为了求解给定的偏微分方程 (PDE)，即

$$\begin{cases} -\epsilon \Delta u(x,y) + \dfrac{\partial u}{\partial x}(x,y) = \sin(\pi x)\sin(\pi y) \quad (x,y) \in \Omega \\ u|_{\partial\Omega} = 0, \end{cases} \tag{1}$$

其中 $0 < \epsilon \ll 1$，且 $\Omega = [0,1] \times [0,1]$，我们需要构造一种数值方法，对于 $\epsilon \ll h$ 时仍能保持高精度。

我们可以采用有限差分法来离散化该PDE，并应用紧致格式来保持高精度。

1. 离散化域

将区域 Ω 分成一个 $N \times N$ 的网格，网格步长为 $h = \frac{1}{N-1}$。

令 $x_i = ih$ 和 $y_j = jh$ (其中 $i, j = 0, 1, \ldots, N-1$)。

2. 离散化方程

方程的离散形式为：

$$-\epsilon \Delta u_{i,j} + \frac{\partial u_{i,j}}{\partial x} = \sin(\pi x_i)\sin(\pi y_j) \tag{2}$$

其中，$\Delta u_{i,j}$ 和 $\frac{\partial u_{i,j}}{\partial x}$ 的差分格式分别为：

$$\Delta u_{i,j} \approx \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2} \tag{3}$$

$$\frac{\partial u_{i,j}}{\partial x} \approx \frac{u_{i+1,j} - u_{i-1,j}}{2h} \tag{4}$$

因此，离散化后的方程为：

$$-\epsilon \left(\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2} \right) + \frac{u_{i+1,j} - u_{i-1,j}}{2h} = \sin(\pi x_i)\sin(\pi y_j) \tag{5}$$

3. 紧致格式

为了在 $\epsilon \ll h$ 时保持高精度，我们可以采用五点紧致差分格式来离散化二阶导数项。具体为：

$$\Delta u_{i,j} \approx \frac{1}{12h^2}(-u_{i+2,j} + 16u_{i+1,j} - 30u_{i,j} + 16u_{i-1,j} - u_{i-2,j}) + \frac{1}{12h^2}(-u_{i,j+2} + 16u_{i,j+1} - 30u_{i,j} +$$

一阶导数项仍用中心差分格式：

$$\frac{\partial u_{i,j}}{\partial x} \approx \frac{u_{i+1,j} - u_{i-1,j}}{2h} \tag{7}$$

4. 边界条件

由于 $u|_{\partial\Omega} = 0$，所以在边界上的值为零，即：

$$u_{i,j} = 0 \quad \text{if } i = 0, i = N-1, j = 0, j = N-1 \tag{8}$$

5. 离散方程的求解

将离散方程转换为线性方程组 $Ax = b$ 的形式，其中 A 是系数矩阵，x 是未知量向量，b 是已知量向量。

我们可以使用迭代方法（如 Gauss-Seidel 迭代法或共轭梯度法）来求解该线性方程组。

6. 实验和比较

选择不同的 ϵ 和步长 h，计算数值解并比较结果。可以通过以下步骤进行比较：

1. 计算不同 ϵ 和 h 的数值解。
2. 比较不同情况下数值解的误差。
3. 分析不同 ϵ 和 h 对结果的影响。

Python 实现

下面是一个使用Python实现上述方法的简单代码框架：

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.sparse import diags
from scipy.sparse.linalg import spsolve

def solve_pde(N, epsilon):
    h = 1.0 / (N - 1)
    x = np.linspace(0, 1, N)
    y = np.linspace(0, 1, N)
    X, Y = np.meshgrid(x, y)

    # Right-hand side function f(x, y) = sin(pi * x) * sin(pi * y)
    f = np.sin(np.pi * X) * np.sin(np.pi * Y)

    # Initialize the solution array
    u = np.zeros((N, N))

    # Construct the coefficient matrix A and right-hand side vector
b
    A = np.zeros((N*N, N*N))
    b = np.zeros(N*N)

    for i in range(1, N-1):
        for j in range(1, N-1):
            idx = i * N + j
            A[idx, idx] = -30 * epsilon / (12 * h**2)
            if i > 1:
                A[idx, idx - N] = 16 * epsilon / (12 * h**2)
                A[idx, idx - 2 * N] = -epsilon / (12 * h**2)
            if i < N - 2:
                A[idx, idx + N] = 16 * epsilon / (12 * h**2)
                A[idx, idx + 2 * N] = -epsilon / (12 * h**2)
            if j > 1:
                A[idx, idx - 1] = 16 * epsilon / (12 * h**2)
                A[idx, idx - 2] = -epsilon / (12 * h**2)
            if j < N - 2:
                A[idx, idx + 1] = 16 * epsilon / (12 * h**2)
                A[idx, idx + 2] = -epsilon / (12 * h**2)
```

```
            A[idx, idx - 1] -= 1 / (2 * h)
            A[idx, idx + 1] += 1 / (2 * h)

            b[idx] = f[i, j]

    # Boundary conditions u = 0 on the edges
    for i in range(N):
        A[i, i] = 1
        b[i] = 0
        A[(N-1)*N + i, (N-1)*N + i] = 1
        b[(N-1)*N + i] = 0
        A[i * N, i * N] = 1
        b[i * N] = 0
        A[i * N + (N - 1), i * N + (N - 1)] = 1
        b[i * N + (N - 1)] = 0

    # Solve the linear system
    u_flat = spsolve(A, b)
    u = u_flat.reshape((N, N))

    return X, Y, u, f

def compute_error(u, u_exact):
    return np.sqrt(np.sum((u - u_exact)**2) / np.size(u))

# Parameters
N = 50
epsilon_values = [1e-2, 1e-4, 1e-6]

for epsilon in epsilon_values:
    X, Y, u, f = solve_pde(N, epsilon)
    plt.figure(figsize=(12, 4))
    plt.subplot(131)
    plt.contourf(X, Y, u, levels=50, cmap='viridis')
    plt.colorbar()
    plt.title(f'Solution with $\epsilon = {epsilon}$')
    plt.xlabel('x')
    plt.ylabel('y')

    plt.subplot(132)
    plt.contourf(X, Y, f, levels=50, cmap='viridis')
    plt.colorbar()
    plt.title('Right-hand side $f(x,y)$')
    plt.xlabel('x')
    plt.ylabel('y')

    plt.subplot(133)
    plt.contourf(X, Y, np.abs(u - np.sin(np.pi * X) * np.sin(np.pi
* Y)), levels=50, cmap='viridis')
    plt.colorbar()
    plt.title('Error $|u - f|$')
    plt.xlabel('x')
    plt.ylabel('y')

    plt.tight_layout()
    plt.show()

    # Compute and print the error
```

```python
    exact_solution = np.sin(np.pi * X) * np.sin(np.pi * Y)
    error = compute_error(u, exact_solution)
    print(f'Error for epsilon = {epsilon}: {error:.2e}')
```

通过不同的 ϵ 值进行计算并比较结果，我们可以观察到不同 ϵ 值对解的影响，从而验证离散格式的高精度。