MATH 446: Project 07

Zachary Ferguson

March 21, 2017

Contents

Code

Jacobi Method

```
% Solve a system of linear equations, Ax = b, using the Jacobi Method
% Written by Zachary Ferguson

function xc = jacobi_method(A, b, x0, eps)
    % Solve the equation Ax = b using the Jacobi Method
    % Input:
    %   A - matrix of coefficients to the linear equations
    %   b - Right hand side of the linear equations
    %   x0 - intial guess for solution vector
    %   eps - tolerance of forward error
    % Output:
    %   xc - computed solution to a eps tolerance
    if nargin < 4
        eps = 1e-6;
    end

    n = size(A, 1);
    D = spdiags(spdiags(A, 0), 0, n, n);
    L_plus_U = A - D; % L + U = A - D
    D_inv = spdiags(spdiags(A,0).^-1, 0, n, n); % = D^-1

    x_prev = x0;
    xc = D_inv*(b - L_plus_U * x0);
    n_steps = 1;
    while (norm(xc - x_prev, inf) >= 0.5 * eps)
        x_prev = xc;
        xc = D_inv*(b - (L_plus_U) * xc);
        n_steps = n_steps + 1;
    end
    fprintf('\tNumber of steps to find solution: %d\n', n_steps);
end
```

Gauss-Seidel Method

```
% Solve a system of linear equations, Ax = b, using the Guass-Seidel Method
% Written by Zachary Ferguson

function xc = gauss_seidel_method(A, b, x0, eps)
    % Solve the equation Ax = b using the Guass-Seidel Method
    % Input:
    %   A - matrix of coefficients to the linear equations
    %   b - Right hand side of the linear equations
    %   x0 - intial guess for solution vector
    %   eps - tolerance of forward error
    % Output:
    %   xc - computed solution to a eps tolerance
    if nargin < 4
        eps = 1e-6;
    end

    U = triu(A, 1);
    L_plus_D_inv = (A - U)^-1; % (L + D = A - U)^-1

    x_prev = x0;
    xc = L_plus_D_inv*(b - U * x0);
    n_steps = 1;
    while (norm(xc - x_prev, inf) >= 0.5 * eps)
        x_prev = xc;
        xc = L_plus_D_inv*(b - U * xc);
        n_steps = n_steps + 1;
    end
    fprintf('\tNumber of steps to find solution: %d\n', n_steps);
end
```

Successive Over Relaxation

```
% Solve a system of linear equations, Ax = b, using the Successive Over
% Relaxation.
% Written by Zachary Ferguson

function xc = successive_over_relaxation(A, b, x0, omega, eps)
    % Solve the equation Ax = b using the Successive Over Relaxation
    % Input:
    %   A - matrix of coefficients to the linear equations
    %   b - Right hand side of the linear equations
    %   x0 - intial guess for solution vector
    %   omega - parameter for how much to relax
    %   eps - tolerance of forward error
    % Output:
    %   xc - computed solution to a eps tolerance
    if nargin < 5
        eps = 1e-6;
    end

    % A = L + D + U
```

```
    U = triu(A, 1);
    L = tril(A, -1);
    D = A - L - U;
    omegaL_plus_D_inv = (omega*L + D)^-1; % (omegaL + D)^-1
    omega_rhs = omega*omegaL_plus_D_inv*b; % omega(omegaL + D)^-1b

    x_prev = x0;
    % x_{k+1} = (omegaL + D)^-1[(1-omega)Dx_k - omegaUx_k]+omega(omegaL + D)^-1b
    xc = omegaL_plus_D_inv*((1-omega)*D*x0 - omega*U*x0) + omega_rhs;
    n_steps = 1;
    while (norm(xc - x_prev, inf) >= 0.5 * eps)
        x_prev = xc;
        xc = omegaL_plus_D_inv*((1-omega)*D*xc - omega*U*xc) + omega_rhs;
        n_steps = n_steps + 1;
    end
    fprintf('\tNumber of steps to find solution: %d\n', n_steps);
end
```

Main

```
% MATH 446: Project 07
% Written by Zachary Ferguson

function main()
    fprintf('MATH 446: Project 07\nWritten by Zachary Ferguson\n\n');

    fprintf('Jacobi Method:\n\n')
    n = 100;
    fprintf('Q1a:\n\tn=%d\n', n);
    [A, b] = build_system(n);
    x = ones(n, 1);
    xc = jacobi_method(A, b, zeros(n, 1));
    print_errors(A, b, x, xc);

    m = 100000;
    fprintf('Q1b:\n\tn=%d\n', m);
    [C, d] = build_system(m);
    y = ones(m, 1);
    yc = jacobi_method(C, d, zeros(m, 1));
    print_errors(C, d, y, yc);

    fprintf('\nGauss-Seidel Method:\n\n');
    fprintf('Q5a:\n\tn=%d\n', n);
    xc = gauss_seidel_method(A, b, zeros(n, 1));
    print_errors(A, b, x, xc);

    fprintf('\nSuccessive Over Relaxation:\n\n');
    fprintf('Q5b:\n\tn=%d\n', n);
    xc = successive_over_relaxation(A, b, zeros(n, 1), 1.2);
    print_errors(A, b, x, xc);
end

function [A, b] = build_system(n)
```

```
    diag_elements = [[-1 * ones(n-1, 1); 0], 3*ones(n, 1), ...
        [[0; -1 * ones(n-1, 1)]]];
    diag_indices = [-1; 0; 1];
    A = spdiags(diag_elements, diag_indices, n, n);
    b = [2; ones(n-2, 1); 2];
end

function print_errors(A, b, x, xc)
    BE = norm(b - A*xc, inf); % infiniry norm
    FE = norm(x - xc, inf);

    fprintf('\tBackwards Error = %g\n', BE);
    fprintf('\tForwards Error = %g\n', FE);
end
```

Output

```
MATH 446: Project 07
Written by Zachary Ferguson

Jacobi Method:

Q1a:
    n=100
    Number of steps to find solution: 35
    Backwards Error = 6.86783e-07
    Forwards Error = 6.86761e-07
Q1b:
    n=100000
    Number of steps to find solution: 35
    Backwards Error = 6.86783e-07
    Forwards Error = 6.86761e-07

Gauss-Seidel Method:

Q5a:
    n=100
    Number of steps to find solution: 21
    Backwards Error = 4.77934e-07
    Forwards Error = 4.76837e-07

Successive Over Relaxation:

Q5b:
    n=100
    Number of steps to find solution: 18
    Backwards Error = 2.63134e-07
    Forwards Error = 6.61383e-08
```