Numerical Analysis Using Finite Difference Method

Your Name

1 Introduction

Numerical analysis is a branch of mathematics that deals with the development and implementation of algorithms for solving mathematical problems. One common approach to numerical analysis is the finite difference method, which involves approximating derivatives of a function by evaluating it at a set of discrete points.

The finite difference method can be used to solve a wide range of problems, including differential equations, optimization problems, and interpolation. In this article, we will focus on its application to the solution of differential equations.

2 Finite Difference Method

Consider the following second-order differential equation:

$$y''(x) = f(x) \tag{1}$$

where $y(x)$ is the unknown function, and $f(x)$ is a given function. To solve this equation using the finite difference method, we first need to discretize the domain of x into a set of N equally spaced points, with a spacing of h. Let $x_i = ih$, for $i = 0, 1, \ldots, N$.

We can then approximate the second derivative of $y(x)$ at x_i using a finite difference formula:

$$y''(x_i) \approx \frac{y(x_{i+1}) - 2y(x_i) + y(x_{i-1})}{h^2} \tag{2}$$

Using this approximation, we can rewrite the differential equation (1) as a system of linear equations:

$$\frac{1}{h^2}\begin{bmatrix}1 & 0 & 0 & \dots & 0\\ 1 & -2 & 1 & \dots & 0\\ 0 & 1 & -2 & \dots & 0\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \dots & 1 & -2\\ 0 & 0 & \dots & 0 & 1\end{bmatrix}\begin{bmatrix}y_0\\ y_1\\ y_2\\ \vdots\\ y_{N-1}\\ y_N\end{bmatrix}=\begin{bmatrix}f(x_0)\\ f(x_1)\\ f(x_2)\\ \vdots\\ f(x_{N-1})\\ f(x_N)\end{bmatrix} \tag{3}$$

where $y_i \approx y(x_i)$.

Solving this system of linear equations gives us an approximation to the solution of the differential equation (1) at the discrete points x_i.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

# Define functions
def diff2_central(f, x, h=0.1):
    return (f(x+h)-2*f(x)+f(x-h))/h**2

def f(xx):
    C = (11-81/12)/3
    return xx**4/12 - xx**2 + C*xx

def fn(xx):
    return xx**2 - 2

# Set up the graph
fig, ax = plt.subplots()
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_xlim(0, 3)
ax.set_ylim(-1, 1)

# Plot the initial function f(x)
x = np.arange(0, 3, 0.01)
y = f(x)
ax.plot(x, y, color='red')

# Define variables and initial conditions
z = 3
x = 0
dx = 0.01
h = 0.2
xp = np.arange(0, 3+h, h)
m = f(z)/z
```

```
fp = m*xp
fp2 = m*xp

# Plot the initial function fp(x)
line, = ax.plot(xp, fp, color='blue', marker='o')

# Create an array of colors
colors = ['orange', 'green', 'purple', 'brown', 'teal', 'maroon', 'navy']

# Define update function for animation
def update(n):
    global fp, fp2
    fdata = []
    for i in range(1, len(xp)-1):
        fp2[i] = 0.5*(fp[i+1]+fp[i-1]-h**2*(fn(xp[i])))

    for i in range(len(xp)):
        fp[i] = fp2[i]
        fdata.append([xp[i], fp[i]])

    # Update the line data with the new values
    line.set_data(xp, fp)

    # Set line color based on the color array and current iteration
    color_index = n // 50 % len(colors)
    line.set_color(colors[color_index])

    return [line]

# Create animation object
anim = FuncAnimation(fig, update, frames=range(N), interval=50)

plt.show()
```

3 Conclusion

The finite difference method is a powerful technique for solving a wide range of problems in numerical analysis. In this article, we have seen how it can be applied to the solution of differential equations. By discretizing the domain of a function into a set of discrete points, we can use finite difference formulas to approximate derivatives and rewrite differential equations as systems of linear equations. Solving these systems of linear equations gives us an approximation to the solution of the original problem.