Project Report

MATH 205 Numerical Analysis

Kapetanović Kerim 220302143

Kruho Vedad 220302139

Ibrulj Tarik 220302142

Professor: dr. **SeyedNima Rabiei**

Sarajevo, Spring 2024.

Table of Contents

Table of Figures

Project 3 (It is worth 10 points)

Write a user-defined Julia function that solves a second-order boundary value problem of the form:

$$y^{(2)} + a_1(x)y^{(1)} + a_0(x)y = f(x), \; x \in [a, b], \; y(a) = \alpha, \; y(b) = \beta.$$

Using the two point central difference formula and three point central difference formula for approximating the first and second derivatives, respectively.

Figure 1. Project Problem

Full Code

```
using LinearAlgebra
using Plots

function LU_decomposition(A, b)
    n = size(A, 1) # calculation of square matrix
    L = zeros(n, n) # initialization of empty matrix, will store LT part of LU decomposition
    U = copy(A) # UT part of LU decomposition
    P = Matrix(I, n, n) # permutation matrix to keep track of row interchanges and for better stability

    for k in 1:n-1
        val, index = findmax(abs.(U[k:n, k])) # finds the index of MAX absolute value in given submatrix; used for partial pivoting
        index += k - 1 # we need to make it relative to original matrix, not submatrix

        if val == 0 # if MAX absolute value is zero
            error("Division by 0!")
        end

        # partial pivoting, element with the largest absolute value is placed at diagonal for stability; swaping rows 'k' and 'index' between 'U', 'L' and 'P'
        U[[k, index], :] = U[[index, k], :]
        L[[k, index], :] = L[[index, k], :]
        P[[k, index], :] = P[[index, k], :]
```

```
        for i in k+1:n
            L[i, k] = U[i, k] / U[k, k] # calculating to eliminate entries below
diagonal
            for j in k+1:n
                U[i, j] -= L[i, k] * U[k, j] # eliminates the entries below
diagonal in column 'k' of 'U'
            end
        end
    end

    for i in 1:n
        L[i, i] = 1.0 # set diagonal entries to 1
    end

    # apply permutation to b
    b = P * b
    n = length(b) # length of vector b
    y = zeros(n)
    x = zeros(n)

    # forward substitution: Ly = b
    for i in 1:n # itterating over each eq in system
        y[i] = b[i] # from each eq y, to correspond b
        for j in 1:i-1
            y[i] -= L[i, j] * y[j] # updating 'y[i]', subtracting product of
correspond element from 'L' and previously calculated value 'y[j]'
        end
        y[i] = y[i] / L[i, i]
    end

    # backward substitution: Ux = y
    for i in n:-1:1 # in reverse order from last eq
        x[i] = y[i]
        for j in i+1:n # iterates over elements of 'x'
            x[i] -= U[i, j] * x[j] # updaitng 'x[i]', similarly like in forward
substitution
        end
        x[i] = x[i] / U[i, i] # dividing by diagonal of 'U' at '[i,i]'
    end

    return x
end

function differential_eq(a, b, c, f, a0, b0, alpha, beta, N)
    # Define step size
```

```julia
    h = (b0 - a0) / N

    # Define grid points
    x = range(a0, stop=b0, length=N+1)

    # Initialize matrix and vector for finite difference method, Ax=b
    A_Mat = zeros(N+1, N+1)
    B_Vec = zeros(N+1)

    # Fill matrix and vector
    for i in 2:N
        # Evaluate coefficients at each grid point
        a_i = a(x[i])
        b_i = b(x[i])
        c_i = c(x[i])

        # Fill the matrix A and vector B using central difference approximations
        A_Mat[i, i-1] = 1 / h^2 - b_i / (2 * h)
        A_Mat[i, i] = -2 / h^2 + c_i
        A_Mat[i, i+1] = 1 / h^2 + b_i / (2 * h)
        B_Vec[i] = f(x[i])
    end

    # Apply boundary conditions
    A_Mat[1, 1] = 1.0
    A_Mat[N+1, N+1] = 1.0
    B_Vec[1] = alpha
    B_Vec[N+1] = beta

    # Solve the linear system using LU decomposition
    y = LU_decomposition(A_Mat, B_Vec)

    return x, y
end

a(x) = 1.0
b(x) = 0.5
c(x) = 3.0
f(x) = cos(2*x)
a0 = 0.0
b0 = 4.0
alpha = 4.0
beta = 2.1
N = 100
```

```
x, y = differential_eq(a, b, c, f, a0, b0, alpha, beta, N)

# Print the solution
println("Solution for the differential equation:")
for i in 1:length(x)
    println("x = ", x[i], ", y = ", y[i])
end

# Plot the solution
plot(x, y, label="Numerical Solution", xlabel="x", ylabel="y", title="Numerical
Solution of Second Order BVP")
savefig("05_05.png")
```

Figure 2. Full Code

```
a(x) = 1.0
b(x) = 0.5
c(x) = 3.0
f(x) = cos(2*x)
a0 = 0.0
b0 = 4.0
alpha = 4.0
beta = 2.1
N = 100

x, y = differential_eq(a, b, c, f, a0, b0, alpha, beta, N)
```

Figure 3. Initial Values

```
function differential_eq(a, b, c, f, a0, b0, alpha, beta, N)
    # Define step size
    h = (b0 - a0) / N

    # Define grid points
    x = range(a0, stop=b0, length=N+1)

    # Initialize matrix and vector for finite difference method, Ax=b
    A_Mat = zeros(N+1, N+1)
    B_Vec = zeros(N+1)

    # Fill matrix and vector
    for i in 2:N
        # Evaluate coefficients at each grid point
        a_i = a(x[i])
        b_i = b(x[i])
        c_i = c(x[i])

        # Fill the matrix A and vector B using central difference approximations
        A_Mat[i, i-1] = 1 / h^2 - b_i / (2 * h)
        A_Mat[i, i] = -2 / h^2 + c_i
        A_Mat[i, i+1] = 1 / h^2 + b_i / (2 * h)
        B_Vec[i] = f(x[i])
    end

    # Apply boundary conditions
    A_Mat[1, 1] = 1.0
    A_Mat[N+1, N+1] = 1.0
    B_Vec[1] = alpha
    B_Vec[N+1] = beta

    # Solve the linear system using LU decomposition
    y = LU_decomposition(A_Mat, B_Vec)

    return x, y
end
```

Figure 4. Function differential_eq

Function differential_eq: Defines the system of nonlinear equations.

```
function LU_decomposition(A, b)
    n = size(A, 1) # calculation of square matrix
    L = zeros(n, n) # initialization of empty matrix, will store LT part of LU
decomposition
    U = copy(A) # UT part of LU decomposition
    P = Matrix(I, n, n) # permutation matrix to keep track of row interchanges
and for better stability

    for k in 1:n-1
        val, index = findmax(abs.(U[k:n, k])) # finds the index of MAX absolute
value in given submatrix; used for partial pivoting
        index += k - 1 # we need to make it relative to original matrix, not
submatrix

        if val == 0 # if MAX absolute value is zero
            error("Division by 0!")
        end

        # partial pivoting, element with the largest absolute value is placed at
diagonal for stability; swaping rows 'k' and 'index' between 'U', 'L' and 'P'
        U[[k, index], :] = U[[index, k], :]
        L[[k, index], :] = L[[index, k], :]
        P[[k, index], :] = P[[index, k], :]

        for i in k+1:n
            L[i, k] = U[i, k] / U[k, k] # calculating to eliminate entries below
diagonal
            for j in k+1:n
                U[i, j] -= L[i, k] * U[k, j] # eliminates the entries below
diagonal in column 'k' of 'U'
            end
        end
    end
```

Figure 5. LU Decomposition Function

Function LU_decomposition(A, b): Performes the LU decomposition. It decomposes the given matrix A into an upper triangular matrix U and lower triangular matrix L, ***A = LU.*** And after that solves the LU decomposition using forward and backward substitution.

```
# Plot the solution
plot(x, y, label="Numerical Solution", xlabel="x", ylabel="y", title="Numerical Solution of Second Order BVP")
savefig("05_05.png")
```

Figure 6. Plotting

Plotting: The result is plotted on the graph.

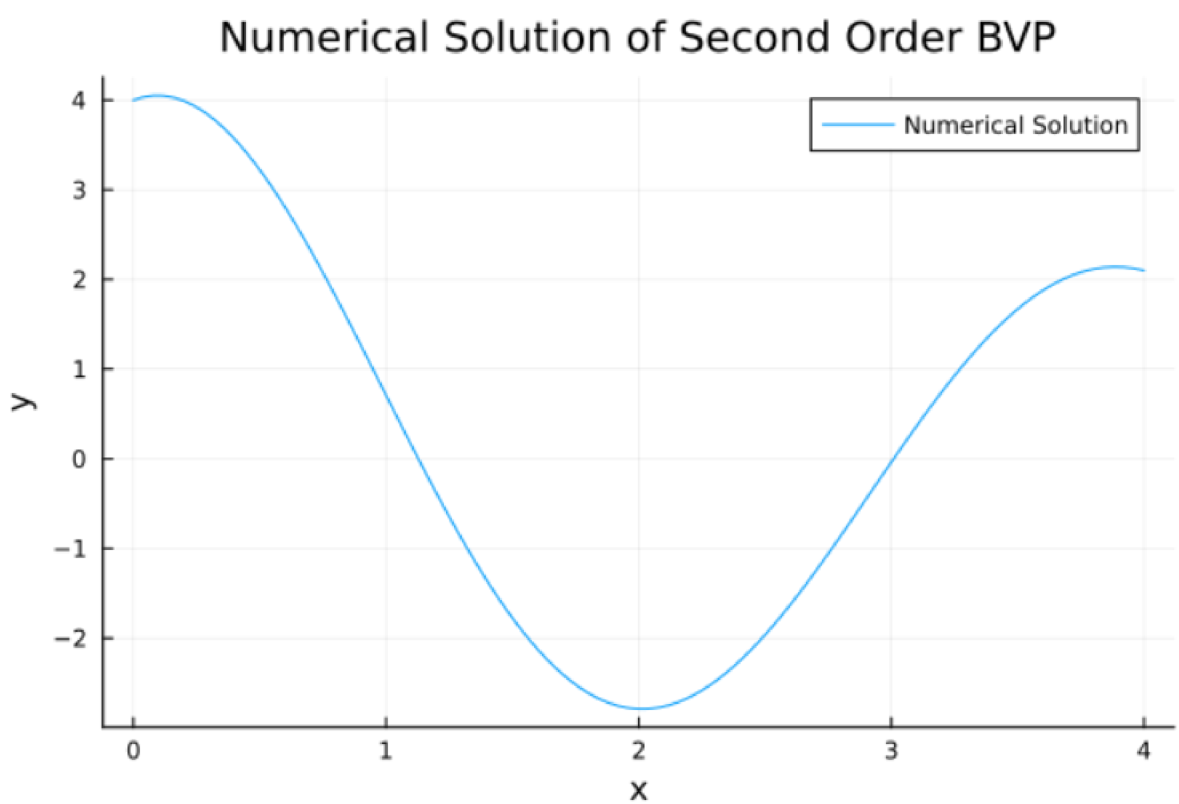

Figure 7. Graph of the Second Order Differential Equation

Results: Figure 7 shows the graph of the second order differential equation

One more example

```
a(x) = 1.0
b(x) = 0.0
c(x) = x
f(x) = sin(2*x)
a0 = 1.27
b0 = 10.0
alpha = -0.84
beta = -0.70
N = 100
```