Assignment 2
Numerical solution of the one-dimensional Poisson's equation with the Finite-Difference Method

Numerical Analysis For PDE's (WI3730TU)
Kaloyan Yanchev

September 2024

$-\frac{\partial^2 u_i}{\partial x^2} = f_i, x \in (0,3); \qquad u_i(0) = 1, u_i(3) = 1, i = 1,2; \qquad f_1(x) = 3x - 2, f_2(x) = x^2 + 3x - 2, x \in [0,3].$

1. $u_1^{ex}(x) = \iint -f_1(x)\,dx^2 = \int -\frac{3x^2}{2} + 2x + a\,dx = -\frac{x^3}{2} + x^2 + ax + b$

   $$\begin{cases} u_1(0) = 1 => b = 1 \\ \\ u_1(3) = 1 => -\frac{27}{2} + 9 + 3a + 1 = 1 => a = \frac{3}{2} \end{cases} => u_1^{ex}(x) = -\frac{x^3}{2} + x^2 + \frac{3x}{2} + 1$$

   $u_2^{ex}(x) = \iint -f_2(x)\,dx^2 = \int -\frac{x^3}{3} - \frac{3x^2}{2} + 2x + a\,dx = -\frac{x^4}{12} - \frac{x^3}{2} + x^2 + ax + b$

   $$\begin{cases} u_2(0) = 1 => b = 1 \\ \\ u_2(3) = 1 => -\frac{81}{12} - \frac{27}{2} + 9 + 3a + 1 = 1 => a = \frac{15}{4} \end{cases} => u_2^{ex}(x) = -\frac{x^4}{12} - \frac{x^3}{2} + x^2 + \frac{15x}{4} + 1$$

2. (a) $h = \frac{b-a}{n} = \frac{3-0}{5} = 0.6$

   2 boundary points: $x_0 = 0, x_5 = 3$

   4 internal points: $x_1 = 0.6, x_2 = 1.2, x_3 = 1.8, x_4 = 2.4$

   8 unknowns: $u_i(x_j), i = 1,2; j = 1,2,3,4$

   (b) $-u''(x_i) = \frac{2u(x_i) - u(x_{i-1}) - u(x_{i+1})}{h^2} + \frac{u^{(4)}(x_i)}{12}h^2 + ...$

   (c) $$\begin{cases} u(x_0) = u(0) = 1 \\ -\frac{1}{h^2}(u(x_0) - 2u(x_1) + u(x_2)) = f(x_1) \\ -\frac{1}{h^2}(u(x_1) - 2u(x_2) + u(x_3)) = f(x_2) \\ -\frac{1}{h^2}(u(x_2) - 2u(x_3) + u(x_4)) = f(x_3) \\ -\frac{1}{h^2}(u(x_3) - 2u(x_4) + u(x_5)) = f(x_4) \\ u(x_5) = u(3) = 1 \end{cases}$$

   $$\begin{cases} -\frac{25}{9}(-2u_1(x_1) + u_1(x_2)) = \frac{116}{45} = 2.5\overline{7} \\ -\frac{25}{9}(u_1(x_1) - 2u_1(x_2) + u_1(x_3)) = \frac{8}{5} = 1.6 \\ -\frac{25}{9}(u_1(x_2) - 2u_1(x_3) + u_1(x_4)) = \frac{17}{5} = 3.4 \\ -\frac{25}{9}(u_1(x_3) - 2u_1(x_4)) = \frac{359}{45} = 7.9\overline{7} \end{cases}$$

   $$\begin{cases} -\frac{25}{9}(-2u_2(x_1) + u_2(x_2)) = \frac{661}{225} = 2.93\overline{7} \\ -\frac{25}{9}(u_2(x_1) - 2u_2(x_2) + u_2(x_3)) = \frac{76}{25} = 3.04 \\ -\frac{25}{9}(u_2(x_2) - 2u_2(x_3) + u_2(x_4)) = \frac{166}{25} = 6.64 \\ -\frac{25}{9}(u_2(x_3) - 2u_2(x_4)) = \frac{3091}{225} = 13.73\overline{7} \end{cases}$$

(d) $A = -\frac{25}{9}\begin{bmatrix} -2 & 1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 1 & -2 \end{bmatrix}, \quad \mathbf{f_1} = \begin{bmatrix} \frac{116}{45} \\ \frac{8}{5} \\ \frac{17}{5} \\ \frac{359}{45} \end{bmatrix}, \quad \mathbf{f_2} = \begin{bmatrix} \frac{661}{225} \\ \frac{76}{25} \\ \frac{166}{25} \\ \frac{3091}{225} \end{bmatrix}$

(e) $Lv_i = \lambda_i v_i \quad => \quad \lambda_i = \frac{4}{h^2} sin^2(\frac{\pi i}{2N})$

$\lambda_1 = \frac{100}{9} sin^2(\frac{\pi}{10}) = 1.0610166979169586$

$\lambda_2 = \frac{100}{9} sin^2(\frac{\pi}{5}) = 3.8387944756947365$

$\lambda_3 = \frac{100}{9} sin^2(\frac{3\pi}{10}) = 7.2723166354163755$

$\lambda_4 = \frac{100}{9} sin^2(\frac{2\pi}{5}) = 10.050094413194152$

(f) $-v_i'' = \lambda_i v_i \quad => \quad \tilde{\lambda}_i = (\frac{\pi i}{D})^2$

$\tilde{\lambda}_1 = \frac{\pi^2}{9} = 1.096622711232151$

$\tilde{\lambda}_2 = \frac{4\pi^2}{9} = 4.386490844928604$

$\tilde{\lambda}_3 = \pi^2 = 9.869604401089358$

$\tilde{\lambda}_4 = \frac{16\pi^2}{9} = 17.545963379714415$

i	λ_i	$\tilde{\lambda}_i$	computed λ_i
1	1.0610166979169586	1.096622711232151	1.0610167
2	3.8387944756947365	4.386490844928604	3.83879448
3	7.2723166354163755	9.869604401089358	7.27231664
4	10.050094413194152	17.545963379714415	10.05009441

3.
```
# Solution of 1D Poisson's equation with FDM
# Kaloyan Yanchev (c) 2024
import numpy as np
import matplotlib.pyplot as plt
```

4. (a)
```
# Create a grid that splits the [0, 3] range in n equal length intervals
# n intervals => n+1 points
xgrid = np.linspace( start: 0.0,  stop: 3.0, n+1)

# Length of single interval over [0, 3]
h = 3 / n
```

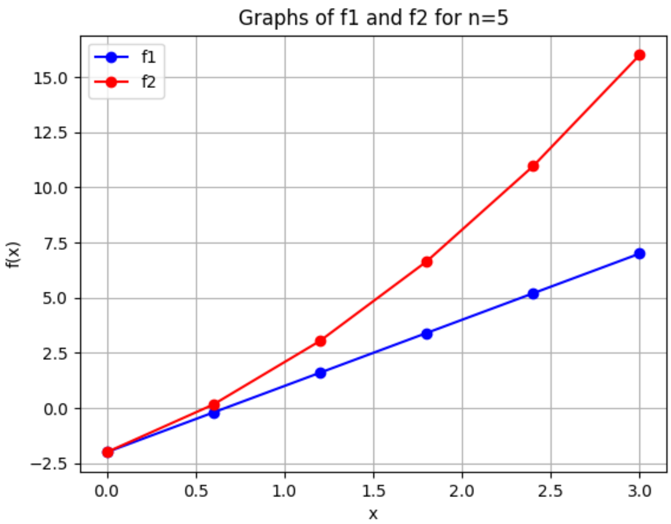
(b)
```
def func1(x):
    return 3*x - 2

def func2(x):
    return x**2 + 3*x - 2
```

(c)
```
# Compute values of f1 and f2 over the grid
f1 = func1(xgrid)
f2 = func2(xgrid)
```

(d)

```
# Create a plot of values of f1 and f2 over the grid
plt.figure()
plt.plot( *args: xgrid, f1, marker='o', label='f1', color='blue')
plt.plot( *args: xgrid, f2, marker='o', label='f2', color='red')
plt.legend()
plt.title("Graphs of f1 and f2 for n=5")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.grid(True)
plt.show()
```



(e)

```
def assignment2(n: int = 5) -> (float, float):
```

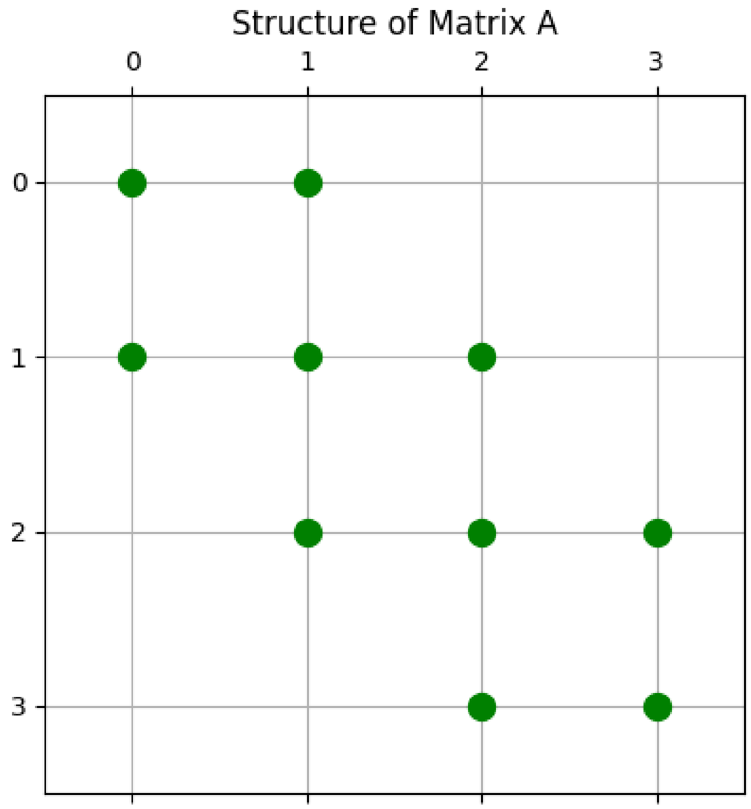
5. (a) $A \in \mathbb{R}^{(N-1)\times(N-1)}$

(b)

```
# Construct coefficient matrix A
a = np.zeros((n-1, n-1))
a = a + np.diag(np.ones((n-2)), k: 1) + np.diag(np.ones((n-2)), -1) + np.diag(np.ones((n-1)), k: 0) * (-2)
a = a * (-1/h**2)
```

(c)

```
# Create a plot of the structure of A
plt.spy(a, marker='o', color='green')
plt.title("Structure of Matrix A")
plt.grid(True)
plt.show()
```



(d)

```
# Compute and print eigenvalues of A
vals = np.linalg.eigvals(a)
vals.sort()
print(vals)
```

6. (a)

```
# Compute the vector values f1 and f2
f1rhs = f1[1:-1]
f1rhs[0] = f1rhs[0] + 1 / h ** 2
f1rhs[-1] = f1rhs[-1] + 1 / h ** 2
f2rhs = f2[1:-1]
f2rhs[0] = f2rhs[0] + 1 / h ** 2
f2rhs[-1] = f2rhs[-1] + 1 / h ** 2
```

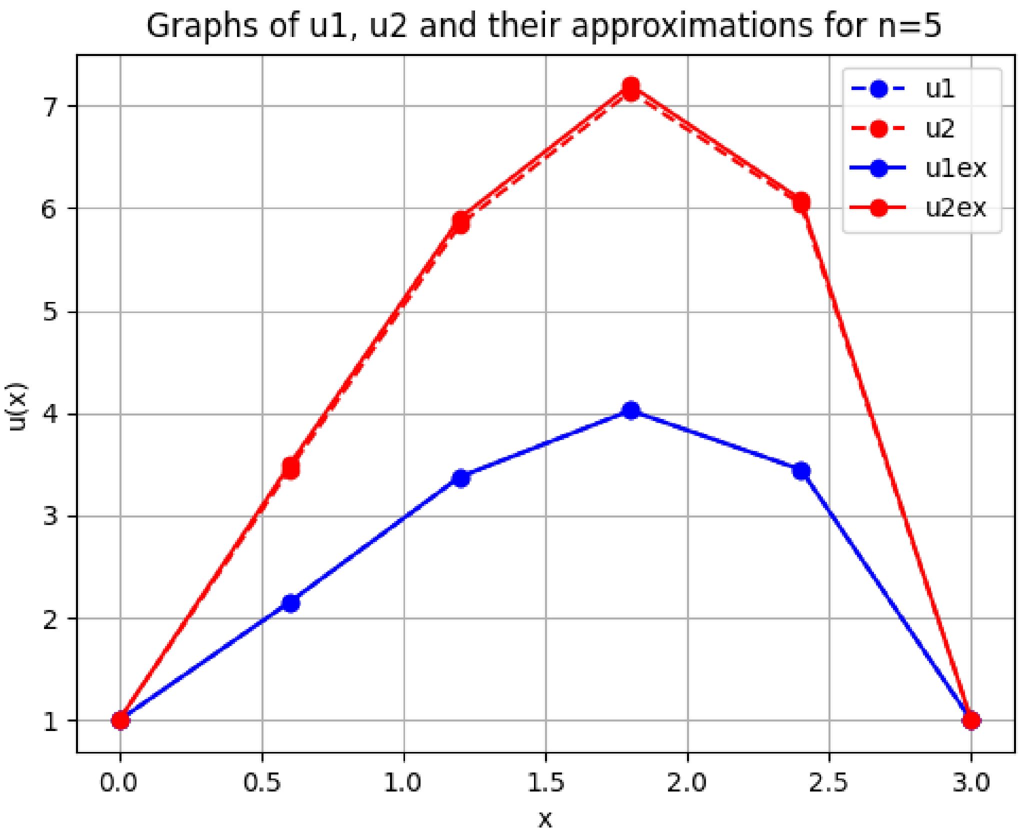
(b)

```
# Solve equation Au=f
u1 = np.linalg.solve(a, f1rhs)
u2 = np.linalg.solve(a, f2rhs)
```

(c)

```
# Insert boundary points into solution
u1 = np.insert(u1, obj: 0, values: 1)
u1 = np.append(u1, values: 1)
u2 = np.insert(u2, obj: 0, values: 1)
u2 = np.append(u2, values: 1)

# Create a plot of u1, u2 and their approximations over the grid
plt.figure()
plt.plot( *args: xgrid, u1, marker='o', label='u1', color='blue', linestyle='--')
plt.plot( *args: xgrid, u2, marker='o', label='u2', color='red', linestyle='--')
plt.plot( *args: xgrid, u1ex, marker='o', label='u1ex', color='blue')
plt.plot( *args: xgrid, u2ex, marker='o', label='u2ex', color='red')
plt.legend()
plt.title("Graphs of u1, u2 and their approximations for n=5")
plt.xlabel("x")
plt.ylabel("u(x)")
plt.grid(True)
plt.show()
```



```python
# Compute the piece-wise difference of u and its approximation
diff1 = u1ex - u1
diff2 = u2ex - u2

# Compute the RMSE global error
err1 = np.sqrt(np.sum(diff1 ** 2) / (n - 1))
err2 = np.sqrt(np.sum(diff2 ** 2) / (n - 1))
```

7. (a)

$error_1 = 8.881784197001252 \text{ x } 10^{-16}$
$error_2 = 0.055069410746804284$

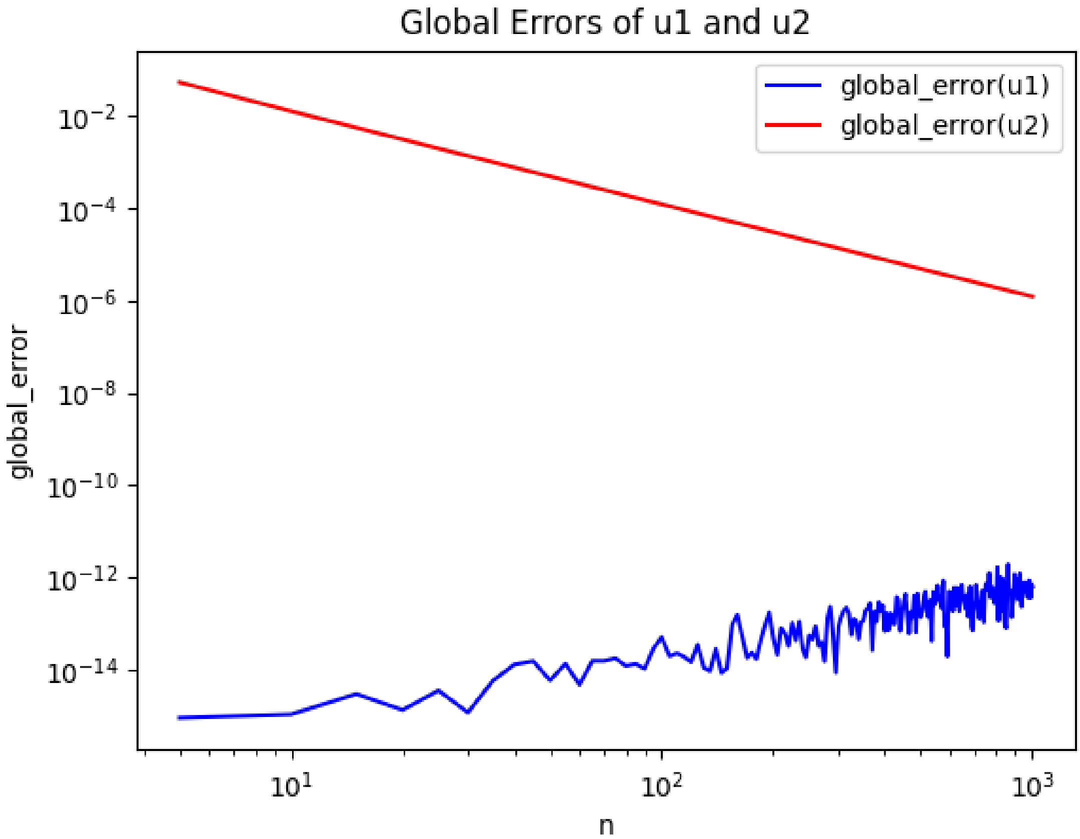
In 2b), the $O(h^2)$ approximation error is specified by its explicit remainder term of $\frac{u^{(4)}(x_i)}{12}$. By knowing u_1^{ex} is a third-degree function, it can be concluded that this $O(h^2)$ error is 0 as $u_1^{ex^{(4)}} = 0$. However, there is still an error of magnitude of 10^{-16} which due to the floating point precision limitations of the computational machine. u_2^{ex}, on the other hand, is a forth-degree function, thus there would be a constant $O(h^2)$ error, which for such a small n, thus relatively high h, would result of error in a much larger magnitude than u_1^{ex}.

```python
# Variables for storing global error convergence
index = []
gl1 = []
gl2 = []

# Iterate n from 5 to 1000 with step 5
for i in range(5, 1001, 5):
    er1, er2 = assignment2(i)
    index.append(i)
    gl1.append(er1)
    gl2.append(er2)

# Plot global error convergence
plt.figure()
plt.loglog( *args: index, gl1, label='global_error(u1)', color='blue')
plt.loglog( *args: index, gl2, label='global_error(u2)', color='red')
plt.legend()
plt.title("Global Errors of u1 and u2")
plt.xlabel("n")
plt.ylabel("global_error")
plt.show()
```

(b)



As it can be observed, u_2 indeed experiences a constant negative change, thus for very large values of n, therefore h approximating 0, the error must converge to 0. However, as it can be observed for u_1, a true convergence is never reached due to the floating point precision limitations of the machine. In fact, the global error of u_1 is increasing because there are much more point-wise errors to be computed therefore much more machine-produced errors are introduced.