<u>Computational Fluid Dynamics</u>

Kazezova Anar Utemisovna

1. Formulation of the problem:

Find numerical solution of one-dimensional heat equation:

$$T_t = \alpha T_{xx}, \quad 0 < x < 1, \quad t > 0$$

with the following boundary conditions:

$$T(0,t) = 0, \quad T(1,t) = 0, \quad t > 0$$

and initial condition:

$$T(x,0) = 1, \quad 0 \le x \le 1$$

2. Numerical solution:

Simple Iteration Method

$$\frac{T_i^{n+1} - T_i^n}{\Delta t} = \alpha \frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{\Delta x^2}$$

$$T_i^{n+1} = T_i^n + \frac{\alpha \Delta t}{\Delta x^2}(T_{i+1}^n - 2T_i^n + T_{i-1}^n)$$

Approximation error - $O(\Delta x^2, \Delta t)$

Stability condition: von Neumann Analysis

$$\varepsilon = e^{at} e^{ikx}$$

$$\frac{e^{a(t+\Delta t)} e^{ikx} - e^{at} e^{ikx}}{\Delta t} = \alpha \left(\frac{e^{at} e^{ik(x+\Delta x)} - 2e^{at} e^{ikx} + e^{at} e^{ik(x-\Delta x)}}{\Delta x^2} \right)$$

Divide into $e^{at} e^{ikx}$ and using $2cosx = e^{ix} + e^{-ix}$ we get:

$$e^{a\Delta t} = 1 + \alpha \Delta t \left(\frac{2cosk\Delta x - 2}{\Delta x^2} \right)$$

So that the error does not grow with each step:

$$\varepsilon^{n+1} = e^{a\Delta t} \varepsilon^n \quad \rightarrow \quad \left| e^{a\Delta t} \right| \le 1$$

Therefore, the scheme is stable under the condition:

$$\left| 1 + \alpha \Delta t \left(\frac{2cosk\Delta x - 2}{\Delta x^2} \right) \right| \le 1$$

Using the trigonometric identity $2\sin^2\frac{\alpha}{2} = 1 - cos\alpha$, we rewrite the relation in the following form:

$$\left|1 - \frac{4\alpha\Delta t}{\Delta x^2}\sin^2\frac{k\Delta x}{2}\right| \le 1$$
$$-1 \le 1 - \frac{4\alpha\Delta t}{\Delta x^2}\sin^2\frac{k\Delta x}{2} \le 1$$
$$0 \le \frac{4\alpha\Delta t}{\Delta x^2}\sin^2\frac{k\Delta x}{2} \le 2$$

- $0 \le \frac{4\alpha\Delta t}{\Delta x^2}\sin^2\frac{k\Delta x}{2}$

  Always true.
- $\frac{4\alpha\Delta t}{\Delta x^2}\sin^2\frac{k\Delta x}{2} \le 2, \qquad max\sin^2\frac{k\Delta x}{2} = 1$

  $\frac{4\alpha\Delta t}{\Delta x^2} \le 2$

  $$\frac{\alpha\Delta t}{\Delta x^2} \le \frac{1}{2}$$

  Stability condition of the considered scheme - $\frac{\alpha\Delta t}{\Delta x^2} \le \frac{1}{2}$. Since this imposes a restriction on the ratio of steps in time and spatial coordinate, we cannot choose any Δx and Δt.

Stopping criterion, which means that our task has reached a stationary state:

$$\left|T_i^{n+1} - T_i^n\right| < \varepsilon$$

Program code:

```
import numpy as np
import matplotlib.pyplot as plt
```

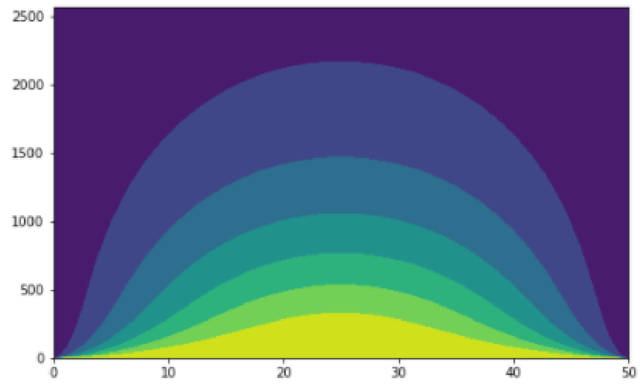
```
def f(x):
    return 1
```

```
a = 1
dx = 0.02
dt = 0.0001
x_list = np.arange(0, 1+dx, dx)
if ((a*dt)/dx**2) <= 0.5:
    n = 0
    T = np.array([np.zeros(len(x_list))])
    for i in range(len(x_list)):
        T[n][i] = f(x_list[i]) #initial condition
    T[n][0] = 0 #left boundary condition
    T[n][-1] = 0 #right boundary condition

    e = 10**6
    while e > 10**(-4):
        T = np.append(T, [np.zeros(len(x_list))], axis=0)
        for i in range(1, len(x_list)-1):
            T[n+1][i] = T[n][i] + (((a*dt)/(dx**2)) * (T[n][i+1]-2*T[n][i]+T[n][i-1]))
        T[n+1][0] = 0
        T[n+1][-1] = 0
        e = max([abs(T[n+1][i] - T[n][i]) for i in range(len(x_list))]) #stopping criterion
        n += 1
else:
    print("Not stable conditions.")
```

```
print(f'Number of iterations for simple iteration method: {n}')
fig, ax = plt.subplots()
ax.contourf(T)
fig.set_figwidth(8)
fig.set_figheight(5)
plt.show()
```

Result:

```
Number of iterations for simple iteration method: 2565
```



Implicit method

$$\frac{T_i^{n+1} - T_i^n}{\Delta t} = a\frac{T_{i+1}^{n+1} - 2T_i^{n+1} + T_{i-1}^{n+1}}{\Delta x^2}$$

Approximation error - $O(\Delta x^2, \Delta t)$

We bring this to the form of a general finite difference scheme:

$$AT_{i+1}^{n+1} + BT_i^{n+1} + CT_{i-1}^{n+1} = D_i$$

where $A = -\frac{a}{\Delta x^2}$, $B = \frac{1}{\Delta t} + \frac{2a}{\Delta x^2}$, $C = -\frac{a}{\Delta x^2}$, $D_i = \frac{T_i^n}{\Delta t}$

For numerical solution we use the Thomas Algorithm:

$$T_i = \alpha_{i+1}T_{i+1} + \beta_{i+1}$$

where $\alpha_{i+1} = -\frac{A}{B+C\alpha_i}$, $\beta_{i+1} = \frac{D_i - C\beta_i}{B+C\alpha_i}$

- To find α_1 and β_1 we use the left boundary condition $T(0,t) = 0$:
  $$T_0 = \alpha_1 T_1 + \beta_1$$
  $$0 = \alpha_1 T_1 + \beta_1$$
  It is true if and only if $\alpha_1 = 0$, $\beta_1 = 0$
- To find T_N we use the right boundary condition $T(1,t) = 0 \to T_n = 0$
- Also, from initial condition we know that $T_i^0 = 1$

Stability condition: $|B| \geq |A| + |C|$, which is always true for our approximation, so we can choose any Δx and Δt.

Stopping criterion, which means that our task has reached a stationary state:

$$\left|T_i^{n+1} - T_i^n\right| < \varepsilon$$

Program code:

```python
import numpy as np
import matplotlib.pyplot as plt
```

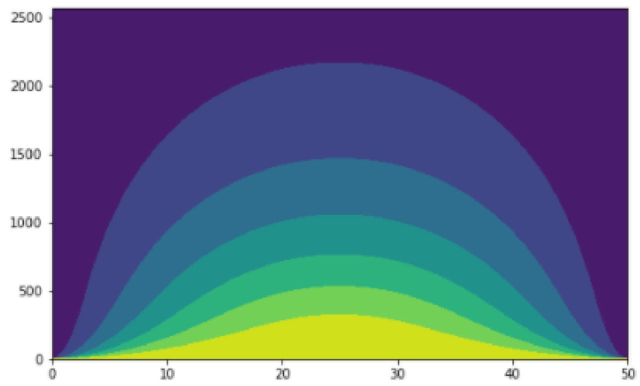
```python
def f(x):
    return 1
```

```python
a = 1
dx = 0.02
dt = 0.0001
x_list = np.arange(0, 1+dx, dx)
length = len(x_list)

A = -a/dx**2
B = (1/dt) + ((2*a)/dx**2)
C = -a/dx**2

n = 0
T = np.array([np.zeros(length)])
for i in range(length):
    T[n][i] = f(x_list[i])

e = 10**6
while e > 10**(-4):

    alpha = np.zeros(length)
    alpha[1] = 0
    beta = np.zeros(length)
    beta[1] = 0

    T = np.append(T, [np.zeros(length)], axis=0)

    for i in range(1, length-1):
        alpha[i+1] = -A/(B+(C*alpha[i]))
        beta[i+1] = (T[n][i]/dt - C*beta[i])/(B+(C*alpha[i]))

    T[n+1][-1] = 0

    for i in range(length-2, -1, -1):
        T[n+1][i] = alpha[i+1]*T[n+1][i+1] + beta[i+1]

    e = max([abs(T[n+1][i] - T[n][i]) for i in range(length)])
    n += 1
```

```python
print(f'Number of iterations for implicit method: {n}')
fig, ax = plt.subplots()
ax.contourf(T)
fig.set_figwidth(8)
fig.set_figheight(5)
plt.show()
```

Result:

```
Number of iterations for implicit method: 2566
```



Crank-Nicolson Method

$$\frac{T_i^{n+1} - T_i^n}{\Delta t} = a\left(\frac{T_{i+1}^{n+1} - 2T_i^{n+1} + T_{i-1}^{n+1}}{2\Delta x^2} + \frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{2\Delta x^2}\right)$$

Approximation error - $O(\Delta x^2, \Delta t^2)$

We bring this to the form of a general finite difference scheme:

$$AT_{i+1}^{n+1} + BT_i^{n+1} + CT_{i-1}^{n+1} = D_i$$

where $A = -\frac{a}{2\Delta x^2}$, $B = \frac{1}{\Delta t} + \frac{a}{\Delta x^2}$, $C = -\frac{a}{2\Delta x^2}$, $D_i = \frac{T_i^n}{\Delta t} + \frac{a}{2\Delta x^2}(T_{i+1}^n - 2T_i^n + T_{i-1}^n)$

For numerical solution we use the Thomas Algorithm:

$$T_i = \alpha_{i+1} T_{i+1} + \beta_{i+1}$$

where $\alpha_{i+1} = -\frac{A}{B + C\alpha_i}$, $\beta_{i+1} = \frac{D_i - C\beta_i}{B + C\alpha_i}$

- To find α_1 and β_1 we use the left boundary condition $T(0,t) = 0$:

  $$T_0 = \alpha_1 T_1 + \beta_1$$

  $$0 = \alpha_1 T_1 + \beta_1$$

  It is true if and only if $\alpha_1 = 0$, $\beta_1 = 0$

- To find T_N we use the right boundary condition $T(1,t) = 0 \rightarrow T_n = 0$

- Also, from initial condition we know that $T_i^0 = 1$

Stability condition: $|B| \geq |A| + |C|$, which is always true for our approximation, so we can choose any Δx and Δt.

Stopping criterion, which means that our task has reached a stationary state:

$$\left|T_i^{n+1} - T_i^n\right| < \varepsilon$$

Program code:

```python
import numpy as np
import matplotlib.pyplot as plt
```

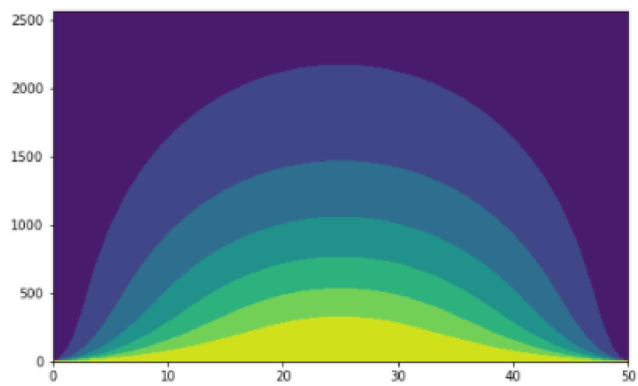
```python
def f(x):
    return 1
```

```python
a = 1
dx = 0.02
dt = 0.0001
x_list = np.arange(0, 1+dx, dx)
length = len(x_list)

A = -a/(2*dx**2)
B = (1/dt) + (a/dx**2)
C = -a/(2*dx**2)

n = 0
T = np.array([np.zeros(length)])
for i in range(length):
    T[n][i] = f(x_list[i])

e = 10**6
while e > 10**(-4):

    alpha = np.zeros(length)
    alpha[1] = 0
    beta = np.zeros(length)
    beta[1] = 0

    T = np.append(T, [np.zeros(length)], axis=0)

    for i in range(1, length-1):
        D = (T[n][i]/dt) + (a/(2*dx**2))*(T[n][i+1]-2*T[n][i]+T[n][i-1])
        alpha[i+1] = -A/(B+(C*alpha[i]))
        beta[i+1] = (D - C*beta[i])/(B+(C*alpha[i]))

    T[n+1][-1] = 0

    for i in range(length-2, -1, -1):
        T[n+1][i] = alpha[i+1]*T[n+1][i+1] + beta[i+1]

    e = max([abs(T[n+1][i] - T[n][i]) for i in range(length)])
    n += 1
```

```python
print(f'Number of iterations for Crank-Nicolson method: {n}')
fig, ax = plt.subplots()
ax.contourf(T)
fig.set_figwidth(8)
fig.set_figheight(5)
plt.show()
```

Result:

```
Number of iterations for Crank-Nicolson method: 2566
```



Combined Method B

$$(1+\theta)\frac{T_i^{n+1}-T_i^n}{\Delta t}-\theta\frac{T_i^n-T_i^{n-1}}{\Delta t}=a\left(\frac{T_{i+1}^{n+1}-2T_i^{n+1}+T_{i-1}^{n+1}}{\Delta x^2}\right)$$

Approximation error - $O(\Delta x^2, \Delta t^2)$

Let's $\theta=\frac{1}{2}$, then:

$$\left(1+\frac{1}{2}\right)\frac{T_i^{n+1}-T_i^n}{\Delta t}-\frac{1}{2}\frac{T_i^n-T_i^{n-1}}{\Delta t}=a\left(\frac{T_{i+1}^{n+1}-2T_i^{n+1}+T_{i-1}^{n+1}}{\Delta x^2}\right)$$

$$\frac{3T_i^{n+1}-4T_i^n+T_i^{n-1}}{2\Delta t}=a\left(\frac{T_{i+1}^{n+1}-2T_i^{n+1}+T_{i-1}^{n+1}}{\Delta x^2}\right)$$

We bring this to the form of a general finite difference scheme:

$$AT_{i+1}^{n+1}+BT_i^{n+1}+CT_{i-1}^{n+1}=D_i$$

where $A=-\frac{a}{\Delta x^2}$, $B=\frac{3}{2\Delta t}+\frac{2a}{\Delta x^2}$, $C=-\frac{a}{\Delta x^2}$, $D_i=\frac{2T_i^n}{\Delta t}-\frac{T_i^{n-1}}{2\Delta t}$

For numerical solution we use the Thomas Algorithm:

$$T_i=\alpha_{i+1}T_{i+1}+\beta_{i+1}$$

where $\alpha_{i+1}=-\frac{A}{B+C\alpha_i}$, $\beta_{i+1}=\frac{D_i-C\beta_i}{B+C\alpha_i}$

- To find α_1 and β_1 we use the left boundary condition $T(0,t)=0$:
  $$T_0=\alpha_1T_1+\beta_1$$
  $$0=\alpha_1T_1+\beta_1$$
  It is true if and only if $\alpha_1=0$, $\beta_1=0$
- To find T_N we use the right boundary condition $T(1,t)=0\rightarrow T_n=0$
- Also, from initial condition we know that $T_i^0=1$

Stability condition: $|B|\geq|A|+|C|$, which is always true for our approximation, so we can choose any Δx and Δt.

Stopping criterion, which means that our task has reached a stationary state:

$$\left|T_i^{n+1}-T_i^n\right|<\varepsilon$$

NOTE:

This scheme needs to have 2 time step to start. Therefore, I use the Simple Iteration Method to get the first time step.

Program code:

```python
import numpy as np
import matplotlib.pyplot as plt
```

```python
def f(x):
    return 1
```

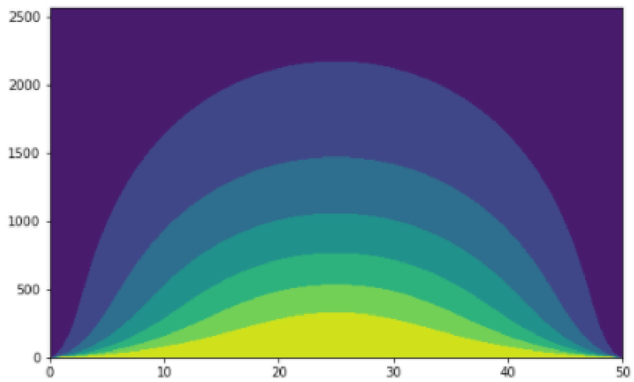
```python
a = 1
dx = 0.02
dt = 0.0001
x_list = np.arange(0, 1+dx, dx)
length = len(x_list)

n = 0
T = np.array([np.zeros(length)])
for i in range(length):
    T[n][i] = f(x_list[i])

T[n][0] = 0
T[n][-1] = 0

# This scheme needs to have 2 time step to start.
# Therefore I use the Simple Iteration Method to get the first time step.

T = np.append(T, [np.zeros(len(x_list))], axis=0)
for i in range(1, len(x_list)-1):
    T[n+1][i] = T[n][i] + (((a*dt)/(dx**2)) * (T[n][i+1]-2*T[n][i]+T[n][i-1]))
T[n+1][0] = 0
T[n+1][-1] = 0

n += 1
```

```python
# Thomas algorithm

A = -a/(dx**2)
B = (3/(2*dt)) + ((2*a)/dx**2)
C = -a/(dx**2)

e = 10**6
while e > 10**(-4):

    alpha = np.zeros(length)
    alpha[1] = 0
    beta = np.zeros(length)
    beta[1] = 0

    T = np.append(T, [np.zeros(length)], axis=0)

    for i in range(1, length-1):
        D = (2*T[n][i]/dt) - (T[n-1][i]/(2*dt))
        alpha[i+1] = -A/(B+(C*alpha[i]))
        beta[i+1] = (D - C*beta[i])/(B+(C*alpha[i]))

    T[n+1][-1] = 0

    for i in range(length-2, -1, -1):
        T[n+1][i] = alpha[i+1]*T[n+1][i+1] + beta[i+1]

    e = max([abs(T[n+1][i] - T[n][i]) for i in range(length)])
    n += 1
```

```python
print(f'Number of iterations for Combined Method B: {n}')
fig, ax = plt.subplots()
ax.contourf(T)
fig.set_figwidth(8)
fig.set_figheight(5)
plt.show()
```

Result:

```
Number of iterations for Combined Method B: 2566
```



Dufort-Frankel Method

$$\frac{T_i^{n+1} - T_i^{n-1}}{2\Delta t} = a\left(\frac{T_{i+1}^n - T_i^{n+1} - T_i^{n-1} + T_{i-1}^n}{\Delta x^2}\right)$$

$$T_i^{n+1} = \frac{a\left(\frac{T_{i+1}^n - T_i^{n-1} + T_{i-1}^n}{\Delta x^2}\right) + \frac{T_i^{n-1}}{2\Delta t}}{\frac{1}{2\Delta t} + \frac{a}{\Delta x^2}}$$

Approximation error - $O(\Delta x^2, \Delta t^2)$

Stopping criterion, which means that our task has reached a stationary state:

$$\left|T_i^{n+1} - T_i^n\right| < \varepsilon$$

NOTE:

This scheme needs to have 2 time step to start. Therefore, I use the Simple Iteration Method to get the first time step.

Program code:

```
import numpy as np
import matplotlib.pyplot as plt
```

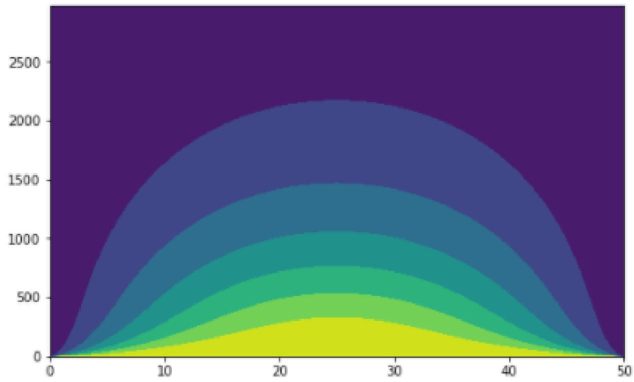
```
def f(x):
    return 1
```

```
a = 1
dx = 0.02
dt = 0.0001
x_list = np.arange(0, 1+dx, dx)
length = len(x_list)

n = 0
T = np.array([np.zeros(length)])
for i in range(length):
    T[n][i] = f(x_list[i])

T[n][0] = 0
T[n][-1] = 0

# This scheme needs to have 2 time step to start.
# Therefore I use the Simple Iteration Method to get the first time step.

T = np.append(T, [np.zeros(len(x_list))], axis=0)
for i in range(1, len(x_list)-1):
    T[n+1][i] = T[n][i] + (((a*dt)/(dx**2)) * (T[n][i+1]-2*T[n][i]+T[n][i-1]))
T[n+1][0] = 0
T[n+1][-1] = 0

n += 1

e = 10**6
while e > 10**(-4):

    T = np.append(T, [np.zeros(length)], axis=0)

    for i in range(1, len(x_list)-1):
        T[n+1][i] = ((T[n-1][i]/(2*dt)) + ((a/(dx**2)) * (T[n][i+1]-T[n-1][i]+T[n][i-1]))) / ((1/(2*dt))+(a/dx**2))

    T[n+1][0] = 0
    T[n+1][-1] = 0
    e = max([abs(T[n+1][i] - T[n][i]) for i in range(len(x_list))])
    n += 1
```

```
print(f'Number of iterations for Dufort-Frankel Method: {n}')
fig, ax = plt.subplots()
ax.contourf(T)
fig.set_figwidth(8)
fig.set_figheight(5)
plt.show()
```

Result:

```
Number of iterations for Dufort-Frankel Method: 2974
```



---

3. Conclusion:

We solved the one-dimensional heat equation with various numerical methods. They are unconditionally stable except for the simple iteration method. In each method, we find T_{ij}^{n+1} by using a loop until the stop criterion: $\left|T_{ij}^{n+1} - T_{ij}^{n}\right| < \varepsilon$ execute. It means that our solution reached the steady state.