06-262 Spring 2014: Newton’s Method in MATLAB

From Pseudo-Code to Implementation in MATLAB
Introduction

• Newton’s Method (also known as Newton-Raphson Method) is used to solve nonlinear (system) of equations, which can be represented as follows:

\[ f(x) = 0 \]

where \( f(\cdot) \) is a general univariate nonlinear function and \( x \) is a (scalar) variable, and

\[ F(x) = 0 \]

where \( F(\cdot) \) is a vector of general multivariate nonlinear functions and \( x \) is a vector of variables.
Examples

• Single nonlinear equation

\[
\frac{1}{2} \frac{\epsilon^{3/2}}{(1 - \epsilon)(1 + \frac{1}{2} \epsilon)^{1/2}} - K_{eq} = 0
\]

where \( \epsilon \) is the (unknown) extent of reaction (variable to be solved for) and \( K_{eq} \) is a (known) equilibrium constant (a number)

• System of nonlinear equations

\[
\mu_{\text{max}} \left(1 - \frac{C_p}{C_p^*}\right)^{0.52} \frac{C_c C_s}{K_s + C_s} - k_d C_c = 0
\]

\[
-\frac{Y_s/c}{c} \mu_{\text{max}} \left(1 - \frac{C_p}{C_p^*}\right)^{0.52} \frac{C_c C_s}{K_s + C_s} - m C_c = 0
\]

\[
\frac{Y_p/c}{c} \mu_{\text{max}} \left(1 - \frac{C_p}{C_p^*}\right)^{0.52} \frac{C_c C_s}{K_s + C_s} = 0
\]

where \( C_c, \ C_p, \) and \( C_s \) are unknowns and the other symbols are known
Newton’s Method: Pseudo-Code

• Idea: given initial guess (crucial, as close to the solution as possible), iterate towards the solution of the nonlinear equation(s) using first-order derivative information (makes convergence fast). Method converges to one solution, but multiple solutions may exist that depend on the chosen initial guess.
• Pseudo-code for the univariate case:

```
Input: initial guess (x₀), maximum number of iterations (N), convergence criterion (tol)
Output: solution to nonlinear equation

n ← 1
while (n ≤ N)
    fₙ₋₁ ← f(xₙ₋₁)
    f′ₙ₋₁ ← f′(xₙ₋₁)
    xₙ ← xₙ₋₁ − fₙ₋₁/f′ₙ₋₁
    if (|fₙ₋₁| ≤ tol) then
        break
    end
end
```
Newton’s Method: MATLAB Code

### Pseudo-Code

<table>
<thead>
<tr>
<th>n \leftarrow 1</th>
<th>n = 2;</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textbf{while } (n \leq N)</td>
<td>\textbf{while } (n &lt;= N + 1)</td>
</tr>
<tr>
<td>\hspace{0.5cm} f_{n-1} \leftarrow f(x_{n-1})</td>
<td>\hspace{0.5cm} fe = f(x(n - 1));</td>
</tr>
<tr>
<td>\hspace{0.5cm} f'<em>{n-1} \leftarrow f'(x</em>{n-1})</td>
<td>\hspace{0.5cm} fpe = fp(x(n - 1));</td>
</tr>
<tr>
<td>\hspace{0.5cm} x_n \leftarrow x_{n-1} - f_{n-1}/f'_{n-1}</td>
<td>\hspace{0.5cm} x(n) = x(n - 1) - fe/fpe;</td>
</tr>
<tr>
<td>\textbf{if } (</td>
<td>f_{n-1}</td>
</tr>
<tr>
<td>\hspace{0.5cm} \textbf{break}</td>
<td>\hspace{0.5cm} \textbf{break};</td>
</tr>
<tr>
<td>\textbf{end}</td>
<td>\textbf{end}</td>
</tr>
<tr>
<td>n \leftarrow n + 1</td>
<td>n = n + 1;</td>
</tr>
<tr>
<td>\textbf{end}</td>
<td>\textbf{end}</td>
</tr>
</tbody>
</table>

### MATLAB

<table>
<thead>
<tr>
<th>n = 2;</th>
<th>n = 2;</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textbf{while } (n &lt;= N + 1)</td>
<td>\textbf{while } (n &lt;= N + 1)</td>
</tr>
<tr>
<td>\hspace{0.5cm} fe = f(x(n - 1));</td>
<td>\hspace{0.5cm} fe = f(x(n - 1));</td>
</tr>
<tr>
<td>\hspace{0.5cm} fpe = fp(x(n - 1));</td>
<td>\hspace{0.5cm} fpe = fp(x(n - 1));</td>
</tr>
<tr>
<td>\hspace{0.5cm} x(n) = x(n - 1) - fe/fpe;</td>
<td>\hspace{0.5cm} x(n) = x(n - 1) - fe/fpe;</td>
</tr>
<tr>
<td>\textbf{if } (\text{abs}(fe) \leq \text{tol}) \textbf{ then}</td>
<td>\textbf{if } (\text{abs}(fe) \leq \text{tol})</td>
</tr>
<tr>
<td>\hspace{0.5cm} \textbf{break};</td>
<td>\hspace{0.5cm} \textbf{break};</td>
</tr>
<tr>
<td>\textbf{end}</td>
<td>\textbf{end}</td>
</tr>
<tr>
<td>n = n + 1;</td>
<td>n = n + 1;</td>
</tr>
<tr>
<td>\textbf{end}</td>
<td>\textbf{end}</td>
</tr>
</tbody>
</table>

- Note that arrays in MATLAB are one-based, thus $x(1) \Leftrightarrow x_0$
- Also note that the function (equation) and its first-order derivative must be provided
- You can use “anonymous functions” in MATLAB to avoid creating an M-file for each function
f = @(x) x + x^(4/3); % Equation definition
fp = @(x) 1 + 4/3*x^(1/3); % First-order derivative of f
x0 = 1; % Initial guess
N = 10; % Maximum number of iterations
tol = 1E-6; % Convergence tolerance
x = zeros(N + 1,1); % Preallocate solution vector where row => iteration
x(1) = x0; % Set initial guess

% Newton's Method algorithm
n = 2;
nfinal = N + 1; % Store final iteration if tol is reached before N iterations
while (n <= N + 1)
    fe = f(x(n - 1));
    fpe = fp(x(n - 1));
    x(n) = x(n - 1) - fe/fpe;
    if (abs(fe) <= tol)
        nfinal = n; % Store final iteration
        break;
    end
    n = n + 1;
end

% Plot evolution of the solution
figure('Color','White')
plot(0:nfinal - 1,x(1:nfinal),'o-')
title('Newton''s Method Solution: $f(x) = x + x^\frac{4}{3}$','FontSize',20,'Interpreter','latex')
xlabel('Iteration','FontSize',16)
ylabel('$x$','FontSize',16,'Interpreter','latex')
Numeric Example: Plot

Newton’s Method Solution: $f(x) = x + x^\frac{1}{3}$
MATLAB Built-in Functions

• For univariate nonlinear equations, you may use the function `fzero`

• For a system of nonlinear equations, you may use the function `fsolve`

• They are more robust and rigorous than the simple implementation shown in the previous slides