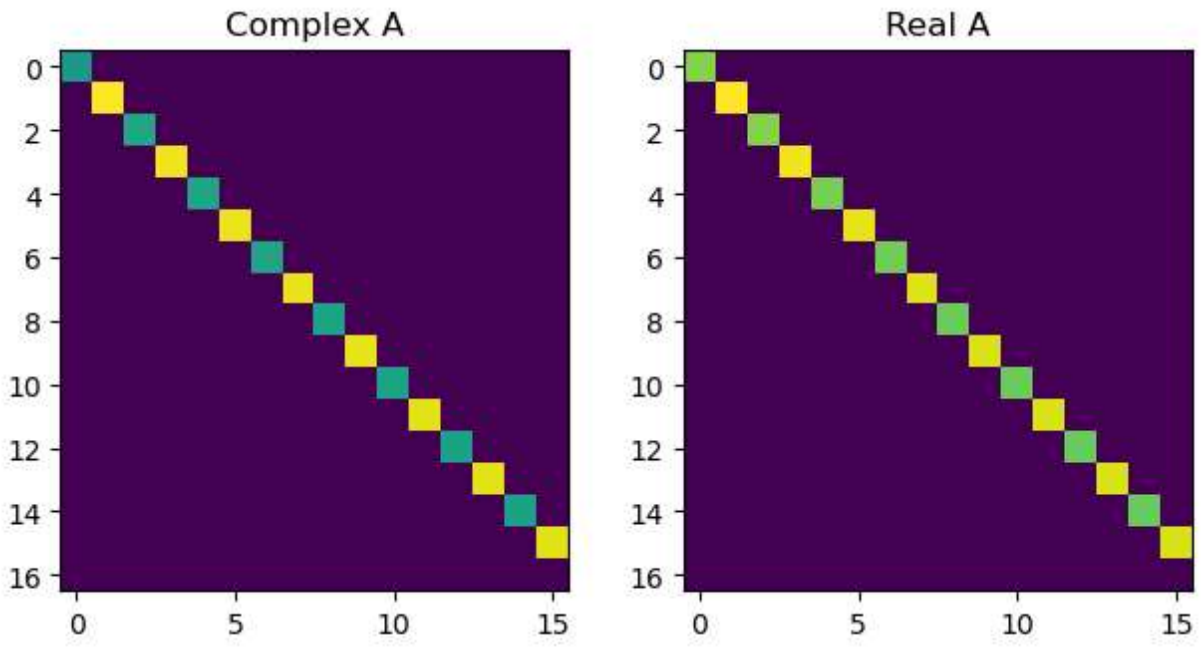


In [ ]:

```
_, h = lanczos_iteration(A, v, N)
# Perform Arnoldi iteration with real A
_, h2 = lanczos_iteration(np.real(A), v, N)

# Plot results
plt.subplot(121)
plt.imshow(np.abs(h))
plt.title("Complex A")
plt.subplot(122)
plt.imshow(np.abs(h2))
plt.title("Real A")
plt.tight_layout()
plt.show()
```

```
<ipython-input-2-6343734e5b98>:4: DeprecationWarning: `np.complex` is a deprecated ali
as for the builtin `complex`. To silence this warning, use `complex` by itself. Doing
this will not modify any behavior and is safe. If you specifically wanted the numpy sc
alar type, use `np.complex128` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/rel
ease/1.20.0-notes.html#deprecations
  Q = np.zeros((A.shape[0],n+2), dtype=np.complex)
<ipython-input-2-6343734e5b98>:5: DeprecationWarning: `np.complex` is a deprecated ali
as for the builtin `complex`. To silence this warning, use `complex` by itself. Doing
this will not modify any behavior and is safe. If you specifically wanted the numpy sc
alar type, use `np.complex128` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/rel
ease/1.20.0-notes.html#deprecations
  h = np.zeros((n+1,n), dtype=np.complex)
```



QR iteration for Hessenburg matrix to find the eigenvalue decomposition.

In [ ]:

```
def Q_i(Q_min, i, j, k):
    if i < k or j < k:
        return float(i == j)
    else:
        return Q_min[i-k][j-k]

def QR_householder(A):
    n = A.shape[0]

    # Set R equal to A, and create Q as a zero matrix of the same size
    R = A.copy()
    Q = np.zeros((n,n))

    # The Householder procedure
    for k in range(n-1):
        I = np.eye(n)
        x = R[k:,k]
        e = I[k:,k]
        alpha = np.sign(x[0]) * np.linalg.norm(x)

        # Using anonymous functions, we create u and v
        u = np.array(list(map(lambda p,q: p + alpha * q, x, e)))
        print(u)
        norm_u = np.linalg.norm(u)
        v = np.array(list(map(lambda p: p/norm_u, u)))

        # Create the Q minor matrix
        Q_min = np.array([ [float(i==j) - 2.0 * v[i] * v[j] for i in range(n-k)] for j

        # "Pad out" the Q minor matrix with elements from the identity
        Q_t = np.array([[ Q_i(Q_min,i,j,k) for i in range(n)] for j in range(n)])

        if k == 0:
            Q = Q_t
            R = Q_t@A
        else:
            Q = Q_t@Q
```

```
            R = Q_t@R

    # Since Q is defined as the product of transposes of Q_t,
    # we need to take the transpose upon returning it
    return Q.T, R
```

T2

Computer problem (in C or C++): Write the functions to achieve Newton's method and Broyden's method. Test these methods for a few equation systems.

In [ ]:

```
from scipy import misc
from scipy.optimize import newton

# Newton's Method
def NewtonsMethod(f, x, tolerance=0.000001):
    while True:
        x1 = x - f(x) / misc.derivative(f, x)
        t = abs(x1 - x)
        if t < tolerance:
            break
        x = x1
    return x

# Broyden' s method
def broyden(x, y, f_equations, J_equations, tol=10e-10, maxIters=50):
    steps_taken = 0

    f = f_equations(x,y)
    J = J_equations(x,y)

    while np.linalg.norm(f,2) > tol and steps_taken < maxIters:

        s = solve(J,-1*f)

        x = x + s[0]
        y = y + s[1]
        newf = f_equations(x,y)
        z = newf - f

        J = J + (np.outer ((z - np.dot(J,s)),s)) / (np.dot(s,s))

        f = newf
        steps_taken += 1

    return steps_taken, x, y
```

In [ ]:

```
def f(x):
    return (1.0/4.0)*x**3+(3.0/4.0)*x**2-(3.0/2.0)*x-2

x = 4

x_0 = NewtonsMethod(f, x)
x_1 = newton(f, x, fprime=None, args=(), tol=1.48e-08, maxiter=50, fprime2=None)
```

In [ ]:

```
x_0,x_1
```

Out[ ]: (2.0000002745869883, 2.000000000000008)

In [ ]:
```
tol = 10.0** -15
maxIters = 50
x0 = 1
y0 = 2

def fs(x,y):
    return np.array([x + 2*y - 2, x**2 + 4*y**2 - 4])

def Js(x,y):
    return np.array([[1,2],
                     [2, 16]])

n, x, y = broyden(x0, y0, fs, Js, tol, maxIters=50 )
print("iterations: ", n)
print("x and y: ", x, y)
```

```
iterations:  8
x and y:  8.159067075518965e-17 0.9999999999999999
```