ME451
Kinematics and Dynamics of Machine Systems

Newton-Raphson Method 4.5
October 28, 2010
Before we get started...

- Last Time
  - Singular configuration of mechanisms (3.7, likely on Th)

- Today:
  - Discuss Newton-Raphson Method (See section 4.5 – needed for take-home)
  - Short lecture, hosting the seminar speaker

- Next week
  - HW due on Tu
  - Exam on Tu (November 2)
  - Review session in this room, Monday, 6:00 PM
  - Exam:
    - Closed book
    - One sheet of paper, both sides – write down any equations you might find useful
Solving a Nonlinear System

- The most important numerical algorithm to understand in Kinematics

- Relied upon heavily by ADAMS, used almost in all analysis modes
  - Kinematics
  - Dynamics
  - Equilibrium

- How does one go about finding the solution?

\[
\begin{align*}
\sqrt{x} - \sin x &= 0 \\
\sqrt{x} - e^y &= 1 \\
\ln (1 + x) - \cos y &= 0
\end{align*}
\]
Newton-Raphson Method

- Start by looking at the one-dimension case:
  - Find the solution $x^*$ of the equation:

$$f(x) = 0$$

- Assumption:
  - The function $f : \mathbb{R} \rightarrow \mathbb{R}$ is twice differentiable
Newton-Raphson Method

Algorithm

Start with initial guess $x^{(0)}$ and then compute $x^{(1)}$, $x^{(2)}$, $x^{(3)}$, etc.

\[
x^{(1)} = x^{(0)} - \frac{f(x^{(0)})}{f'(x^{(0)})}
\]

\[
x^{(2)} = x^{(1)} - \frac{f(x^{(1)})}{f'(x^{(1)})}
\]

\[
x^{(3)} = x^{(2)} - \frac{f(x^{(2)})}{f'(x^{(2)})}
\]

This is called an iterative algorithm:

- First you get $x^{(1)}$, then you improve the predicted solution to $x^{(2)}$, then improve more yet to $x^{(3)}$, etc.
- Iteratively, you keep getting closer and closer to the actual solution
Example:

\[ f(x) = x^2 + \sin x - 1.841471 = 0 \]
\[ f'(x) = 2x + \cos x \]

<table>
<thead>
<tr>
<th>STEP</th>
<th>(x)</th>
<th>(f'(x))</th>
<th>(f(x))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>3.5838</td>
<td>3.0678</td>
</tr>
<tr>
<td>1</td>
<td>(2 - 3.0678 / 3.5838 = 1.1439)</td>
<td>2.7109</td>
<td>0.3775</td>
</tr>
<tr>
<td>2</td>
<td>(1.1439 - 0.3775 / 2.7109 = 1.004)</td>
<td>2.5451</td>
<td>0.0107</td>
</tr>
</tbody>
</table>

The exact answer is \(x^* = 1.0\)
Understanding Newton’s Method

- What you want to do is find the root of a function, that is, you are looking for that $x$ that makes the function zero.

- The stumbling block is the nonlinear nature of the function.

- Newton’s idea was to linearize the nonlinear function:
  - Then look for the root of this new linear function that hopefully approximates well the original nonlinear function.
  - If the only thing that you have is a hammer, then make all your problems a nail.
Understanding Newton’s Method (Cntd.)

- You are at point $q^{(0)}$ and linearize using Taylor’s expansion

$$f(q) = f(q^{(0)}) + \frac{f'(q^{(0)})}{1!}(q - q^{(0)}) + \frac{f''(q^{(0)})}{2!}(q - q^{(0)})^2 + \ldots = \sum_{n=0}^{\infty} \frac{f^{(n)}(q^{(0)})}{n!}(q - q^{(0)})^n$$

- After linearization, you end up with linear approximation $h(q)$ of $f(q)$:

$$h(q) = f(q^{(0)}) + \frac{f'(q^{(0)})}{1!}(q - q^{(0)})$$

**NOTE:** $f(q) \approx h(q)$

- Find now $q^{(1)}$ that is the root of $h$, that is: $h(q^{(1)}) = 0$

$$h(q^{(1)}) = f(q^{(0)}) + \frac{f'(q^{(0)})}{1!}(q^{(1)} - q^{(0)}) = 0 \quad \Rightarrow \quad q^{(1)} = q^{(0)} - \frac{f(q^{(0)})}{f'(q^{(0)})}$$
Newton-Raphson Method

Geometric Interpretation
Newton-Raphson
The Multidimensional Case

- Solve for \( q \in \mathbb{R}^n \) the nonlinear system

\[
\Phi(q, t) = \begin{bmatrix}
\Phi_1(q, t) \\
\Phi_2(q, t) \\
\vdots \\
\Phi_n(q, t)
\end{bmatrix} = 0
\]

- The algorithm becomes

\[
q^{(1)} = q^{(0)} - [\Phi_q(q^{(0)})]^{-1} \Phi(q^{(0)}, t)
\]

- The Jacobian is defined as

\[
\Phi_q(q^{(0)}) = \left. \frac{\partial \Phi}{\partial q} \right|_{q=q^{(0)}}
\]
Putting things in perspective...

- Newton algorithm for nonlinear systems requires:
  - A starting point \( q^{(0)} \) from where the solution starts being searched for
  - An iterative process in which the approximation of the solution is gradually improved:

\[
q^{(1)} = q^{(0)} - \left[ \Phi_q(q^{(0)}) \right]^{-1} \Phi(q^{(0)}, t)
\]

\[
q^{(2)} = q^{(1)} - \left[ \Phi_q(q^{(1)}) \right]^{-1} \Phi(q^{(1)}, t)
\]

\[
q^{(3)} = q^{(2)} - \left[ \Phi_q(q^{(2)}) \right]^{-1} \Phi(q^{(2)}, t)
\]

\[
\cdots \text{ etc.}
\]
Example: Solve nonlinear system:

\[
\begin{align*}
\frac{d}{dx} x - e^y & = 1 \\
\ln (1 + x) - \cos y & = 0
\end{align*}
\]

Initial guess:
\[
\begin{align*}
x & = 1 \\
y & = 0
\end{align*}
\]

Iteration 1
\[
\begin{align*}
x & = 1.6137 \\
y & = 0.6137
\end{align*}
\]

Iteration 2
\[
\begin{align*}
x & = 1.5123 \\
y & = 0.4324
\end{align*}
\]

Iteration 3
\[
\begin{align*}
x & = 1.5042 \\
y & = 0.4085
\end{align*}
\]

Iteration 4
\[
\begin{align*}
x & = 1.5040 \\
y & = 0.4081
\end{align*}
\]

Iteration 5
\[
\begin{align*}
x & = 1.5040 \\
y & = 0.4081
\end{align*}
\]

Residual: \(10^{-6}\)
Example: Position Analysis of Mechanism

- Problem 3.5.6, modeled with reduced set of generalized coordinates (see handout)

\[
\Phi(q, t) = \begin{bmatrix}
x_2 - 0.6 + 0.25 \sin \phi_3 \\
y_2 - 0.25 \cos \phi_3 \\
x_2^2 + y_2^2 - \left( \frac{t}{10} + 0.4 \right)^2
\end{bmatrix} = 0
\]

\[
\Phi_q(q) = \begin{bmatrix}
1 & 0 & 0.25 \cos \phi_3 \\
0 & 1 & 0.25 \sin \phi_3 \\
2x_2 & 2y_2 & 0
\end{bmatrix}
\]

- Iterations carried out like:

\[
q^{(k+1)} = q^{(k)} - [\Phi_q(q^{(k)}, t)]^{-1} \Phi(q^{(k)}, t)
\]

Figure P3.5.6
Dumb approach, which nonetheless works…

% Problem 3.5.6
% Solves the nonlinear system associated with Position Analysis
% Numerical solution found using Newton's method.

% get a starting point, that is, an initial guess
q = [0.2 ; 0.2 ; pi/4]

time = 0;
% start improving the guess
%disp(Iteration 1);
dummy1 = q(1) - 0.6 + 0.25*sin(q(3));
dummy2 = q(2) - 0.25*cos(q(3));
dummy3 = q(1)*q(1) + q(2)*q(2) - (time/10+0.4)*(time/10+0.4);
residual = [ dummy1 ; dummy2 ; dummy3 ];
jacobian = [ 1 0 0.25*cos(q(3)) ;
             0 1 0.25*sin(q(3));
             2*q(1)  2*q(2)  0 ];
correction = jacobian\residual;
q = q - correction
%disp(Iteration 2);
dummy1 = q(1) - 0.6 + 0.25*sin(q(3));
dummy2 = q(2) - 0.25*cos(q(3));
dummy3 = q(1)*q(1) + q(2)*q(2) - (time/10+0.4)*(time/10+0.4);
residual = [ dummy1 ; dummy2 ; dummy3 ];
jacobian = [ 1 0 0.25*cos(q(3)) ;
             0 1 0.25*sin(q(3));
             2*q(1)  2*q(2)  0 ];
correction = jacobian\residual;
q = q - correction
%disp(Iteration 3);
dummy1 = q(1) - 0.6 + 0.25*sin(q(3));
dummy2 = q(2) - 0.25*cos(q(3));
dummy3 = q(1)*q(1) + q(2)*q(2) - (time/10+0.4)*(time/10+0.4);
residual = [ dummy1 ; dummy2 ; dummy3 ];
jacobian = [ 1 0 0.25*cos(q(3)) ;
             0 1 0.25*sin(q(3));
             2*q(1)  2*q(2)  0 ];
correction = jacobian\residual;
q = q - correction

disp(Iteration 4);
dummy1 = q(1) - 0.6 + 0.25*sin(q(3));
dummy2 = q(2) - 0.25*cos(q(3));
dummy3 = q(1)*q(1) + q(2)*q(2) - (time/10+0.4)*(time/10+0.4);
residual = [ dummy1 ; dummy2 ; dummy3 ];
jacobian = [ 1 0 0.25*cos(q(3)) ;
             0 1 0.25*sin(q(3));
             2*q(1)  2*q(2)  0 ];
correction = jacobian\residual;
q = q - correction
%disp(Iteration 5);
dummy1 = q(1) - 0.6 + 0.25*sin(q(3));
dummy2 = q(2) - 0.25*cos(q(3));
dummy3 = q(1)*q(1) + q(2)*q(2) - (time/10+0.4)*(time/10+0.4);
residual = [ dummy1 ; dummy2 ; dummy3 ];
jacobian = [ 1 0 0.25*cos(q(3)) ;
             0 1 0.25*sin(q(3));
             2*q(1)  2*q(2)  0 ];
correction = jacobian\residual;
q = q - correction
Example: Position Analysis of Mechanism

- Output of the code shows how the position configuration at time t=0 is gradually improved

<table>
<thead>
<tr>
<th></th>
<th>Initial guess: q =</th>
<th>Iteration 1: q =</th>
<th>Iteration 2: q =</th>
<th>Iteration 3: q =</th>
<th>Iteration 4: q =</th>
<th>Iteration 5: q =</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.2000 0.2000 0.7854</td>
<td>0.4232 0.1768 0.7854</td>
<td>0.3812 0.1348 1.0228</td>
<td>0.3813 0.1216 1.0635</td>
<td>0.3813 0.1210 1.0654</td>
<td>0.3813 0.1210 1.0654</td>
</tr>
</tbody>
</table>

- After 4 iterations it doesn’t make sense to keep iterating, the solution is already very good
Better way to implement Position Analysis…

% Problem 3.5.6
% Solves the nonlinear system associated with Position Analysis
% Numerical solution found using Newton's method.

% get a starting point, that is, an initial guess
q = [0.2 ; 0.2 ; pi/4]

time = 0;
% start improving the guess
for i = 1:6
    crnt_iteration = strcat('Iteration ', int2str(i));
    disp(crnt_iteration);
    dummy1 = q(1) - 0.6 + 0.25*sin(q(3));
    dummy2 = q(2) - 0.25*cos(q(3));
    dummy3 = q(1)*q(1) + q(2)*q(2) - (time/10+0.4)*(time/10+0.4);
    residual = [ dummy1 ; dummy2 ; dummy3 ];
    jacobian = [ 1 0 0.25*cos(q(3)) ;
                 0 1 0.25*sin(q(3));
                 2*q(1) 2*q(2) 0];
    correction = jacobian\residual;
    q = q - correction
end

This way is better since you introduced an iteration loop and go through six iterations before you stop. Compared to previous solution it saves a lot of coding effort and makes it more clear and compact
Even better way to implement Position Analysis...

% Problem 3.5.6
% Solves the nonlinear system associated with Position Analysis
% Numerical solution found using Newton's method.

% get a starting point, that is, an initial guess
q = [0.2 ; 0.2 ; pi/4]

time = 0;
% start improving the guess
normCorrection = 1.0;
tolerance = 1.E-8;
itrationCounter = 0;
while normCorrection>tolerance,
    iterationCounter = iterationCounter + 1;
crnt_iteration = strcat('Iteration ', int2str(iterationCounter));
disp(crnt_iteration);
dummy1 = q(1) - 0.6 + 0.25*sin(q(3));
dummy2 = q(2) - 0.25*cos(q(3));
dummy3 = q(1)*q(1) + q(2)*q(2) - (time/10+0.4)*(time/10+0.4);
residual = [ dummy1 ; dummy2 ; dummy3 ];
jacobian = [1 0 0.25*cos(q(3)) ;
            0 1 0.25*sin(q(3));
            2*q(1) 2*q(2) 0];
correction = jacobian\residual;
q = q - correction;
normCorrection = norm(correction);
end

disp('Here is the value of q:'), q
% Problem 3.5.6
% Solves nonlinear system associated with Position Analysis
% Numerical solution found using Newton's method.

% get a starting point, that is, an initial guess
q = [0.2 ; 0.2 ; pi/4]

time = 0;
% start improving the guess
normCorrection = 1.0;
tolerance = 1.E-8;
iterationCounter = 1;
while normCorrection>tolerance,
    residual = getPhi(q, time);
    jacobian = getJacobian(q);
    correction = jacobian\residual;
    q = q - correction;
    normCorrection = norm(correction);
    iterationCounter = iterationCounter + 1;
end

disp('Here is the value of q:'), q

function phi = getPhi(q, t)
% computes the violation in constraints
% Input : current position q, and current time t
% Output: returns constraint violations

dummy1 = q(1) - 0.6 + 0.25*sin(q(3));
dummy2 = q(2) - 0.25*cos(q(3));
dummy3 = q(1)*q(1) + q(2)*q(2) - (t/10+0.4)*(t/10+0.4);
phi = [ dummy1 ; dummy2 ; dummy3 ];

function jacobian = getJacobian(q)
% computes the Jacobian of the constraints
% Input : current position q
% Output: Jacobian matrix J

jacobian = [ 1 0 0.25*cos(q(3)) ;
            0 1 0.25*sin(q(3));
            2*q(1) 2*q(2) 0 ];
Newton’s Method: Closing Remarks

- Can ever things go wrong with Newton’s method?
- Yes, there are at least three instances:
  1. Most commonly, the starting point is not close to the solution that you try to find and the iterative algorithm diverges (goes to infinity)
  2. Since a nonlinear system can have multiple solutions, the Newton algorithm finds a solution that is not the one sought (happens if you don’t choose the starting point right)
  3. The speed of convergence of the algorithm is not good (happens if the Jacobian is close to being singular (zero determinant) at the root, not that common)
Newton’s Method: Closing Remarks

- What can you do address these issues?
- You cannot do anything about 3 above, but can fix 1 and 2 provided you choose your starting point carefully.
- Newton’s method converges very fast (quadratically) if started close enough to the solution.
- To help Newton’s method in Position Analysis, you can take the starting point of the algorithm at time $t_k$ to be the value of $q$ from $t_{k-1}$ (that is, the very previous configuration of the mechanism).
% Driver for Position Analysis
% Works for kinematic analysis of any mechanism
% User needs to implement three subroutines:
% providenInitialGuess
% getPhi
% getJacobian

% general settings
normCorrection = 1.0;
tolerance = 1.E-8;
timeStep = 0.05;
timeEnd = 4;
timePoints = 0:timeStep:timeEnd;
results = zeros(3,length(timePoints));
crntOutputPoint = 0;

% start the solution loop
q = providenInitialGuess();
for time = 0:timeStep:timeEnd
    crntOutputPoint = crntOutputPoint + 1;
    while normCorrection>tolerance,
        residual = getPhi(q, time);
        jacobian = getJacobian(q);
        correction = jacobian\residual;
        q = q - correction;
        normCorrection = norm(correction);
    end
    results(:, crntOutputPoint) = q;
    normCorrection = 1;
end

function q_zero = providenInitialGuess()
% purpose of function is to provide an initial
% guess to start the Newton algorithm at
% time=0
q_zero = [0.2 ; 0.2 ; pi/4];

function phi = getPhi(q, t)
% computes the violation in constraints
% Input : current position q, and current time t
% Output: returns constraint violations
dummy1 = q(1) - 0.6 + 0.25*sin(q(3));
dummy2 = q(2) - 0.25*cos(q(3));
dummy3 = q(1)*q(1) + q(2)*q(2) - (t/10+0.4)*(t/10+0.4); phi = [ dummy1 ; dummy2 ; dummy3 ];

function jacobian = getJacobian(q)
% computes the Jacobian of the constraints
% Input : current position q
% Output: Jacobian matrix J
jacobian = [
    1 0 0.25*cos(q(3));
    0 1 0.25*sin(q(3));
    2*q(1) 2*q(2) 0 ];
END: Newton Method

END: Kinematics