MATH 446: Project 09

Zachary Ferguson

April 04, 2017

Contents

Code

Multivariate Newton's Method

```
% Computes the roots of a vector valued function using the Newton's Method.
% Written by Zachary Ferguson

function xc = multivariate_newtons_method(f, df, x0, tol, figHandle)
    % Compute the root to f(x) using Newton's Method
    % Input:
    %   f - vector valued function to find the roots of
    %   df - Jacobian of f(x)
    %   x0 - intial guess
    %   tol - tolerance for the root
    % Output:
    %   xc - computed root to the function f(x).
    if nargin < 4
        tol = 1e-8;
    end
    if nargin < 5
        figHandle = false;
    end

    n_steps = 0;
    xc = x0;
    fe = norm(f(xc), inf);
    errors = [fe];
    while fe > tol
        s = df(xc) \ -f(xc);
        xc = xc + s;
        n_steps = n_steps + 1;
        fe = norm(f(xc), inf);
        errors = [errors fe];
    end
    fprintf('\tNumber of steps to solve to %g accuracy: %d\n', tol, n_steps);
```

```
    if figHandle ~= false
        figure(figHandle);
        plot(1:size(errors, 2), errors, '-ob');
    end
end
```

Broyden's Method I

```
% Computes the roots of a vector valued function using the Broden's Method I.
% Written by Zachary Ferguson

function xc = broydens_method_1(f, A0, x0, tol, figHandle)
    % Compute the root to f(x) using Newton's Method
    % Input:
    %   f - vector valued function to find the roots of
    %   A0 - inital approximation for the Jacobian of f(x)
    %   x0 - intial guess
    %   tol - tolerance for the root
    % Output:
    %   xc - computed root to the function f(x).
    if nargin < 4
        tol = 1e-8;
    end
    if nargin < 5
        figHandle = false;
    end

    n_steps = 0;

    xc = x0;
    A = A0;

    fe = norm(f(xc), inf);
    errors = [fe];
    while fe > tol
        s = A \ -f(xc);
        x_prev = xc;
        xc = xc + s;
        delta_f = f(xc) - f(x_prev);
        delta_x = xc - x_prev;
        A = A + ((delta_f - A * delta_x) * delta_x') / (delta_x' * delta_x);

        n_steps = n_steps + 1;
        fe = norm(f(xc), inf);
        errors = [errors fe];
    end
    fprintf('\tNumber of steps to solve to %g accuracy: %d\n', tol, n_steps);

    % Display the errors per iteration.
    if figHandle ~= false
        figure(figHandle);
        plot(1:size(errors, 2), errors, '-xr');
    end
```

```
end
```

Broyden's Method II

```
% Computes the roots of a vector valued function using the Broden's Method II.
% Written by Zachary Ferguson

function xc = broydens_method_2(f, B0, x0, tol, figHandle)
    % Compute the root to f(x) using Newton's Method
    % Input:
    %   f - vector valued function to find the roots of
    %   A0 - inital approximation for the inverse of the Jacobian of f(x)
    %   x0 - intial guess
    %   tol - tolerance for the root
    % Output:
    %   xc - computed root to the function f(x).
    if nargin < 4
        tol = 1e-8;
    end
    if nargin < 5
        figHandle = false;
    end

    n_steps = 0;

    xc = x0;
    B = B0; % B = A^-1

    fe = norm(f(xc), inf);
    errors = [fe];
    while fe > tol
        s = -B * f(xc); % = A^-1 * -f(xc)
        x_prev = xc;
        xc = xc + s;
        delta_f = f(xc) - f(x_prev); % Big Delta
        delta_x = xc - x_prev; % Little Delta
        B = B + ((delta_x - B * delta_f) * delta_x' * B) / ...
            (delta_x' * B * delta_f);

        n_steps = n_steps + 1;
        fe = norm(f(xc), inf);
        errors = [errors fe];
    end
    fprintf('\tNumber of steps to solve to %g accuracy: %d\n', tol, n_steps);

    % Display the errors per iteration.
    if figHandle ~= false
        figure(figHandle);
        plot(1:size(errors, 2), errors, '-dg');
    end
end
```

Main

```
% MATH 446: Project 09
% Written by Zachary Ferguson

function main()
    fprintf('MATH 446: Project 09\nWritten by Zachary Ferguson\n\n');

    % Function and Jacobian used in part a:
    [f_a, df_a] = build_funtion_and_jacobian([1,1,0], 1, [1,0,1], 1, ...
        [0,1,1], 1);
    % Function and Jacobian used in part b:
    [f_b, df_b] = build_funtion_and_jacobian([1,-2,0], 5, [-2,2,-1], 5, ...
        [4,-2,3], 5);

    titles = ['A', 'B'];
    figures = [];
    for i = 1:4
        figures = [figures figure];
        title(sprintf('Part %s (Solution %d): Comparison of Methods', ...
            titles(ceil(i / 2)), mod(i-1, 2) + 1));
        xlabel('Iteration Step');
        ylabel('Forward error of x_k');
        set(gca, 'YScale', 'log');
        hold on;
    end

    tol = 1e-10;

    % Q5a
    fprintf('Q5a:\n\tIntersection Point 1:\n');
    x0_a1 = zeros(3, 1);
    fprintf('\tx0 = \n');
    disp(x0_a1);
    xc = multivariate_newtons_method(f_a, df_a, x0_a1, tol, figures(1));
    fprintf('\txc = \n');
    disp(xc);

    fprintf('\tIntersection Point 2:\n');
    x0_a2 = 2*ones(3, 1);
    fprintf('\tx0 = \n');
    disp(x0_a2);
    xc = multivariate_newtons_method(f_a, df_a, x0_a2, tol, figures(2));
    fprintf('\txc = \n');
    disp(xc);

    % Q5b
    fprintf('Q5b:\n\tIntersection Point 1:\n');
    x0_b1 = -2*ones(3, 1);
    fprintf('\tx0 = \n');
    disp(x0_b1);
    xc = multivariate_newtons_method(f_b, df_b, x0_b1, tol, figures(3));
    fprintf('\txc = \n');
    disp(xc);
```

```
fprintf('\tIntersection Point 2:\n');
x0_b2 = 2*ones(3, 1);
fprintf('\tx0 = \n');
disp(x0_b2);
xc = multivariate_newtons_method(f_b, df_b, x0_b2, tol, figures(4));
fprintf('\txc = \n');
disp(xc);

% Q9a
fprintf('Q9a:\n\tIntersection Point 1:\n');
xc = broydens_method_1(f_a, eye(3), x0_a1, tol, figures(1));
fprintf('\txc = \n');
disp(xc);

fprintf('\tIntersection Point 2:\n');
xc = broydens_method_1(f_a, eye(3), x0_a2, tol, figures(2));
fprintf('\txc = \n');
disp(xc);

% Q9b
fprintf('Q9b:\n\tIntersection Point 1:\n');
xc = broydens_method_1(f_b, eye(3), x0_b1, tol, figures(3));
fprintf('\txc = \n');
disp(xc);

fprintf('\tIntersection Point 2:\n');
xc = broydens_method_1(f_b, eye(3), x0_b2, tol, figures(4));
fprintf('\txc = \n');
disp(xc);

% Q11a
fprintf('Q11a:\n\tIntersection Point 1:\n');
xc = broydens_method_2(f_a, eye(3), x0_a1, tol, figures(1));
fprintf('\txc = \n');
disp(xc);

fprintf('\tIntersection Point 2:\n');
xc = broydens_method_2(f_a, eye(3), x0_a2, tol, figures(2));
fprintf('\txc = \n');
disp(xc);

% Q11b
fprintf('Q11b:\n\tIntersection Point 1:\n');
xc = broydens_method_2(f_b, eye(3), x0_b1, tol, figures(3));
fprintf('\txc = \n');
disp(xc);

fprintf('\tIntersection Point 2:\n');
xc = broydens_method_2(f_b, eye(3), x0_b2, tol, figures(4));
fprintf('\txc = \n');
disp(xc);

for i = 1:4
```

```
        figure(figures(i));
        legend(['Multivariate Newton''' 's Method'], ...
            ['Broyden''' 's Method I'], ...
            ['Broyden''' 's Method II']);
        hold off;
    end
end

function [f, df] = build_funtion_and_jacobian(c1, r1, c2, r2, c3, r3)
    % Build a function f(x) for the intersection of three circles.
    % Helper function for building f and df in Q5.
    % Function
    f = @(x) [(x(1)-c1(1))^2 + (x(2)-c1(2))^2 + (x(3)-c1(3))^2 - r1^2; ...
              (x(1)-c2(1))^2 + (x(2)-c2(2))^2 + (x(3)-c2(3))^2 - r2^2; ...
              (x(1)-c3(1))^2 + (x(2)-c3(2))^2 + (x(3)-c3(3))^2 - r3^2];
    % Jacobian
    df = @(x) [2*(x(1)-c1(1)) 2*(x(2)-c1(2)) 2*(x(3)-c1(3)); ...
               2*(x(1)-c2(1)) 2*(x(2)-c2(2)) 2*(x(3)-c2(3)); ...
               2*(x(1)-c3(1)) 2*(x(2)-c3(2)) 2*(x(3)-c3(3))];
end
```

Output

```
MATH 446: Project 09
Written by Zachary Ferguson

Q5a:
        Intersection Point 1:
        x0 =
   0
   0
   0
        Number of steps to solve to 1e-10 accuracy: 5
        xc =
   0.33333
   0.33333
   0.33333
        Intersection Point 2:
        x0 =
   2
   2
   2
        Number of steps to solve to 1e-10 accuracy: 6
        xc =
   1.0000
   1.0000
   1.0000
Q5b:
        Intersection Point 1:
        x0 =
  -2
  -2
  -2
```

```
        Number of steps to solve to 1e-10 accuracy: 11
        xc =
   1.0000
   2.0000
   3.0000
        Intersection Point 2:
        x0 =
   2
   2
   2
        Number of steps to solve to 1e-10 accuracy: 5
        xc =
   1.8889
   2.4444
   2.1111
Q9a:
        Intersection Point 1:
        Number of steps to solve to 1e-10 accuracy: 9
        xc =
   0.33333
   0.33333
   0.33333
        Intersection Point 2:
        Number of steps to solve to 1e-10 accuracy: 15
        xc =
   1.00000
   1.00000
   1.00000
Q9b:
        Intersection Point 1:
        Number of steps to solve to 1e-10 accuracy: 20
        xc =
   1.0000
   2.0000
   3.0000
        Intersection Point 2:
        Number of steps to solve to 1e-10 accuracy: 21
        xc =
   1.8889
   2.4444
   2.1111
Q11a:
        Intersection Point 1:
        Number of steps to solve to 1e-10 accuracy: 9
        xc =
   0.33333
   0.33333
   0.33333
        Intersection Point 2:
        Number of steps to solve to 1e-10 accuracy: 13
        xc =
   1.00000
   1.00000
   1.00000
```

```
Q11b:
        Intersection Point 1:
        Number of steps to solve to 1e-10 accuracy: 20
        xc =
   1.0000
   2.0000
   3.0000
        Intersection Point 2:
        Number of steps to solve to 1e-10 accuracy: 21
        xc =
   1.8889
   2.4444
   2.1111
```

Figures