Problem 1 Code

```
import numpy as np
from scipy.sparse import csc_matrix, diags
from scipy.sparse.linalg import spsolve, gmres
import scipy as scp
from scipy import linalg as scplinalg
import matplotlib.pyplot as plt

def construct_1D_laplace(m):
   h = 1./m
   diagonal = 2. * np.ones(m)
   off_diag = -1. * np.ones(m-1)
   mat = np.diag(diagonal, 0) + np.diag(off_diag, 1) + np.diag(off_diag, -1)
   mat[0, -1] = -1.0
   mat[-1, 0] = -1.0
   return (1./(h*h)) * mat

def construct_full_rank_1D_laplace(m):
   h = 1./m
   diagonal = 2. * np.ones(m)
   off_diag = -1. * np.ones(m-1)
   mat = np.diag(diagonal, 0) + np.diag(off_diag, 1) + np.diag(off_diag, -1)
   mat[0, -1] = -1.0
   mat[-1, 0] = -1.0
   mat_fr = np.zeros((m+1, m+1))
   mat_fr[0:m, 0:m] = mat[:, :]
   mat_fr[:, m] = 1.0
   mat_fr[m, :] = 1.0
   mat_fr[m, m] = -1.0
   return (1./(h*h)) * mat_fr

def sample_points_1D(m):
   h = 1./m
   x = np.linspace(0, 1-h, num=m)
   return x, np.zeros_like(x)

def get_rhs(x, const):
   return np.sin(const*np.pi*x)

def compute_residual_norm(A, u, f):
   return np.linalg.norm(f - np.dot(A, u), np.inf)
```

```python
# LU Decomposition
def get_lu_decomposition(mat):
   p, l, u = scp.linalg.lu(mat, permute_l=False)
   return p, l, u

# QR Decomposition
def get_QR_decomposition(mat):
   q, r = np.linalg.qr(mat)
   return q, r

def solve_using_lu(p, l, u, b):
   b1 = np.matmul(np.transpose(p), b)
   y = scplinalg.solve_triangular(l, b1, lower=True)
   x = scplinalg.solve_triangular(u, y, lower=False)
   return x

def solve_using_qr(q, r, b):
   y = np.matmul(np.transpose(q), b)
   x = scplinalg.solve_triangular(r, y)
   return x

def get_sparse_matrix(A):
   return csc_matrix(A)

def solve_using_gmres(A, b, x0, tolerance):
   x = gmres(A, b, x0, tolerance)
   return x

def get_exact_solution(x):
   return (1./(16. * np.pi * np.pi)) * np.sin(4. * np.pi * x)

def plot_solutions(u_lu, u_qr, u_gmres, u_exact):
   plt.plot(u_lu, label='Solution from LU decomposition')
   # plt.plot(u_qr, '.', label='Solution from QR decomposition')
   # plt.plot(u_gmres, 'o-', label='Solution using GMRES')
   plt.plot(u_exact,'--', label='Exact solution with mean 0')
   plt.legend()
   plt.xlabel('x')
   plt.ylabel('u(x)')
   plt.title('Problem 1a: Plot for u(x)')
```

```python
    plt.savefig('1a.png')

m = 100
A = construct_1D_laplace(m)
x, u = sample_points_1D(m)
f = get_rhs(x, 4.0)
print("Residual before solve = {}".format(compute_residual_norm(A, u, f)))

p, Al, Au = get_lu_decomposition(A)
u_lu = solve_using_lu(p, Al, Au, f)
print("Residual after LU solve = {}".format(compute_residual_norm(A, u_lu, f)))

Aq, Ar = get_QR_decomposition(A)
u_qr = solve_using_qr(Aq, Ar, f)
print("Residual after QR solve = {}".format(compute_residual_norm(A, u_qr, f)))

u_exact = get_exact_solution(x)

A_sparse = get_sparse_matrix(A)
tolerance = 1e-10
u_gmres, gmres_error = solve_using_gmres(A_sparse, f, u, tolerance)
assert(gmres_error == 0)
print("Residual after GMRES solve = {}".format(compute_residual_norm(A, u_gmres,
f)))

# Apply periodic boundary condition
u_lu = np.insert(u_lu, len(u_lu), u_lu[0])
u_qr = np.insert(u_qr, len(u_qr), u_qr[0])
u_exact = np.insert(u_exact, len(u_exact), u_exact[0])
u_gmres = np.insert(u_gmres, len(u_gmres), u_gmres[0])

plot_solutions(u_lu, u_qr, u_gmres, u_exact)

plt.clf()
plt.cla()

# plt.plot(u_lu, label='Solution from LU decomposition')
plt.plot(u_qr, label='Solution from QR decomposition')
# plt.plot(u_gmres, 'o-', label='Solution using GMRES')
plt.plot(u_exact,'--', label='Exact solution with mean 0')
plt.legend()
```

```
plt.xlabel('x')
plt.ylabel('u(x)')
plt.title('Problem 1b: Plot for u(x)')
plt.savefig('1b.png')

print(np.linalg.matrix_rank(A))

A_fr = construct_full_rank_1D_laplace(m)
print(np.linalg.matrix_rank(A_fr))

f_fr = np.insert(f, len(f), 0.0)
u_fr = np.linalg.solve(A_fr, f_fr)

print("Residual after np solve = {}".format(compute_residual_norm(A_fr, u_fr,
f_fr)))

plt.clf()
plt.cla()

# plt.plot(u_lu, label='Solution from LU decomposition')
plt.plot(u_fr, label='Solution after removing null space')
# plt.plot(u_gmres, 'o-', label='Solution using GMRES')
plt.plot(u_exact,'--', label='Exact solution with mean 0')
plt.legend()
plt.xlabel('x')
plt.ylabel('u(x)')
plt.title('Problem 1d: Plot for u(x)')
plt.savefig('1d.png')

plt.clf()
plt.cla()

# plt.plot(u_lu, label='Solution from LU decomposition')
# plt.plot(u_qr, label='Solution from QR decomposition')
plt.plot(u_gmres, label='Solution using GMRES')
plt.plot(u_exact,'--', label='Exact solution with mean 0')
plt.legend()
plt.xlabel('x')
plt.ylabel('u(x)')
plt.title('Problem 1e: Plot for u(x)')
plt.savefig('1e.png')
```

Problem 2 Code

```
from random import sample
from statistics import median_low
import numpy as np
from scipy.sparse import csc_matrix, diags
from scipy.sparse.linalg import spsolve, gmres
import scipy as scp
from scipy import linalg as scplinalg
import matplotlib.pyplot as plt

def construct_1D_laplace(m):
   h = 1./float(m)
   diagonal = 2. * np.ones(m)
   off_diag = -1. * np.ones(m-1)
   mat = np.diag(diagonal, 0) + np.diag(off_diag, 1) + np.diag(off_diag, -1)
   mat[0, -1] = -1.0
   mat[-1, 0] = -1.0
   return (1./(h*h)) * mat

def construct_forward_difference(m):
   h = 1./float(m)
   diagonal = -1. * np.ones(m)
   off_diag = 1. * np.ones(m-1)
   mat = np.diag(diagonal, 0) + np.diag(off_diag, 1)
   mat[-1, 0] = 1.0
   return (1./h) * mat

def construct_backward_difference(m):
   h = 1./float(m)
   diagonal = 1. * np.ones(m)
   off_diag = -1. * np.ones(m-1)
   mat = np.diag(diagonal, 0) + np.diag(off_diag, -1)
   mat[0, -1] = -1.0
   return (1./h) * mat

def construct_full_rank_1D_laplace(m):
   h = 1./float(m)
   diagonal = 2. * np.ones(m)
   off_diag = -1. * np.ones(m-1)
   mat = np.diag(diagonal, 0) + np.diag(off_diag, 1) + np.diag(off_diag, -1)
```

```
    mat[0, -1] = -1.0
    mat[-1, 0] = -1.0
    mat_fr = np.zeros((m+1, m+1))
    mat_fr[0:m, 0:m] = mat[:, :]
    mat_fr[:, m] = 1.0
    mat_fr[m, :] = 1.0
    mat_fr[m, m] = -1.0
    return (1./(h*h)) * mat_fr

def sample_points_1D(m):
    h = 1./float(m)
    x = np.linspace(0, 1-h, num=m)
    return x, np.zeros_like(x)

def get_f(x):
    return np.sin(4.*np.pi*x)

def get_f_prime(x):
    return 4. * np.pi * np.cos(4.*np.pi*x)

def get_f_4prime(x):
    return np.power(4.0 * np.pi, 4) * np.sin(4.*np.pi*x)

def compute_residual_norm(A, u, f):
    return np.linalg.norm(f - np.dot(A, u), np.inf)

# LU Decomposition
def get_lu_decomposition(mat):
    p, l, u = scp.linalg.lu(mat, permute_l=False)
    return p, l, u

def solve_using_lu(p, l, u, b):
    b1 = np.matmul(np.transpose(p), b)
    y = scplinalg.solve_triangular(l, b1, lower=True)
    x = scplinalg.solve_triangular(u, y, lower=False)
    return x

def lu_solve(A, f):
    p, Al, Au = get_lu_decomposition(A)
    u_lu = solve_using_lu(p, Al, Au, f)
    print("Residual after LU solve = {}".format(compute_residual_norm(A, u_lu, f)))
```

```
    return u_lu

# GMRES Decomposition
def get_sparse_matrix(A):
    return csc_matrix(A)

def gmres_solve(A, b, x0, tolerance):
    A_sparse = get_sparse_matrix(A)
    x, gmres_error = gmres(A_sparse, b, x0, tolerance)
    assert(gmres_error == 0)
    return x

def run_experiment_a():
    m_list = np.arange(1, 21) * 10
    error = np.zeros_like(m_list).astype(float)
    h = np.zeros_like(m_list).astype(float)
    for i, m in enumerate(m_list):
        x, _ = sample_points_1D(m)
        f = get_f(x)
        f_prime = get_f_prime(x)
        D_plus = construct_forward_difference(m)
        D_plus_f = np.matmul(D_plus, f)
        h[i] = 1./float(m)
        error[i] = np.linalg.norm(f_prime-D_plus_f, 2) * np.sqrt(h[i])
    print( 'Problem 2a: ', (np.log(error[-1]) - np.log(error[-2]))/(np.log(h[-1]) -
np.log(h[-2])))
    # plt.loglog(h, error)
    # plt.show()
    return h, error

def run_experiment_b():
    m_list = np.arange(1, 21) * 10
    error = np.zeros_like(m_list).astype(float)
    h = np.zeros_like(m_list).astype(float)
    for i, m in enumerate(m_list):
        x, _ = sample_points_1D(m)
        f = get_f(x)
        f_prime = get_f_prime(x)
        D_plus = construct_forward_difference(m)
        D_minus = construct_backward_difference(m)
        D1 = 0.5 * (D_plus + D_minus)
```

```
        D1f = np.dot(D1, f)
        h[i] = 1./float(m)
        error[i] = np.linalg.norm(f_prime-D1f, 2) * np.sqrt(h[i])
    print( 'Problem 2b: ', (np.log(error[-1]) - np.log(error[-2]))/(np.log(h[-1]) -
np.log(h[-2])))
    return error

def run_experiment_c():
    m_list = np.arange(1, 21) * 10
    for i, m in enumerate(m_list):
        x, _ = sample_points_1D(m)
        f = get_f(x)
        f_prime = get_f_prime(x)
        D_plus = construct_forward_difference(m)
        D_minus = construct_backward_difference(m)
        A = -1.0 * construct_1D_laplace(m)
        D2 = np.matmul(D_plus, D_minus)
        D_1 = 0.5 * (D_plus + D_minus)
        D_1_squared = np.matmul(D_1, D_1)
        print(m, '\t', np.linalg.norm(D2-A), '\t', np.linalg.norm(D2-D_1_squared))

def run_experiment_d():
    m_list = np.arange(1, 21) * 10
    error = np.zeros_like(m_list).astype(float)
    h = np.zeros_like(m_list).astype(float)
    for i, m in enumerate(m_list):
        x, _ = sample_points_1D(m)
        f = get_f(x)
        f_4prime = get_f_4prime(x)
        D_plus = construct_forward_difference(m)
        D_minus = construct_backward_difference(m)
        D2 = -1.0 * construct_1D_laplace(m) #np.matmul(D_plus, D_minus)
        D4 = np.matmul(D2, D2)
        D4f = np.matmul(D4, f)
        h[i] = 1./float(m)
        error[i] = np.linalg.norm( f_4prime - D4f, 2) * np.sqrt(h[i])
    print( 'Problem 2d: ', (np.log(error[-1]) - np.log(error[-2]))/(np.log(h[-1]) -
np.log(h[-2])))
    # plt.loglog(h, error)
    # plt.xlabel('log h')
    # plt.ylabel('log e')
```

```
    # plt.show()
    return error

def run_experiment_e():
    m = 100
    x, u = sample_points_1D(m)
    f = get_f(x)

    D_plus = construct_forward_difference(m)
    D_minus = construct_backward_difference(m)

    D1 = 0.5 * (D_plus + D_minus)
    D2 = np.matmul(D_plus, D_minus)
    D4 = np.matmul(D2, D2)

    A = D4 - D2 + D1

    u_lu = lu_solve(A, f)

    tolerance = 1e-10
    u_gmres = gmres_solve(A, f, u, tolerance)

    u_lu = np.insert(u_lu, len(u_lu), u_lu[0])
    u_gmres = np.insert(u_gmres, len(u_gmres), u_gmres[0])

    print(A.shape, np.linalg.matrix_rank(A))

    plt.clf()
    plt.cla()

    plt.plot(u_lu, label='Solution from LU decomposition')
    plt.plot(u_gmres,'--', label='Solution from GMRES')
    plt.legend()
    plt.xlabel('x')
    plt.ylabel('u(x)')
    plt.title('Problem 2e: Plot for u(x)')
    plt.savefig('2e.png')


def run_experiment_f():
    m = 100
```

```python
    x, u = sample_points_1D(m)
    f = get_f(x)

    D_plus = construct_forward_difference(m)
    D_minus = construct_backward_difference(m)

    D1 = 0.5 * (D_plus + D_minus)
    D2 = np.matmul(D_plus, D_minus)
    D4 = np.matmul(D2, D2)

    A = D4 - D2 + D1

    A_fr = np.zeros((m+1, m+1))
    A_fr[0:m, 0:m] = A[:, :]
    A_fr[m, :] = 1.0
    A_fr[:, m] = 1.0
    A_fr[m, m] = 0.0

    f_fr = np.insert(f, len(f), 0.0)
    u_fr = lu_solve(A_fr, f_fr)

    print(np.linalg.matrix_rank(A), np.linalg.matrix_rank(A_fr))

    # plt.plot(u_fr)
    # # plt.plot(u_gmres)
    # plt.show()
    plt.clf()
    plt.cla()

    plt.plot(u_fr, label='Solution after removing nullspace')
    plt.legend()
    plt.xlabel('x')
    plt.ylabel('u(x)')
    plt.title('Problem 2f: Plot for u(x)')
    plt.savefig('2f.png')

h, ea = run_experiment_a()
eb = run_experiment_b()
run_experiment_c()
ed = run_experiment_d()
```

```python
plt.cla()
plt.clf()
plt.loglog(h, ea, label="Problem 2a: e = ||f' - D+(f)||")
plt.legend()
plt.xlabel('log h')
plt.ylabel('log e')
plt.title("Problem 2: Approximation error of derivatives")
plt.savefig('2a.png')

plt.cla()
plt.clf()
plt.loglog(h, eb, label="Problem 2b: e = ||f'' - D2(f)||")
plt.legend()
plt.xlabel('log h')
plt.ylabel('log e')
plt.title("Problem 2: Approximation error of derivatives")
plt.savefig('2b.png')

plt.cla()
plt.clf()
plt.loglog(h, ed, label="Problem 2d: e = ||f'''' - D4(f)||")
plt.legend()
plt.xlabel('log h')
plt.ylabel('log e')
plt.title("Problem 2: Approximation error of derivatives")
plt.savefig('2d.png')

run_experiment_e()

run_experiment_f()
```