Sayısal Çözüm Yöntemleri Dersi

Son Ödev

Alirıza Bilir 18014125

Eğri Uydurma Fonksiyonları:

Vandermonde Yöntemi

lineer.m | sor.m | gauss_seidel.m | jacobi.m | vandermondeTechnique.m

```
function coefficients = vandermondeTechnique(x, y)
    n = length(x);

    if n ~= length(y)
        error('x ve y dizileri aynı uzunlukta olmalıdır.');
    end

    % Vandermonde matrisi oluşturma
    A = zeros(n, n);
    for i = 1:n
        A(:, i) = x.^(i-1);
    end

    % Katsayıları bulma
    coefficients = A \ y'; % Katsayıları bulmak için matris bölme işlemi

    % Sonuçları ekrana yazdırma
    disp('Polynomial coefficients:');
    disp(coefficients');
end
```

Lagrange Yöntemi

lagrangeInterpolation.m

```
function result = lagrangeInterpolation(x, y, targetX)
    n = length(x);

    if n ~= length(y)
        error('x ve y dizileri aynı uzunlukta olmalıdır.');
    end

    result = 0;
    for i = 1:n
        term = y(i);
        for j = 1:n
            if j ~= i
                term = term * (targetX - x(j)) / (x(i) - x(j));
            end
        end
        result = result + term;
    end

    disp(['Interpolated value at x = ' num2str(targetX) ' is ' num2str(result)]);
end
```

Newton İnterpolasyon Yöntemi

numericalMethodsMenu.m | lineer.m | sor.m | gauss_seidel.m | jacobi.m | vandermondeTechnique.m | newtonInterpolation.m | secantMethod

```m
function result = newtonInterpolation(x, y, targetX)
    n = length(x);

    if n ~= length(y)
        error('x ve y dizileri aynı uzunlukta olmalıdır.');
    end

    % İleri farkları tablosu oluşturma
    forwardDiffTable = zeros(n);
    forwardDiffTable(:, 1) = y';
    for j = 2:n
        for i = 1:n-j+1
            forwardDiffTable(i, j) = (forwardDiffTable(i+1, j-1) - forwardDiffTable(i, j-1)) / (x(i+j-1) - x(i));
        end
    end

    % Katsayıları ve interpolasyonu hesaplama
    coefficients = forwardDiffTable(1, :);
    result = coefficients(1);
    temp = 1;
    for i = 2:n
        temp = temp * (targetX - x(i-1));
        result = result + coefficients(i) * temp;
    end

    disp(['Interpolated value at x = ' num2str(targetX) ' is ' num2str(result)]);
end
```

Lineer Denklem Sistemi Çözüm Yöntemleri:

Jacobi Yöntemi

numericalMethodsMenu.m | lineer.m | sor.m | gauss_seidel.m | jacobi.m | vandermondeTech

```m
function x = jacobi(A, b, x0, tol, max_iter)
    n = length(b);
    x = x0;
    iteration = 0;
    error_history = [];

    while iteration < max_iter
        x_old = x;

        for i = 1:n
            sigma = A(i, :) * x_old - A(i, i) * x_old(i);
            x(i) = (b(i) - sigma) / A(i, i);
        end

        iteration = iteration + 1;
        error = norm(x - x_old, inf);
        error_history = [error_history; error];

        if error < tol
            break;
        end
    end

    fprintf('Jacobi method converged in %d iterations.\n', iteration);
    fprintf('Solution: ');
    disp(x);

    % Plot error vs iteration
    figure;
    semilogy(error_history);
    xlabel('Iteration');
    ylabel('Error');
    title('Error vs Iteration (Jacobi)');
end
```

Gauss Yöntemi

numericalMethodsMenu.m | lineer.m | sor.m | gauss_seidel.m | jacobi.m | vandermondeTechnique

```m
function x = gauss_seidel(A, b, x0, tol, max_iter)
    n = length(b);
    x = x0;
    iteration = 0;
    error_history = [];

    while iteration < max_iter
        x_old = x;

        for i = 1:n
            sigma = A(i, 1:i-1) * x(1:i-1) + A(i, i+1:end) * x_old(i+1:end);
            x(i) = (b(i) - sigma) / A(i, i);
        end

        iteration = iteration + 1;
        error = norm(x - x_old, inf);
        error_history = [error_history; error];

        if error < tol
            break;
        end
    end

    fprintf('Gauss-Seidel method converged in %d iterations.\n', iteration);
    fprintf('Solution: ');
    disp(x);

    % Plot error vs iteration
    figure;
    semilogy(error_history);
    xlabel('Iteration');
    ylabel('Error');
    title('Error vs Iteration (Gauss-Seidel)');
end
```

SOR Yöntemi

numericalMethodsMenu.m | lineer.m | sor.m | gauss_seidel.m | jacobi.m | vandermondeTechnique.m

```m
function x = sor(A, b, x0, omega, tol, max_iter)
    n = length(b);
    x = x0;
    iteration = 0;
    error_history = [];

    while iteration < max_iter
        x_old = x;

        for i = 1:n
            sigma = A(i, :) * x - A(i, i) * x(i);
            x(i) = (1 - omega) * x_old(i) + (omega / A(i, i)) * (b(i) - sigma);
        end

        iteration = iteration + 1;
        error = norm(x - x_old, inf);
        error_history = [error_history; error];

        if error < tol
            break;
        end
    end

    fprintf('SOR method converged in %d iterations.\n', iteration);
    fprintf('Solution: ');
    disp(x);

    % Plot error vs iteration
    figure;
    semilogy(error_history);
    xlabel('Iteration');
    ylabel('Error');
    title('Error vs Iteration (SOR)');
end
```

Nonlineer Denklem Sistemi Çözüm Yöntemleri:

Sabit Nokta İterasyon Yöntemi

```
function [root, iterations] = fixedPointIteration(g, x0, tol, maxIter)
    iterations = 0;
    while iterations < maxIter
        x = g(x0);
        if abs(x - x0) < tol
            root = x;
            return;
        end
        x0 = x;
        iterations = iterations + 1;
    end
    root = NaN;
end
```

Newton Yönteimi

```
function [root, iterations] = newtonMethod(func, derivative, x0, tol, maxIter)
    iterations = 0;
    while iterations < maxIter
        fx = func(x0);
        if abs(fx) < tol
            root = x0;
            return;
        end

        fprime_x = derivative(x0);
        if fprime_x == 0
            error('Derivative is zero. Division by zero error.');
        end

        x = x0 - fx / fprime_x;
        if abs(x - x0) < tol
            root = x;
            return;
        end

        x0 = x;
        iterations = iterations + 1;
    end
    root = NaN;
end
```

Nonlineer Denklem Çözüm Yötemleri :

Secant Yöntemi

```
function [root, iterations] = secantMethod(func, x0, x1, tol, maxIter)
    iterations = 0;
    while iterations < maxIter
        fx0 = func(x0);
        fx1 = func(x1);
        x = x1 - (fx1 * (x1 - x0)) / (fx1 - fx0);
        if abs(x - x1) < tol
            root = x;
            return;
        end
        x0 = x1;
        x1 = x;
        iterations = iterations + 1;
    end
    root = NaN;
end
```

Bisection Yöntemi

bisectionMethod.m

```m
function root = bisectionMethod(func, a, b, tol, maxIter)
    if func(a) * func(b) > 0
        error('f(a) ve f(b) işaretleri farklı olmalıdır.');
    end

    iterations = 0;
    while iterations < maxIter
        c = (a + b) / 2;
        if func(c) == 0 || (b - a) / 2 < tol
            root = c;
            return;
        end
        iterations = iterations + 1;

        if func(c) * func(a) < 0
            b = c;
        else
            a = c;
        end
    end
    root = NaN;
end
```

Menü

numericalMethodsMenu.m | lineer.m | sor.m | gauss_seidel.m | jacobi.m

```m

function numericalMethodsMenu()
    while true
        disp('Numerical Methods Menu:');
        disp('1. Nonlinear Equation Solution');
        disp('2. Nonlinear Equation System Solution');
        disp('3. Linear Equation System Solution');
        disp('4. Curve fitting');
        disp('5. Exit');
        choice = input('Enter your choice: ');
        switch choice
            case 1
                nonlinearEquationSolutionMenu();
            case 2
                nonlinearEquationSystemSolutionMenu();
            case 3
                linearEquationSystemSolutionMenu();
            case 4
                curveFittingMenu();
            case 5
                disp('Programdan Çıkılıyor');
                break;
            otherwise
                disp('Geçersiz Seçim');
        end
    end
end
function nonlinearEquationSystemSolutionMenu()
    disp('Nonlinear Equation Solution Menu:');
    disp('1. Fixed Point Iteration Method');
    disp('2. Newton Method');
    methodChoice = input('Enter your choice: ');
    switch methodChoice
         case 1
            disp('Fixed Point Iteration Method');
            g = input('Enter the function g(x): ');
            x0 = input('Enter initial guess x0: ');
            tolerance = input('Enter tolerance: ');
            maxIterations = input('Enter maximum number of iterations: ');
            fixedPointIteration(g, x0, tolerance, maxIterations);
        case 2
            disp('Newton Method');
            func = input('Enter the function f(x): ');
            derivative = input('Enter the derivative of f(x): ');
            x0 = input('Enter initial guess x0: ');
            tolerance = input('Enter tolerance: ');
            maxIterations = input('Enter maximum number of iterations: ');
            newtonMethod(func, derivative, x0, tolerance, maxIterations);
        otherwise
            disp('Geçersiz Seçim');
    end
end
```

```
end

function nonlinearEquationSolutionMenu()
    disp('Nonlinear Equation  Solution Menu:');
    disp('1. Bisection Method');
    disp('1. Secant Method');
    methodChoice = input('Enter your choice: ');
    switch methodChoice
        case 1
            disp('Bisection Method');
            func = input('Enter function (örn: @(x) x^2 - 4;): ');
            a = input('Enter lower bound: ');
            b = input('Enter upper bound: ');
            tol = input('Enter tolerance: ');
            maxIter = input('Enter maximum number of iterations: ');
            bisectionMethod(func, a, b, tol, maxIter);
        case 2
            disp('Secant Method');
            func = input('Enter function (örn: @(x) x^2 - 4;): ');
            x0 = input('Enter initial guess x0: ');
            x1 = input('Enter initial guess x1: ');
            tol = input('Enter tolerance: ');
            maxIter = input('Enter maximum number of iterations: ');
            secantMethod(func, x0, x1, tol, maxIter);
        otherwise
            disp('Geçersiz Seçim.');
    end
end

function linearEquationSystemSolutionMenu()
    disp('Linear Equation System Solution Menu:');
    disp('1. Jacobi Method');
    disp('2. Gauss-Seidel Method');
    disp('3. SOR Technique');
    methodChoice = input('Enter your choice: ');
    switch methodChoice
         case 1
            disp('Jacobi Method');
            A = input('Enter matrix A: ');
            b = input('Enter vector b: ');
            x0 = input('Enter initial guess x0: ');
            tol = input('Enter tolerance: ');
            max_iter = input('Enter maximum number of iterations: ');
            jacobi(A, b, x0, tol, max_iter);
        case 2
            disp('Gauss-Seidel Method');
            A = input('Enter matrix A: ');
            b = input('Enter vector b: ');
            x0 = input('Enter initial guess x0: ');
            tol = input('Enter tolerance: ');
            max_iter = input('Enter maximum number of iterations: ');
            gauss_seidel(A, b, x0, tol, max_iter);
        case 3
            disp('SOR Technique');
            A = input('Enter matrix A: ');
            b = input('Enter vector b: ');
            x0 = input('Enter initial guess x0: ');
            omega = input('Enter relaxation factor (omega): ');
            tol = input('Enter tolerance: ');
            max_iter = input('Enter maximum number of iterations: ');
            sor(A, b, x0, omega, tol, max_iter);
        otherwise
            disp('Geçersiz Seçim');
    end
end
```

```
function curveFittingMenu()
    disp('Curve Fitting Menu:');
    disp('1. Vandermonde Technique');
    disp('2. Lagrange Interpolation');
    disp('3. Newton Interpolation');
    methodChoice = input('Enter your choice: ');
    switch methodChoice
        case 1
            disp('Vandermonde Technique');
            x = input('Enter x values (e.g., [x1, x2, x3]): ');
            y = input('Enter corresponding y values (e.g., [y1, y2, y3]): ');
            vandermondeTechnique(x, y);
        case 2
            disp('Lagrange Interpolation');
            x = input('Enter x values (e.g., [x1, x2, x3]): ');
            y = input('Enter corresponding y values (e.g., [y1, y2, y3]): ');
            targetX = input('Enter the target x value for interpolation: ');
            lagrangeInterpolation(x, y, targetX);
        case 3
            disp('Newton Interpolation');
             x = input('Enter x values (e.g., [x1, x2, x3]): ');
             y = input('Enter corresponding y values (e.g., [y1, y2, y3]): ');
            targetX = input('Enter the target x value for interpolation: ');
             newtonInterpolation(x, y, targetX);
        otherwise
            disp('Geçersiz Seçim');
    end
end


% çağırmak için -> numericalMethodsMenu();
```

Test 1. Menü Arayıcılığı ile Vandermonde Yöntemi Kullanımı

```
Command Window
>>
>> numericalMethodsMenu();
Numerical Methods Menu:
1. Nonlinear Equation Solution
2. Nonlinear Equation System Solution
3. Linear Equation System Solution
4. Curve fitting
5. Exit
Enter your choice: 4
Curve Fitting Menu:
1. Vandermonde Technique
2. Lagrange Interpolation
3. Newton Interpolation
Enter your choice: 1
Vandermonde Technique
Enter x values (e.g., [x1, x2, x3]): [1, 2, 3]
Enter corresponding y values (e.g., [y1, y2, y3]): [4, 7, 10]
Polynomial coefficients:
     1     3     0
```

Test 2. Menü Arayıcılığı ile Jacobi Yöntemi Kullanımı

```
Command Window
Numerical Methods Menu:
1. Nonlinear Equation Solution
2. Nonlinear Equation System Solution
3. Linear Equation System Solution
4. Curve fitting
5. Exit
Enter your choice: 3
Linear Equation System Solution Menu:
1. Jacobi Method
2. Gauss-Seidel Method
3. SOR Technique
Enter your choice: 1
Jacobi Method
Enter matrix A: [12 -2 3 1; -2 15 6 -3; 1 6 20 -4; 0 -3 2 9]
Enter vector b: [-3; 5; 20; 4]
Enter initial guess x0: [1; 2; 3; 4]
Enter tolerance: 1e-12
Enter maximum number of iterations: 50
Jacobi method converged in 31 iterations.
Solution:    -0.5639
   -0.1539
    1.1039
    0.1478
```