KITAVI DUNCAN GITAU

SCT211-0031/2021

HEAT EQUATION PLOTTING

Equation used:

$$\frac{du}{dt}=\alpha\frac{d^2u}{dx^2}+e^{-t}sin(\pi x)+\frac{\pi^2}{2}e^{-t}cos(\pi x)$$

Conditions necessary for this is:

$u(0,t) = 0 $

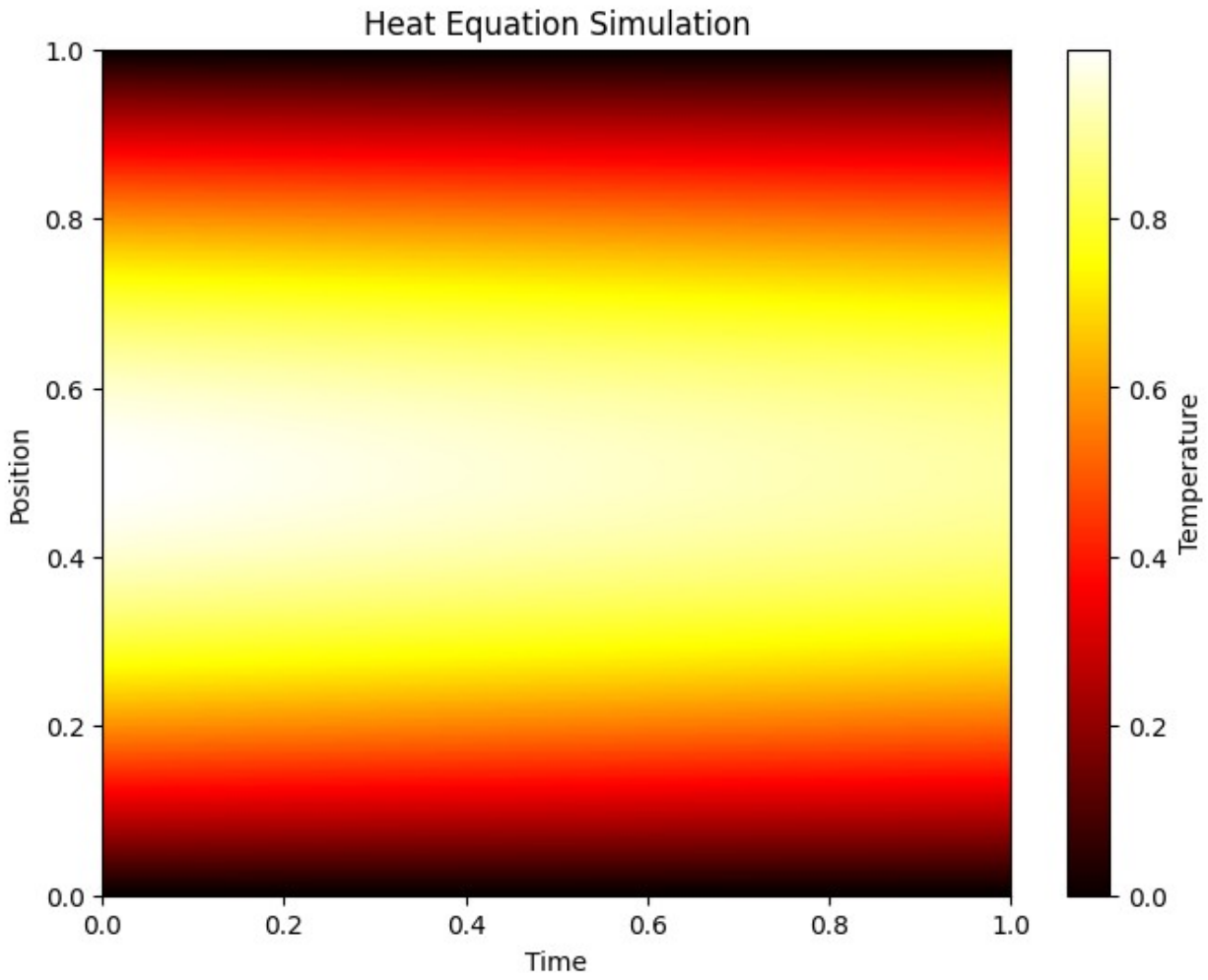
$$u(L,t)=e^{-t}+sin(\pi t)$$

The code is below

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

# Parameters
L = 1.0  # Length of the rod
T = 1.0  # Total time
Nx = 100  # Number of grid points along x
Nt = 1000  # Number of time steps
alpha = 0.01  # Thermal diffusivity

#setting up the grid
dx = L / (Nx - 1)
dt = T / Nt
x = np.linspace(0, L, Nx)
t = np.linspace(0, T, Nt)

# Source term
def source_term(x, t):
    return np.exp(-t) * np.sin(np.pi * x) + 0.5 * np.pi**2 * np.exp(-
t) * np.cos(np.pi * x)

# The conditions
left_bc = 0.0  # Temperature at left boundary

# Method used
```

```
A = diags([1, -2, 1], [-1, 0, 1], shape=(Nx-2, Nx-2)).toarray()
I = np.eye(Nx-2)
A *= alpha * dt / dx**2
B = I - 0.5 * A
C = I + 0.5 * A

#solution matrix
u = np.zeros((Nx, Nt))

#initial condition
u[:, 0] = np.sin(np.pi * x)

# Apply the conditions
u[0, :] = left_bc

# Time-stepping loop
for n in range(Nt-1):
    b = np.dot(C, u[1:-1, n])
    b[-1] -= 0.5 * alpha * dt / dx * (np.exp(-t[n+1]) + np.sin(np.pi *
t[n+1]))
    u[1:-1, n+1] = spsolve(B, b)

# Plotting
plt.figure(figsize=(8, 6))
plt.imshow(u, aspect='auto', extent=[0, T, 0, L], origin='lower',
cmap='hot')
plt.colorbar(label='Temperature')
plt.xlabel('Time')
plt.ylabel('Position')
plt.title('Heat Equation Simulation')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/scipy/sparse/linalg/_dsolve/
linsolve.py:229: SparseEfficiencyWarning: spsolve requires A be CSC or
CSR matrix format
  warn('spsolve requires A be CSC or CSR matrix format',
```



Using Dirichlet Boundary conditions

Equation used is:

$$U = -Q.log(R)$$

An equation for temperature distribution U based on the heat Source profile.

Equation

$$U = Q.e^{\frac{(X-x_0)^2+(Y-y_0)^2}{\sigma^2}}$$

U been the temperature

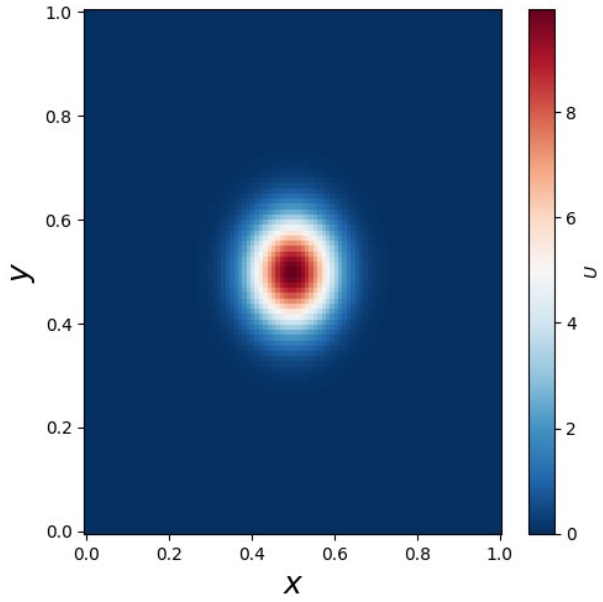
Q is the strength of the heat source

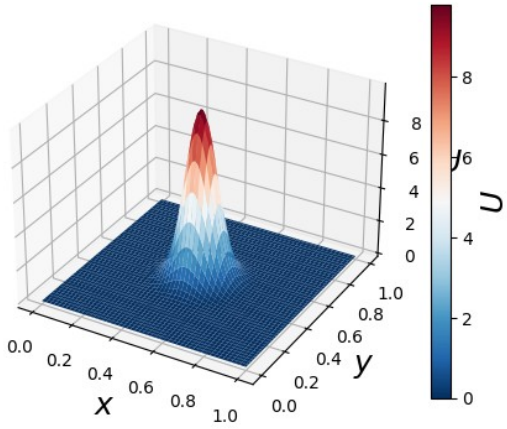
σ is std of spread of heat

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

```
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D

#parameters USED
N = 100  # Grid size
Q = 10   # Strength of the heat source
x0, y0 = 0.5, 0.5  # Position of the heat source
sigma = 0.1  # Standard deviation of the heat source profile

#grid
x = np.linspace(0, 1, N)
y = np.linspace(0, 1, N)
X, Y = np.meshgrid(x, y)

#temperature distribution with Gaussian heat source profile
U = Q * np.exp(-((X - x0)**2 + (Y - y0)**2) / sigma**2)

# Plotting
fig = plt.figure(figsize=(14, 5.5))
cmap = mpl.cm.get_cmap('RdBu_r')

# Plot using pcolor
ax = fig.add_subplot(1, 2, 1)
c = ax.pcolor(X, Y, U, cmap=cmap)
ax.set_xlabel(r"$x$", fontsize=20)
ax.set_ylabel(r"$y$", fontsize=20)
plt.colorbar(c, ax=ax, label=r"$U$")

#plot_surface used for plotting
ax = fig.add_subplot(1, 2, 2, projection='3d')
p = ax.plot_surface(X, Y, U, cmap=cmap)
ax.set_xlabel(r"$x$", fontsize=20)
ax.set_ylabel(r"$y$", fontsize=20)
ax.set_zlabel(r"$U$", fontsize=20)
cb = plt.colorbar(p, ax=ax, shrink=0.75)
cb.set_label(r"$U$", fontsize=20)

plt.show()
```

```
<ipython-input-3-2f9d8ccdd0a3>:22: MatplotlibDeprecationWarning: The
get_cmap function was deprecated in Matplotlib 3.7 and will be removed
two minor releases later. Use ``matplotlib.colormaps[name]`` or
``matplotlib.colormaps.get_cmap(obj)`` instead.
  cmap = mpl.cm.get_cmap('RdBu_r')
```





Producingwith Matplotlib's triangulation functions.

Using equation:

$$\frac{du}{dt} = D\left(\frac{d^2u}{dx^2} + \frac{d^2u}{dy^2}\right) + f(u)$$

$u(x, y, z) is the species used$

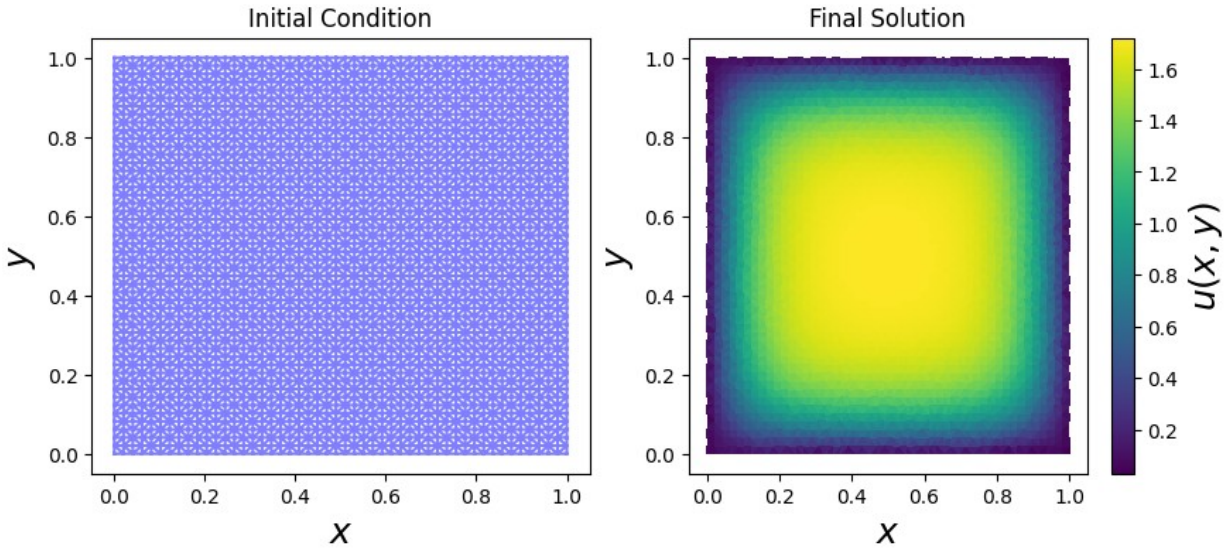
D is the diffusion coefficient,

$f(u)$ is a reaction term.

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

# Parameters
Lx = Ly = 1.5
Nx = Ny = 70
T = 20.0
Nt = 30000
D = 0.05

#setup
x = np.linspace(0, Lx, Nx)
y = np.linspace(0, Ly, Ny)
X, Y = np.meshgrid(x, y)

# Initial condition
u0 = np.random.rand(Nx, Ny) * 0.1  # Random initial concentration
```

```
# Reaction
def reaction_term(u):
    return u - u**3 / 3

#difference scheme
def reaction_diffusion_equation(u0, Nx, Ny, Nt, D, dt, dx, dy):
    u = u0.copy()
    for _ in range(Nt):
        un = u.copy()
        u[1:-1, 1:-1] = un[1:-1, 1:-1] + D * dt * (
            (un[2:, 1:-1] - 2 * un[1:-1, 1:-1] + un[:-2, 1:-1]) /
dx**2 +
            (un[1:-1, 2:] - 2 * un[1:-1, 1:-1] + un[1:-1, :-2]) /
dy**2
        ) + dt * reaction_term(un[1:-1, 1:-1])
    return u

# Solve the reaction-diffusion equation
u_final = reaction_diffusion_equation(u0, Nx, Ny, Nt, D, T/Nt, Lx/(Nx-
1), Ly/(Ny-1))

# Plotting the result using Matplotlib's triangulation functions
triang = tri.Triangulation(X.flatten(), Y.flatten())
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))

# Plotting the initial condition
ax1.triplot(triang, color='blue', alpha=0.5)
ax1.set_xlabel(r"$x$", fontsize=18)
ax1.set_ylabel(r"$y$", fontsize=18)
ax1.set_title('Initial Condition')

# Plotting the final solution
cmap = plt.cm.get_cmap('viridis')
c = ax2.tripcolor(triang, u_final.flatten(), cmap=cmap)
cb = plt.colorbar(c, ax=ax2)
cb.set_label(r"$u(x, y)$", fontsize=15)
ax2.set_xlabel(r"$x$", fontsize=15)
ax2.set_ylabel(r"$y$", fontsize=15)
ax2.set_title('Final Solution')

plt.show()
```

```
<ipython-input-15-67a9e59bcde9>:49: MatplotlibDeprecationWarning: The
get_cmap function was deprecated in Matplotlib 3.7 and will be removed
two minor releases later. Use ``matplotlib.colormaps[name]`` or
``matplotlib.colormaps.get_cmap(obj)`` instead.
  cmap = plt.cm.get_cmap('viridis')
```



Using Temperature distribution along the outer boundary of the perforated unit circle

#Parametric curve

Using equation::

$$x(t)=R\cdot(2cos(t)-cos(2t))$$

$$y(t)=R\cdot(2sin(t)-sin(2t))$$

This is done by executing the function:

$$u(x,y)=cos(x^2+y^2)$$

thus python code to bring out the curve:

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

#parametric equations for the curve
def x_parametric(t, R):
    return R * (2*np.cos(t) - np.cos(2*t))

def y_parametric(t, R):
    return R * (2*np.sin(t) - np.sin(2*t))

#the functions u(x, y) = cos(x^2 + y^2)
def u_function(x, y):
    return np.cos(x**2 + y**2)

# Define the range of the parameter t
num_points = 100
t_values = np.linspace(0, 2*np.pi, num_points)
```

```
# Define the radius
R = 1.0

# Evaluate the function u along the cardioid curve
x_values = x_parametric(t_values, R)
y_values = y_parametric(t_values, R)
u_values = u_function(x_values, y_values)

# Sort the values of u and t based on t
order = np.argsort(t_values)
t_sorted = t_values[order]
u_sorted = u_values[order]

# Plot u against t
fig, ax = plt.subplots(1, 1, figsize=(8, 4))
ax.plot(t_sorted, u_sorted, 's-', lw=2)
ax.set_ylabel(r"u(x,y) along curve", fontsize=18)
ax.set_xlabel(r"$t$", fontsize=18)
ax.set_title("u(x,y)Parametric curve")
plt.show()
```