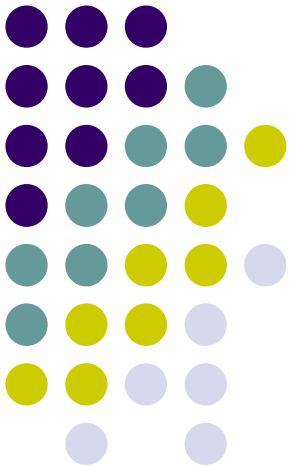
ME751
Advanced Computational Multibody Dynamics

Implicit Integration Methods

BDF Methods

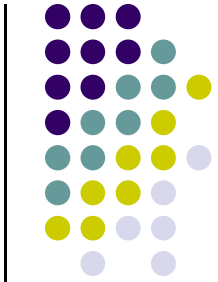
Handling Second Order EOMs

April 06, 2010



"Do what you can, with what you have, where you are."
Theodore Roosevelt

Before we get started…



- Last Time:
  - Discussed Implicit Integration Methods, general considerations
- Today:
  - BDF Methods (one of several families of implicit numerical integration methods)
  - Dealing with 2nd order IVPs
- HW due on Th – challenging
- Exam coming up on April 29, 7:15 PM
  - Closed books (no book to open anyway)
  - Can bring one normal sheet of paper with formulas (both sides)
  - I'll provide the cheat sheet that you received a while ago
- Trip to John Deere & NADS:
  - Need head count by April 8, use forum discussion topic to indicate if you are in or not
  - Transportation and hotel will be covered
  - Leave late afternoon on May 3, return on May 4 at 10 pm or so

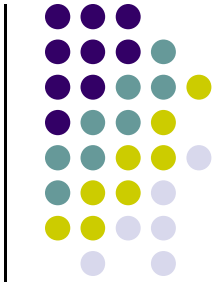
BDF Methods

- BDF stands for Backward Differentiation Formula
- Proposed by Bill Gear in 1970
  - Super nice person
  - Back in ’70s he was a professor in Comp. Science at UIUC
  - Former director of NEC Research Institute
  - Professor Emeritus, Princeton

Bill Gear

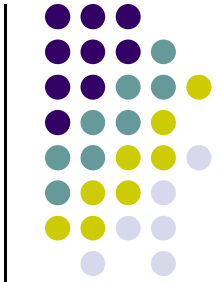
- BDF methods are the most widely used implicit multistep methods
- BDF methods, together with HHT methods, are the two most used to integration formulas in ADAMS (the software package)

BDF Methods
Relation to AB or AM Methods



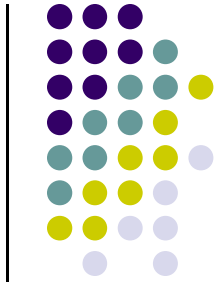
- Recall that in getting AB-M and AM-M methods, the important decision was to integrate the value of function f
  - Integrating at t_n over $f_{n-1},\ldots,f_{n-k}$ led to AB-M
  - Integrating at t_n over $f_n,\ldots,f_{n-k}$ led to AM-M
- The interpolation polynomial was integrated and the outcome was the family of Adams integration formulas

BDF Methods
Relation to AB or AM Methods



- Compared to AM and/or AB Methods, BDF formulas are obtained by making completely opposite choices
  - Rather than interpolating f, we will interpolate y
  - Rather than using the interpolation polynomial to perform a time integration, the interpolation polynomial will be used in computing a time derivative
- Specifically, the points $y_n,\ldots,y_{n-k}$ are used to generate a polynomial that approximates y(t)
- If one takes the time derivative of this interpolation polynomial at time t_n the value obtained should be an approximation of the time derivative of y(t). By setting this time derivative to $f(t_n,y_n)$ one gets a BDF method.

BDF Methods



- The BDF methods are implicit methods
- With α_o=1, they assume the form

$$\sum_{i=0}^{k} \alpha_i y_{n-i} = h\beta_0 f(t_n, y_n)$$

- NOTE: for k>6, the absolute stability region of the resulting BDF methods is so small that they are useless
- Example: BDF of order two

$$y_n - \tfrac{4}{3} y_{n-1} + \tfrac{1}{3} y_{n-2} = \tfrac{2}{3} h f(t_n, y_n)$$

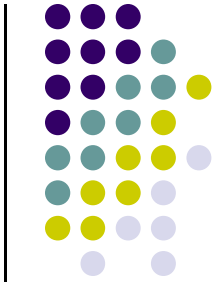
$$\text{Or equivalently,}$$

$$y_n = \tfrac{4}{3} y_{n-1} - \tfrac{1}{3} y_{n-2} + \tfrac{2}{3} h f(t_n, y_n)$$

- Since BDF is a multistep method you'll need to 'prime' the method; i.e., providing the solution for a number of steps before the method is self sufficient

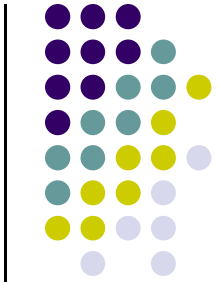
Exercise

[AO, Handout]



- Find the BDF that uses y_n, y_{n-1}, y_{n-2} in approximating the solution of t_n.
  - Hint: Take the following steps
    - Build polynomial p(t) that passes through y_n, y_{n-1}, y_{n-2}
    - Take time derivative of p(t) at t_n and set it to $f(t_n,y_n)$

BDF Methods: $\sum_{i=0}^{k} \alpha_i y_{n-i} = h\beta_0 f(t_n, y_n)$



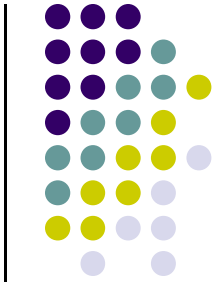
- Table below provides convergence order p, the number of steps k of the M method, the coefficients β_0, and the values of the coefficients α_0, α_1,…

p	k	β_0	α_0	α_1	α_2	α_3	α_4	α_5	α_6
1	1	1	1	-1					
2	2	2/3	1	-4/3	1/3				
3	3	6/11	1	-18/11	9/11	-2/11			
4	4	12/25	1	-48/25	36/25	-16/25	3/25		
5	5	60/137	1	-300/137	300/137	-200/137	75/137	-12/137	
6	6	60/147	1	-360/147	450/147	-400/147	225/147	-72/147	10/147

- Example: based on the table above, the second order BDF formula (k=2) is

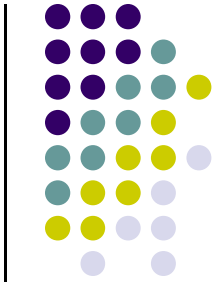
$$y_n - \frac{4}{3}y_{n-1} + \frac{1}{3}y_{n-2} = \frac{2}{3}hf(t_n, y_n) \quad \Rightarrow \quad y_n = \frac{4}{3}y_{n-1} - \frac{1}{3}y_{n-2} + \frac{2}{3}hf(t_n, y_n)$$

Exercise



- Plot the absolute stability regions for the BDF formulas up to order 6
- Comment on the size of the region of absolute stability

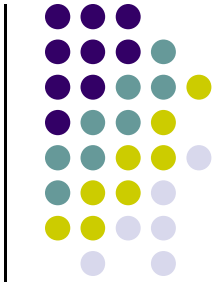
Exercise



- Prove that the BDF method with k=4 is convergent with order 4
- Approach:
  - Compute the roots of the characteristic equations to prove zero-stability
  - Verify that the order conditions are satisfied up to order 4
  - Use theorem that says that

    Zero-stability + Accuracy to order p $\Rightarrow$ Convergence of order p

Exercise

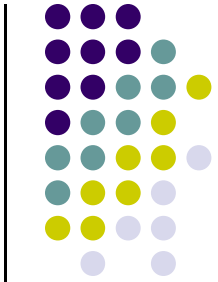


- Generate the convergence plot for the BDF method of order 6 for the following IVP:

$$\begin{cases} \dot{y} &= -5ty + \frac{5}{t} - \frac{1}{t^2} \\ y(1) &= 1 \end{cases}$$

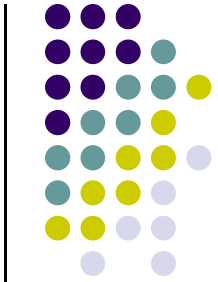
- Use the analytical solution, that is, y(t)=1/t, t $\in$ [1,4] to generate the starting points (history) required by the integration formula
  - Note that in practice you can't count on this break for the starting points, so you will have to use RK methods or gradually increase the order of the M method used to build the required history

BDF Method: Implementation Details (Newton Iteration)



- Recall that the BDF method is an implicit method
- Requires at each time step the solution of a nonlinear system of equations
- Approaches discussed so far for implicit multistep methods when it comes to solving the nonlinear discretization system:
  - Functional Iteration
  - PECE schemes (which can still be regarded as a flavor of Functional Iteration)
- Note that these two approaches only work for nonstiff IVPs, unless one is open to the idea of having very small step-sizes (which defeats the purpose of implicit integration…)

BDF Method: Implementation Details (Newton Iteration)



- The approach adopted for stiff problems is to solve the discretization nonlinear system by using Newton-Raphson or some variant
- Recall the nonlinear algebraic problem that we have to solve at each time step t_n:

$$\sum_{i=0}^{k} \alpha_i \mathbf{y}_{n-i} = h\beta_0 \mathbf{f}(t_n, \mathbf{y}_n)$$

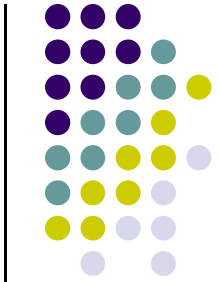
- It boils down to solving the following system of nonlinear equations:

$$\mathbf{g}(\mathbf{y}_n) \equiv \mathbf{y}_n - h\beta_0 \mathbf{f}(t_n, \mathbf{y}_n) + \mathbf{c}_n^{\mathbf{y}}(l) = \mathbf{0}$$

- Note that $\mathbf{c}_n^{\mathbf{y}}(l)$ is a constant quantity that only depends on previous values of the unknown function y (l stands for the order of the BDF):

$$\mathbf{c}_n^{\mathbf{y}}(l) = \sum_{i=1}^{k} \alpha_i \mathbf{y}_{n-i}$$

BDF Method: Implementation Details (Newton Iteration)



- The Newton-Raphson iteration assumes the expression:

$$\begin{cases} \left(\mathbf{I} - h\beta_0 \frac{\partial \mathbf{f}}{\partial \mathbf{y}}(t_n, \mathbf{y}_n^{(\nu)})\right) \Delta \mathbf{y}_n^{(\nu)} = -\mathbf{g}(t_n, \mathbf{y}_n^{(\nu)}) \\ \\ \mathbf{y}_n^{(\nu+1)} = \mathbf{y}_n^{(\nu)} + \Delta \mathbf{y}_n^{(\nu)} \end{cases}$$

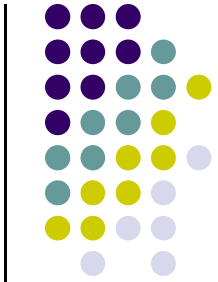
- The starting point is usually chosen like

$$\mathbf{y}_n^{(0)} = \mathbf{y}_{n-1}$$

- In practice, a modified Newton method is used since in the classical Newton-Raphson algorithm
  - Computing the Jacobian $\frac{\partial \mathbf{f}}{\partial \mathbf{y}}(t_n, \mathbf{y}_n^{(\nu)})$ at each iteration is expensive
  - Computing at each iteration the $\mathbf{LU}$ factorization of the iteration matrix $\mathbf{\Psi} \equiv \mathbf{I} - h\beta_0 \frac{\partial \mathbf{f}}{\partial \mathbf{y}}(t_n, \mathbf{y}_n^{(\nu)})$ is expensive

BDF Method: Implementation Details

The Modified Newton step

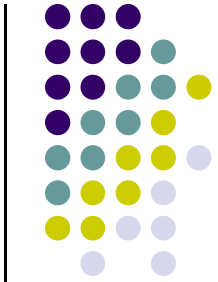


- The modified-Newton assumes the form (note the (0) superscript):

$$\begin{cases} \left(\mathbf{I} - h\beta_0 \frac{\partial \mathbf{f}}{\partial \mathbf{y}}(t_n, \mathbf{y}_n^{(0)})\right) \Delta \mathbf{y}_n^{(\nu)} = -\mathbf{g}(t_n, \mathbf{y}_n^{(\nu)}) \\ \\ \mathbf{y}_n^{(\nu+1)} = \mathbf{y}_n^{(\nu)} + \Delta \mathbf{y}_n^{(\nu)} \end{cases}$$

- In other words, the iteration matrix is evaluated once at the beginning of the step based on the predicted value $\mathbf{y}_n^{(0)}$
- The coefficient matrix is factored and subsequently used for all the iterations taken during that step
- You won't win the 'speed' category for your simEngine unless you implement this modified Newton approach
- This is the approach used in ADAMS

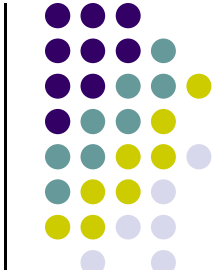
Loose Ends



- We have not discusses about the following aspects of M methods
  - Local error estimation
  - Step size selection
  - Order selection
  - The three strategy for changing the step size
    - Fixed-coefficient strategy
    - Variable-coefficient strategy
    - Fixed leading-coefficient strategy
- These aspects discussed in Ascher-Petzold book

[New Topic]

Handling 2nd Order IVP



- Example:

$$\begin{cases} m\ddot{x} + c\dot{x}^3 + kx^3 = sin(2t) \\ x(0) = x_0 \quad \rightarrow \quad \text{given to you} \\ \dot{x}(0) = v_0 \quad \rightarrow \quad \text{given to you} \end{cases}$$

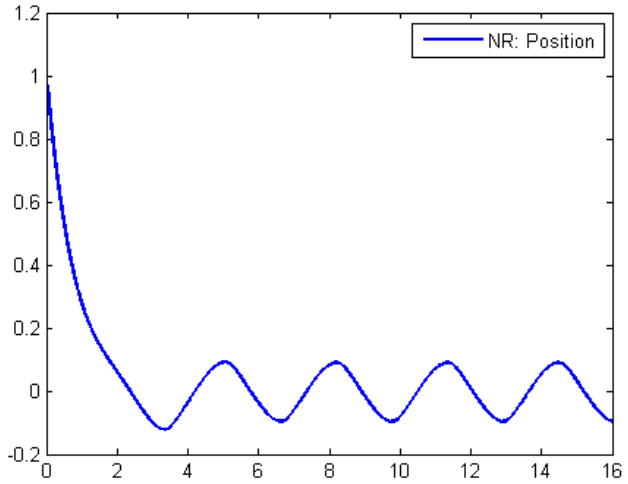
- Remarks, assumptions, notation used:
  - EOM for a mass-spring-damper system, see ME340 for derivation of EOM.
  - m, c, k - mass, damping coefficient, spring constant, respectively
  - Spring is nonlinear, so is damping (if they were linear there was no need to Newton method to solve the ensuing problem)
  - A time periodic force, $sin(2t)$, acts on the mass m
- We are in the business of finding approximations for x and $\dot{x}$, or x and v, given the model (through the m, c, k parameters) and the force acting on the mass
  - In other words, we need to find the position and velocity of the body as a function of time t
- We assume that c is large, which leads to a damped problem – you should use an implicit integrator to efficiently find the solution of this IVP

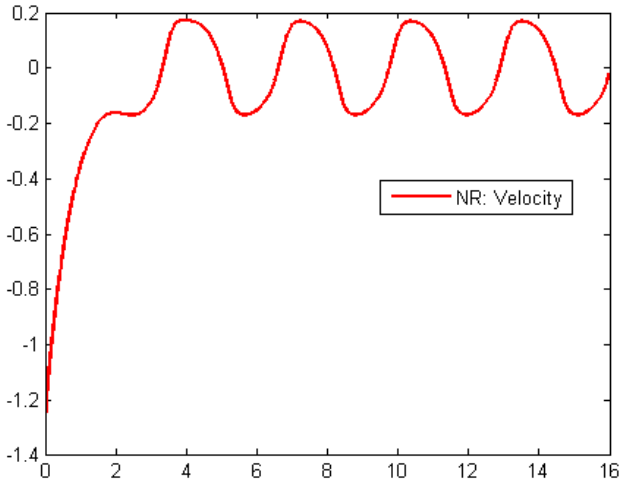
Outcome, Dynamics Analysis

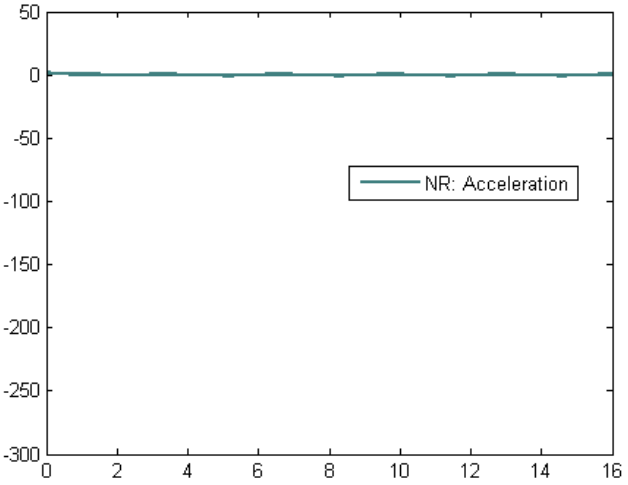
[Nonlinear Mass-Spring-Damper]

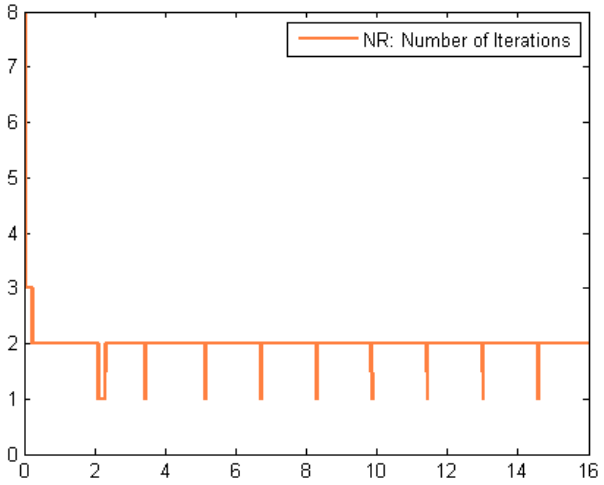
Model Params.
$m = 2$
$c = 200$
$k = 400.$

ICs:
$x_0 = 1$ and $v_0 = -1.$

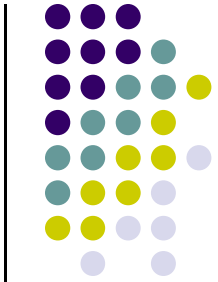








Keeping $\dot{\mathbf{y}}$ (instead of $\mathbf{f}(t, \mathbf{y})$) into the Picture



- Two ways to solve this IVP, they are equivalent yet lead to different perspectives on solving the same problem:
  - Reduce 2nd order IVP to a first order IVP of dimension two and apply your favorite implicit integration formula (say BDF)
  - Keep the IVP as is, and make a simple change to your favorite implicit integration formula (we'll work with the second order BDF to introduce this approach)
- Second order BDF:

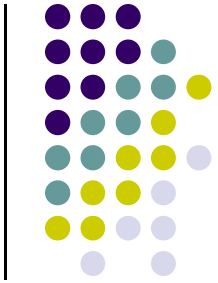
$$y_n = \frac{4}{3}y_{n-1} - \frac{1}{3}y_{n-2} + \frac{2}{3}hf(t_n, y_n)$$

- Equivalently (small but important different way of looking at the same thing),

$$y_n = \frac{4}{3}y_{n-1} - \frac{1}{3}y_{n-2} + \frac{2}{3}h\dot{y}_n$$

- Recall that we need to find the position and velocity of the mass. To this end, apply the generic BDF formula above to get through numerical integration the velocity v_n first, and then the position x_n of the mass at time step t_n:

Expressing the Position and Velocity as Functions of Acceleration



- For velocity:

$$v_n = \frac{4}{3}v_{n-1} - \frac{1}{3}v_{n-2} + \frac{2}{3}h\dot{v}_n$$

- That is,

$$v_n = \frac{4}{3}v_{n-1} - \frac{1}{3}v_{n-2} + \frac{2}{3}ha_n$$

- Handling the position x_n now:

$$x_n = \frac{4}{3}x_{n-1} - \frac{1}{3}x_{n-2} + \frac{2}{3}h\dot{x}_n$$

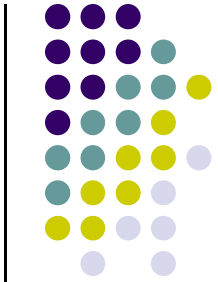
- That is,

$$x_n = \frac{4}{3}x_{n-1} - \frac{1}{3}x_{n-2} + \frac{2}{3}hv_n$$

- Based on the expression of v_n above, it follows that

$$x_n = \frac{4}{3}x_{n-1} - \frac{1}{3}x_{n-2} + \frac{2}{3}h(\frac{4}{3}v_{n-1} - \frac{1}{3}v_{n-2} + \frac{2}{3}ha_n) = \frac{4}{3}x_{n-1} - \frac{1}{3}x_{n-2} + \frac{8}{9}hv_{n-1} - \frac{2}{9}hv_{n-2} + \frac{4}{9}h^2a_n$$

Separating the Terms: Known vs. Unknown



- Important observation: take a look at the expression of the BDF integration formulas. No matter what BDF formula you use, the expression of x_n and v_n has two parts: one that depends on previous data (computed at t_{n-1}, t_{n-2}, t_{n-3}, t_{n-4}, etc. - in blue below), and one that depends on data that you don't know yet, but are about to compute at t_n (in red below)

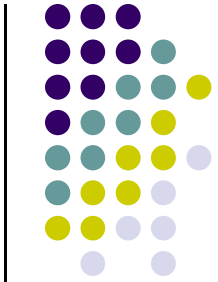
$$x_n = \frac{4}{3}x_{n-1} - \frac{1}{3}x_{n-2} + \frac{8}{9}hv_{n-1} - \frac{2}{9}hv_{n-2} + \frac{4}{9}h^2 a_n$$

$$v_n = \frac{4}{3}v_{n-1} - \frac{1}{3}v_{n-2} + \frac{2}{3}ha_n$$

- For any BDF formula you use, say you use the one of order l, it is easy to see that x_n and v_n can be expressed as

$$\begin{aligned} x_n &= C_n^x(l) + \beta_0^2 h^2 a_n \\ v_n &= C_n^v(l) + \beta_0 h a_n \end{aligned}$$

Separating the Terms: The Known Terms



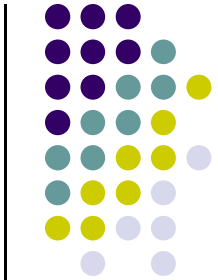
- Nomenclature:
  - The C in $C_n^x(l)$ is meant to suggest that $C_n^x(l)$ is quantity is a *constant*, which is evaluated based on values that were computed at previous time steps: t_{n-1}, t_{n-2}, t_{n-3}, t_{n-4}, etc.
  - The n in $C_n^x(l)$ is indicating that $C_n^x(l)$ is evaluated at time step t_n
  - The x in $C_n^x(l)$ is indicating that this constant $C_n^x(l)$ is the one associated with the position x. There is a constant term that is evaluated to enter the computation of v_n, like in $C_n^v(l)$
  - The l in $C_n^x(l)$ is indicating that $C_n^x(l)$ is as obtained for the BDF of order l. The higher the order, the more terms $C_n^x(l)$ and $C_n^v(l)$ will contain.
  - HOMEWORK: We just saw how to determine $C_n^x(2)$ and $C_n^v(2)$. Determine $C_n^x(1)$ and $C_n^v(1)$, as well as $C_n^x(3)$ and $C_n^v(3)$.

- **NOTE**: The relationships between position and acceleration, and between velocity and acceleration provided on the previous slide are **very important**. We will use it again when we solve the dynamics problem in the $\mathbf{r}-\mathbf{p}$ formulation, and here's how:

$$\mathbf{r}_n = \mathbf{C}_n^{\mathbf{r}}(l) + \beta_0^2 h^2 \ddot{\mathbf{r}}_n \qquad \dot{\mathbf{r}}_n = \mathbf{C}_n^{\dot{\mathbf{r}}}(l) + \beta_0 h \ddot{\mathbf{r}}_n$$

$$\mathbf{p}_n = \mathbf{C}_n^{\mathbf{p}}(l) + \beta_0^2 h^2 \ddot{\mathbf{p}}_n \qquad \dot{\mathbf{p}}_n = \mathbf{C}_n^{\dot{\mathbf{p}}}(l) + \beta_0 h \ddot{\mathbf{p}}_n$$

The Nonlinear System



- Recall the important expressions we derived on the previous slide:

$$x_n = C_n^x(l) + \beta_0^2 h^2 a_n$$
$$v_n = C_n^v(l) + \beta_0 h a_n$$

- Recall the expression of the EOM, discretized at time t_n (discretized here means that you take the continuum ODE problem and focus on the discrete form it assumes at time t_n):

$$m\ddot{x}_n + c\dot{x}_n^3 + kx_n^3 = sin(2t_n)$$

- Equivalently,

$$ma_n + cv_n^3 + kx_n^3 = sin(2t_n)$$

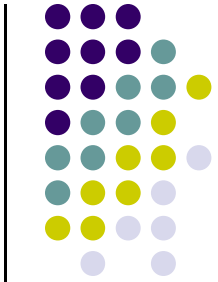
- Recall now that actually both v_n and x_n depend on a_n, according to our important expressions. With this in mind, define the following function g that only depends on a_n:

$$g(a_n) = ma_n + cv_n^3 + kx_n^3 - sin(2t_n)$$

- What we want to find is the root of $g(a_n)$; i.e., the solution of the equation

$$g(a_n) = 0$$

Computing Sensitivities



x_n with respect to a_n

v_n with respect to a_n

- The key thing to keep in mind is this: when attempting to find the root of the function g, although you see x_n and v_n in its expression, recall that they both depend on a_n as indicated by the BDF formulas (our two important relations)
- Since x_n and v_n depend on a_n, in the process of searching for the root of g we will need the sensitivities of x_n and v_n with respect to a_n. Here they are:

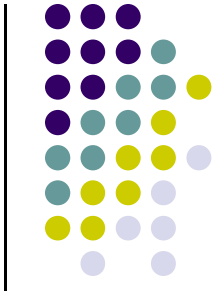
$$\frac{\partial x_n}{\partial a_n} = \beta_0^2 h^2 \qquad \frac{\partial v_n}{\partial a_n} = \beta_0 h$$

- Being at this, recall that we'll use the same approach when carrying out Dynamics Analysis for multibody systems. The sensitivities there are computed as (see previous slide)

$$\frac{\partial \mathbf{r}_n}{\partial \ddot{\mathbf{r}}_n} = \beta_0^2 h^2 \, \mathbf{I}_{3nb\times 3nb} \qquad \frac{\partial \dot{\mathbf{r}}_n}{\partial \ddot{\mathbf{r}}_n} = \beta_0 h \, \mathbf{I}_{3nb\times 3nb}$$

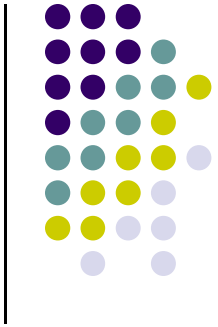
$$\frac{\partial \mathbf{p}_n}{\partial \ddot{\mathbf{p}}_n} = \beta_0^2 h^2 \, \mathbf{I}_{4nb\times 4nb} \qquad \frac{\partial \dot{\mathbf{p}}_n}{\partial \ddot{\mathbf{p}}_n} = \beta_0 h \, \mathbf{I}_{4nb\times 4nb}$$

Newton-Type Methods: Three Flavors



- We'll use variations of the same theme when it comes to finding the root of the function $g(a_n)$. The theme is *using Newton's method*, and the variations are as follows:
  - Straight Newton-Raphson
  - Modified Newton
  - Quasi-Newton
- Differences between the three approaches:
  - Straight Newton-Raphson updates the iteration matrix Ψ at each iteration: $\Psi \equiv \frac{\partial g}{\partial a_n}$. We discussed this before.
  - Modified Newton updates the iteration matrix Ψ only once in a while, typically at onset of the iterative process (we'll assume that this is the case in what follows, that is, that Ψ is evaluated at $a_n^{(0)}$). We discussed this before.
  - Quasi-Newton actually works only with a rough approximation of $\frac{\partial g}{\partial a_n}$ when searching for the root of $g(a_n)$. We didn't discuss this before.
- There is a subtle difference between Modified Newton and Quasi-Newton
  - In Modified Newton, you correctly and fully evaluate $\frac{\partial g}{\partial a_n}$ for $a_n^{(0)}$ (at the beginning of the iterative process). This *exact-when-computed* Jacobian is subsequently recycled a number of times in the iterative process
  - In Quasi-Newton, you **approximate** $\frac{\partial g}{\partial a_n}$. In other words, because actually computing $\frac{\partial g}{\partial a_n}$ is expensive, you give up on some computationally expensive terms that enter its expression to obtain only an approximation of $\frac{\partial g}{\partial a_n}$

Newton-Type Methods:
[Geometric Interpretation]



- Straight Newton-Raphson: when moving towards the root of the function, at each iteration you search in the direction given by the current tangent

- Modified Newton: when moving towards the root of the function, at each iteration you search in the direction given by the tangent evaluated at the point where you started the iterative process

- Quasi-Newton: when moving towards the root of the function, at each iteration you search in some direction that is not given by any tangent to the function. Yet, it is a direction that you computed cheaply and hopefully is not very different than the direction of the actual tangent