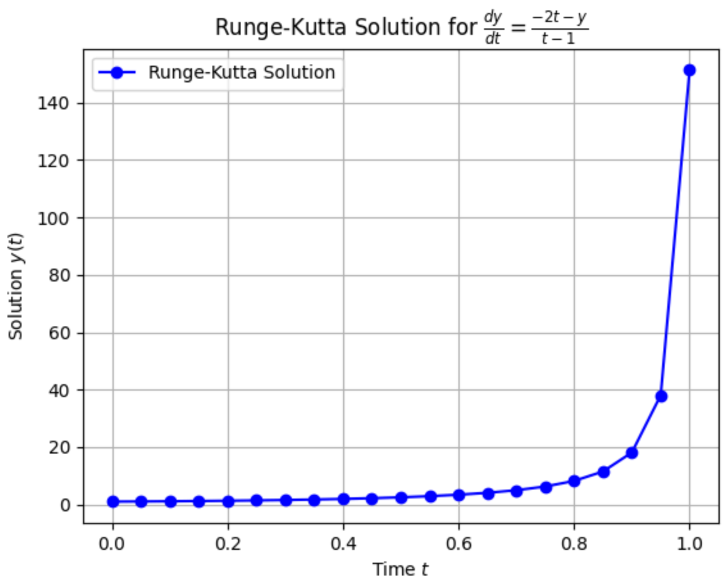
Final Math 5712

Sean Dudley

December 3, 2024

Problem 1:

Graph:



Code:

```
#PBM1: Runge-Kutta Method

import numpy as np
import matplotlib.pyplot as plt

#function
def f(t, y):  1 usage
    return (-2 * t - y) / (t - 1) if t != 1 else 0
#Runge-Kutta Method
def runge_kutta(f, y0, t0, k, T):  1 usage
    M = int((T - t0) / k)
    t = np.arange(t0, T + k, k)
    y = np.zeros(len(t))
    y[0] = y0
    #Runge-Kutta steps
    for n in range(M):
        t_n, y_n = t[n], y[n]
        k1 = f(t_n, y_n)
        k2 = f(t_n + k / 2, y_n + (k / 2) * k1)
        k3 = f(t_n + k / 2, y_n + (k / 2) * k2)
        k4 = f(t_n + k, y_n + k * k3)
        y[n + 1] = y_n + (k / 6) * (k1 + 2 * k2 + 2 * k3 + k4)
    return t, y
#Parameters
y0 = 1
t0 = 0
k = 0.05
T = 1
#Solve the ODE
t_values, y_values = runge_kutta(f, y0, t0, k, T)
```

Final Math 5712

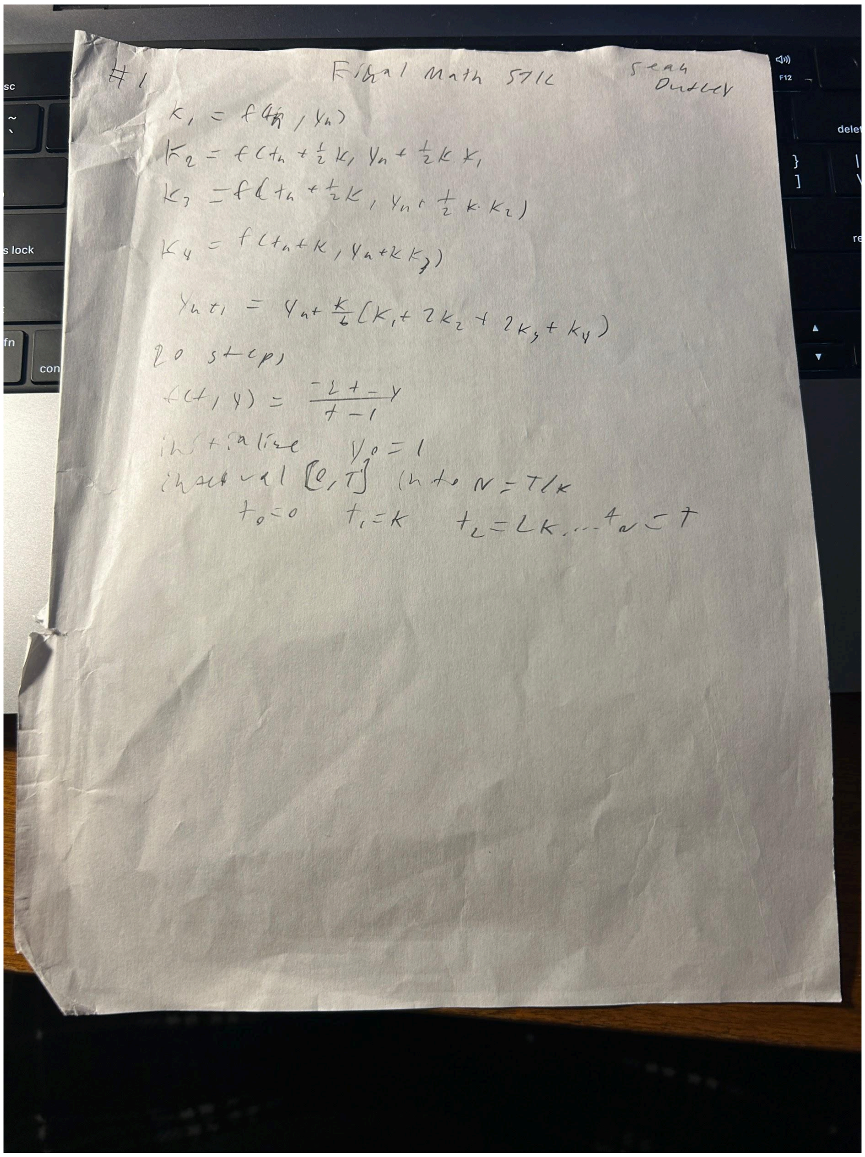
Sean Dudley

December 3, 2024

```python

#Plot
plt.plot( *args: t_values, y_values, 'b-o', label='Runge-Kutta Solution')
plt.title("Runge-Kutta Solution for $\\frac{dy}{dt} = \\frac{-2t - y}{t - 1}$")
plt.xlabel("Time $t$")
plt.ylabel("Solution $y(t)$")
plt.legend()
plt.grid()
plt.show()
```

Written steps:



#1 Final Math 5712 Sean Dudley

$k_1 = f(t_n, y_n)$

$k_2 = f(t_n + \frac{1}{2}k, y_n + \frac{1}{2}k\, k_1$

$k_3 = f(t_n + \frac{1}{2}k, y_n + \frac{1}{2}k \cdot k_2)$

$k_4 = f(t_n + k, y_n + k\, k_3)$

$y_{n+1} = y_n + \frac{k}{6}(k_1 + 2k_2 + 2k_3 + k_4)$

20 steps

$f(t, y) = \frac{-2t - y}{t - 1}$

initialize $y_0 = 1$

interval $[0, T]$ (n to $N = T/k$

$t_0 = 0 \quad t_1 = k \quad t_2 = 2k, \ldots t_N = T$

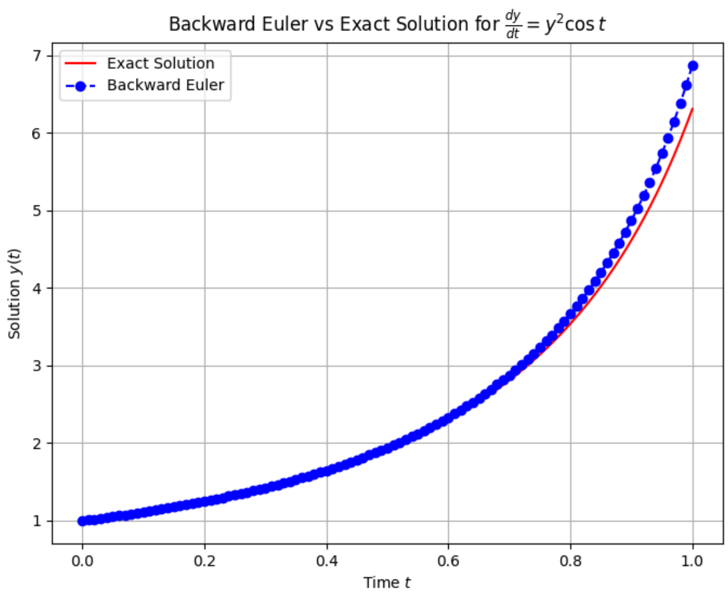
Final Math 5712

Sean Dudley

December 3, 2024

Problem 2:

Graph:



Final Math 5712

Sean Dudley

December 3, 2024

Code:

```
#PBM2: Backward Euler and Newton method
#exact solution
def exact_solution(t):  1 usage
    return -1 / (np.sin(t) - 1)
#ODE function for Backward Euler
def f(t, y):  1 usage
    return y**2 * np.cos(t)

#Newton Method
def newton_method(y_n, t_next, k, max_iter=3):  1 usage
    #Initial guess
    y_guess = y_n
    for _ in range(max_iter):
        g = y_guess - y_n - k * y_guess**2 * np.cos(t_next)
        g_prime = 1 - 2 * k * y_guess * np.cos(t_next)
        y_guess -= g / g_prime
    return y_guess
#Backward Euler Method
def backward_euler(f, y0, t0, k, T, max_iter=3):  1 usage
    #Time and solution arrays
    t_values = np.arange(t0, T + k, k)
    y_values = np.zeros(len(t_values))
    y_values[0] = y0
    #solve using Backward Euler
    for n in range(len(t_values) - 1):
        t_next = t_values[n + 1]
        y_values[n + 1] = newton_method(y_values[n], t_next, k, max_iter)
    return t_values, y_values
```

```
#Parameters
y0 = 1
t0 = 0
k = 0.01
T = 1
#Solve with Backward Euler
t_values, y_numerical = backward_euler(f, y0, t0, k, T)
#exact solution
y_exact = exact_solution(t_values)
#error at T=1
error_at_T = abs(y_exact[-1] - y_numerical[-1])

#Plot
plt.figure(figsize=(8, 6))
plt.plot( *args: t_values, y_exact, 'r-', label='Exact Solution')
plt.plot( *args: t_values, y_numerical, 'b--o', label='Backward Euler')
plt.title("Backward Euler vs Exact Solution for $\\frac{dy}{dt} = y^2 \\cos t$")
plt.xlabel("Time $t$")
plt.ylabel("Solution $y(t)$")
plt.legend()
plt.grid()
plt.show()
# error
print(f"Error at T={T}: {error_at_T:.6f}")
```

Output:

```
Error at T=1: 0.563184
Option Price 10.4043
```

Final Math 5712

Sean Dudley

December 3, 2024

Witten Steps:

#2 exact: $\frac{dy}{dt} = y^2 \cos(t)$

$$\frac{1}{y^2}\,dy = \cos(t)\,dt$$

$$\int \frac{1}{y^2}\,dy = \int \cos(t)\,dt$$

$$-\frac{1}{y} = \sin(t) + c$$

$$y(t) = \frac{-1}{\sin(t)+c} \qquad c = -1$$

$$y(t) = \frac{-1}{\sin(t)-1}$$

100 steps

Backward euler

$$\frac{y_{i+1} - y_i}{k} = y_{i+1}^2 \cos(t_{i+1})$$

$$F(y_{i+1}) = y_{i+1} - k y_{i+1}^2 \cos(t_{i+1}) - y_i = 0$$

Newtons method! $F(y_{i+1}) = y_{i+1} - k^2 y_{i+1} \cos(t_{i+1})$

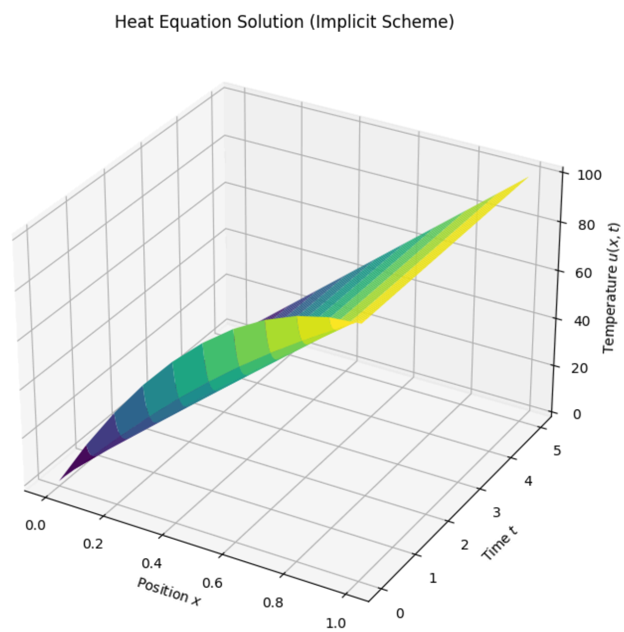
derrivative $F'(y_{i+1}) = 1 - 2k y_{i+1} \cos(t_{i+1})$

$$y_{i+1}^{(k+1)} = y_{i+1}^{k} - \frac{F(y_{i+1}^{k})}{F'(y_{i+1}^{k})}$$

Final Math 5712
Sean Dudley
December 3, 2024

Problem 3:

Graph:



Code:

Final Math 5712

Sean Dudley

December 3, 2024

```python
#PBM 3 Implicit scheme for Heat equation

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Parameters
L = 1
T = 5
h = 1 / 10
k = h / 2
s = h**2 / k
tau = 2 + s
r = 1
# Discretization
x = np.arange(0, L + h, h)
t = np.arange(0, T + k, k)
N_x = len(x)
N_t = len(t)
# Initial and boundary conditions
u = np.zeros((N_t, N_x))
u[0, :] = 100 * x * (2 - x)
u[:, 0] = 0
u[:, -1] = 100
# Tridiagonal matrix A
main_diag = tau * np.ones(N_x - 2)
lower_diag = -r * np.ones(N_x - 3)
upper_diag = -r * np.ones(N_x - 3)
```

```python
# Thomas Algorithm
def thomas_algorithm(a, b, c, d):  1 usage
    n = len(b)
    c_prime = np.zeros(n - 1)
    d_prime = np.zeros(n)
    c_prime[0] = c[0] / b[0]
    d_prime[0] = d[0] / b[0]
    for i in range(1, n - 1):
        temp = b[i] - a[i - 1] * c_prime[i - 1]
        c_prime[i] = c[i] / temp
        d_prime[i] = (d[i] - a[i - 1] * d_prime[i - 1]) / temp
    d_prime[-1] = (d[-1] - a[-2] * d_prime[-2]) / (b[-1] - a[-2] * c_prime[-2])
    x = np.zeros(n)
    x[-1] = d_prime[-1]
    for i in range(n - 2, -1, -1):
        x[i] = d_prime[i] - c_prime[i] * x[i + 1]
    return x
# Time-stepping loop
for n in range(N_t - 1):
    # Right-hand side
    b = s * u[n, 1:-1]
    b[0] += r * 0
    b[-1] += r * 100
# Solve the system for the next time step
    u[n + 1, 1:-1] = thomas_algorithm(lower_diag, main_diag, upper_diag, b)
```

Final Math 5712

Sean Dudley

December 3, 2024

```python
# 3D Plot
X, T_mesh = np.meshgrid( *xi: x, t)
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, T_mesh, u, cmap='viridis')
ax.set_title("Heat Equation Solution (Implicit Scheme)")
ax.set_xlabel("Position $x$")
ax.set_ylabel("Time $t$")
ax.set_zlabel("Temperature $u(x, t)$")
plt.show()
```

Written steps:

Final Math 5712

Sean Dudley

December 3, 2024

#3 Final Math 5712 Sean Dudley

$$\frac{u_i^{n+1} - u_i^n}{k} = \frac{1}{10} \, \frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i+1}^{n+1}}{h^2}$$

$$r = \frac{k}{10h^2} \Rightarrow -r u_{i-1}^{n+1} + (1 + 2r) u_i^{n+1} - r u_{i+1}^{n+1} = u_i^n$$

tridiagonal

$$A = \begin{bmatrix} 1+2r & -r & 0 & \cdots & \cdots & 0 \\ -r & 1+2r & 0 & \cdots & & \\ 0 & -r & 1+2r & \ddots & & \\ \vdots & & \ddots & \ddots & \ddots & \\ & & & \ddots & \ddots & -r \\ & & 0 & \cdots & -r & 1+2r \end{bmatrix}$$

discretize domain

rearranging

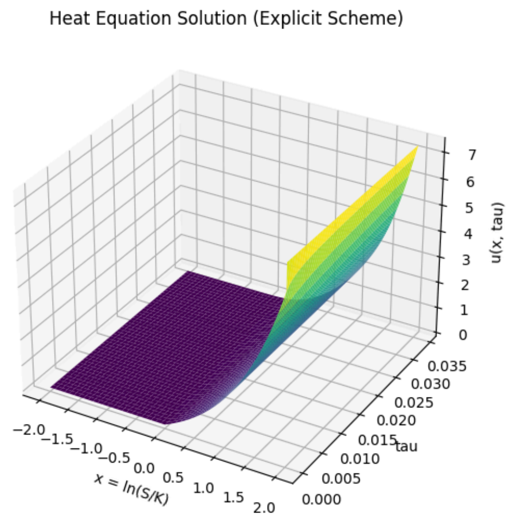
Final Math 5712

Sean Dudley

December 3, 2024

Problem 4:

Graph:



Code:

Final Math 5712

Sean Dudley

December 3, 2024

```python
# PBM 4

#Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
#Parameters
S = 85
K = 85
r = 0.04
sigma = 0.5
T = 100 / 365
h = 0.004
x_min = -2
x_max = 2
k = (sigma**2 * T) / (2 * 10000)
tau_max = sigma**2 * T / 2
#discretization
x = np.arange(x_min, x_max + h, h)
tau = np.arange(0, tau_max + k, k)
N_x = len(x)
N_tau = len(tau)
sigma_explicit = k / h**2
#sol array
u = np.zeros((N_tau, N_x))
#initial/boundary condition
u[0, :] = np.maximum(np.exp(x) - 1, 0)
u[:, 0] = 0
u[:, -1] = np.exp(x_max)
```

Final Math 5712

Sean Dudley

December 3, 2024

```python
#Explicit scheme
for n in range(N_tau - 1):
    for i in range(1, N_x - 1):
        u[n + 1, i] = (
            sigma_explicit * u[n, i - 1] +
            (1 - 2 * sigma_explicit) * u[n, i] +
            sigma_explicit * u[n, i + 1] )
#compute
k_transformed = 2 * r / sigma**2
tau_final = tau_max
ln_S_K = np.log(S / K)
index = np.argmin(np.abs(x - ln_S_K))
u_interpolated = u[-1, index]

#Option price
C = K * u_interpolated * np.exp(
    -0.5 * (k_transformed - 1) * ln_S_K -
    0.25 * (k_transformed + 1)**2 * tau_final)
#Price
print(f"The price call option is {C:.4f}")
#Plot
X, T_mesh = np.meshgrid( *xi: x, tau)
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, T_mesh, u, cmap='viridis')
ax.set_title("Heat Equation Solution (Explicit Scheme)")
ax.set_xlabel("x = ln(S/K)")
ax.set_ylabel("tau")
ax.set_zlabel("u(x, tau)")
plt.show()
```

Output of call price:

```
Error at T=1: 0.563184
Option Price 10.4043
```

Written steps:

Final Math 5712

Sean Dudley

December 3, 2024

#4 Final Math 5712 Sean Dudley

explicit $u_i^{n+1} = \sigma u_{i-1}^n + (1-2\sigma)u_i^n + \sigma u_{i+1}^n$

$\sigma = \frac{k}{h^2}$

$u(x,0) = \max(e^x - 1, 0)$

2) $x \to -\infty$ $u(x,t) = 0$ $x \to \infty$ $u(x,t) = e^x$

Discretization

$$N_x = \frac{x_{max} - x_{min}}{h} + 1$$

time step $k = \frac{\sigma^2 t}{2 \cdot 10.000}$

total steps $N_t = \frac{\sigma^2 T}{2k}$ temporal

$$N_x = \frac{x_{max} - x_{min}}{h} + 1$$

u_i^{n+1} for interior points

$$u_i^{n+1} = \sigma u_{i-1}^n + (1-2\sigma)u_i^n + \sigma u_{i+1}^n$$

$$C(S,0) = K u(\ln(S/K), \tau) e^{-\frac{1}{2}(k-1)\ln(S/K) - \frac{1}{4}(k+1)^2 \frac{\sigma^2 T}{2}}$$

$\tau_n = n \cdot k$ $n = 0, 1, \dots, N_\tau$

$u_0^{n+1} = 0$

$u_{N_x}^{n+1} = e^{x_{max}}$