Numerical Integration in Structural Dynamics

CEE 541. Structural Dynamics
Department of Civil & Environmental Engineering
Duke University

Henri P. Gavin
Fall 2018

Introduction

A damped structural system subjected to dynamic forces and possibly experiencing nonlinear material behavior is modeled by

$$M\ddot{x}(t) + C\dot{x}(t) + Kx(t) + R(x(t), \dot{x}(t)) = f^{\text{ext}}(t), \tag{1}$$

where x is a vector of displacements of structural coordinates, M is a positive definite mass matrix, C is a non-negative definite damping matrix, and K is a non-negative definite stiffness matrix. The nonlinear restoring forces are given in $R(x, \dot{x})$ and $f^{\text{ext}}(t)$ is a vector of external dynamic loads. At any point in time, $t = t_{i+1} = (i+1)h$, we may solve for the accelerations in terms of the displacements, velocities, and the applied forces.

$$\ddot{x}(t_{i+1}) = -M^{-1}[C\dot{x}(t_{i+1}) + Kx(t_{i+1}) + R(x(t_{i+1}), \dot{x}(t_{i+1})) - f^{\text{ext}}(t_{i+1})]. \tag{2}$$

Given values for accelerations, velocities, displacements, and applied forces at time t_i, if we can extrapolate the velocities and displacements forward in time by a time step h to time $t_{i+1} = t_i + h$, then we may compute the acceleration $\ddot{x}(t_{i+1})$ using equation 2. The various numerical integration algorithms described in this document[1] [2] differ primarily in the manner in which $x(t_{i+1})$ and $\dot{x}(t_{i+1})$ are computed from $x(t_i)$, $\dot{x}(t_i)$, $\ddot{x}(t_i)$, $f^{\text{ext}}(t_i)$, and $f^{\text{ext}}(t_{i+1})$.

There are two general classifications of numerical integration methods: *explicit* and *implicit*. In explicit methods, displacements and velocities at t_{i+1} can be determined in closed form from displacements, velocities, accelerations, at t_i, and from external forcing at t_i and, potentially, t_{i+1}. For structural systems with linear elastic stiffness and linear viscous damping, such *discrete-time systems* may be written

$$\begin{bmatrix} x(t_{i+1}) \\ \dot{x}(t_{i+1}) \end{bmatrix} = A \begin{bmatrix} x(t_i) \\ \dot{x}(t_i) \end{bmatrix} + B\ f^{\text{ext}}(t_i) \tag{3}$$

where A is a $2n \times 2n$ *discrete time dynamics* matrix which depends upon M, C, K, the time step, h, and some algorithmic parameters. Implicit methods involve the solution of a set of nonlinear algebraic equations at each time step. For example, the displacements and velocities at time t_{i+1}, $[x(t_{i+1}), \dot{x}(t_{i+1})]$, are determined from the roots of a nonlinear equation in terms of $[x(t_{i+1}), \dot{x}(t_{i+1})]$. Explicit numerical methods are typically more computationally-efficient than implicit methods.

[1] Clough, R. and Penzien, J., *Dynamics of Structures,* 2nd ed., McGraw-Hill, 1993.
[2] Hanson, R.D., CE 611: Structural Dynamics, class notes, The University of Michigan, 1988

Accuracy and Stability

Numerical methods for integrating equations of motion are assessed and evaluated in terms of their accuracy and stability. In general, accuracy and stability depend upon the ratio of the time step, h, to the shortest natural period in the system model. For a system with many coordinates ($n > 10^3$), the shortest natural period can be much shorter than the fundamental natural period, $T_n/T_1 > 10^4$. Typically, responses of the highest several modes of a numerical model are physically meaningless, should be insignificantly small, but are potentially lightly-damped, and can dominate the errors in numerical integration. The explicit numerical methods described in these notes can artificially add *numerical damping* to suppress instabilities of the higher mode responses. Implicit numerical integration methods are unconditionally stable.

The Central Difference Method

The central difference approximations for the first and second derivatives are

$$\dot{x}(t_i) = \dot{x}_i \approx \frac{1}{2h}\left(x_{i+1} - x_{i-1}\right), \tag{4}$$

$$\ddot{x}(t_i) = \ddot{x}_i \approx \frac{1}{h^2}\left(x_{i+1} - 2x_i + x_{i-1}\right). \tag{5}$$

Substituting approximations (4) and (5) into equation (1), and re-arranging terms, leads to

$$\left[\frac{1}{h^2}M + \frac{1}{2h}C\right] x_{i+1} = \left[\frac{2}{h^2}M - K\right] x_i + \left[-\frac{1}{h^2}M + \frac{1}{2h}C\right] x_{i-1} + f_i^{\text{ext}}, \tag{6}$$

which may be written $A_0 x_{i+1} = A_1 x_i + A_2 x_{i-1} + f_i^{\text{ext}}$. The matrix A_0 is positive definite and may be factorized prior to the iterative solution for x_i at each time step.

The numerical stability of the central difference method depends on the choice of the time step interval, h. To obtain a stable solution, $h < T_i/\pi$, where T_i is the shortest natural period of the structural system. If the central difference method is used with a time step larger than T_i/π the solution will increase exponentially.

A step-by-step procedure for the central difference method may be written as[3]:

1. Input the mass, M, damping, C, stiffness, K, matrices and the time step interval h.
2. Initialize x_0, $\dot{x}_0$, $\ddot{x}_0$ and compute $x_1 = x_0 - h\dot{x}_0 + \frac{h^2}{2}\ddot{x}_0$.
3. Form the matrices A_0, A_1 and A_2, and triangularize A_0 using LDL^T factorization.
4. for $i = 1$ to N
   (a) solve $A_0 x_{i+1} = A_1 x_i + A_2 x_{i-1} + f_i^{\text{ext}}$ for x_{i+1} using LDL^T back-substitution.
   (b) calculate $\dot{x}_i$ and $\ddot{x}_i$ using equations (4) and (5), if required.
   (c) write results (t_i, x_i, $\dot{x}_i$, $\ddot{x}_i$) to a data file.

[3] K.J. Bathe, *Finite Element Procedures in Engineering Analysis,* Prentice-Hall, 1982, pp. 439–449, 499–506.

The Implicit Linear Acceleration Method

Consider the Taylor series expansions for displacement, velocity, and acceleration:

$$x(t_{i+1}) = x_{i+1} = x_i + h\dot{x}_i + \frac{h^2}{2!}\ddot{x}_i + \frac{h^3}{3!}\dddot{x}_i + \frac{h^4}{4!}\ddddot{x}_i + \cdots \tag{7}$$

$$\dot{x}(t_{i+1}) = \dot{x}_{i+1} = \dot{x}_i + h\ddot{x}_i + \frac{h^2}{2!}\dddot{x}_i + \frac{h^3}{3!}\ddddot{x}_i + \cdots \tag{8}$$

$$\ddot{x}(t_{i+1}) = \ddot{x}_{i+1} = \ddot{x}_i + h\dddot{x}_i + \frac{h^2}{2!}\ddddot{x}_i + \cdots \tag{9}$$

Rearranging equation (9),

$$h\dddot{x}_i = \ddot{x}_{i+1} - \ddot{x}_i - \frac{h^2}{2!}\ddddot{x}_i - \cdots \ , \tag{10}$$

and substituting equation (10) into equations (7) and (8) results in

$$x_{i+1} = x_i + h\dot{x}_i + \frac{h^2}{2}\ddot{x}_i + \frac{h^2}{6}\left(\ddot{x}_{i+1} - \ddot{x}_i - \frac{h^2}{2!}\ddddot{x}_i - \cdots\right) + \frac{h^4}{4!}\ddddot{x}_i + \cdots \ , \tag{11}$$

$$\dot{x}_{i+1} = \dot{x}_i + h\ddot{x}_i + \frac{h}{2}\left(\ddot{x}_{i+1} - \ddot{x}_i - \frac{h^2}{2!}\ddddot{x}_i - \cdots\right) + \frac{h^3}{3!}\ddddot{x}_i + \cdots \tag{12}$$

Truncating the fourth time-derivative and higher from these expansions, the resulting finite difference approximations are

$$x_{i+1} \approx x_i + h\dot{x}_i + \frac{h^2}{6}(\ddot{x}_{i+1} + 2\ddot{x}_i), \tag{13}$$

and

$$\dot{x}_{i+1} \approx \dot{x}_i + \frac{h}{2}(\ddot{x}_{i+1} + \ddot{x}_i). \tag{14}$$

These relationships are *implicit* because $\ddot{x}_{i+1}$ needs to be determined in order to find x_{i+1} and $\dot{x}_{i+1}$, but $\ddot{x}_{i+1}$ can not be found without knowing x_{i+1} and $\dot{x}_{i+1}$. Note that the substitutions above, have eliminated the third time-derivative of x, and that the method is accurate to within $h^4\ddddot{x}$.

This is called the *linear acceleration method* because the third time derivative of x has been eliminated. If the rate of change of acceleration within a time-step is truly constant, then the approximation of truncating the Taylor series at the fourth-order term does not affect the accuracy of the solution.

The Implicit Linear Acceleration Method, Made Explicit for Linear Structural Dynamics

Recall that at time t_{i+1} we can satisfy the equations of motion, by calculating the acceleration with equation (2). Substituting equations (13) and (14) for the displacements and velocities at time t_{i+1} into equation (2),

$$M\ddot{x}_{i+1} + C\{\dot{x}_i + (h/2)(\ddot{x}_i + \ddot{x}_{i+1})\} + K\{x_i + h\dot{x}_i + (h^2/6)(\ddot{x}_{i+1} + 2\ddot{x}_i)\} = f^{\text{ext}}_{i+1} \ , \tag{15}$$

where the nonlinearities, $R(x, \dot{x})$, are negligible. Collecting similar derivatives of x,

$$\left[M + \frac{h}{2}C + \frac{h^2}{6}K\right]\ddot{x}_{i+1} = f^{\text{ext}}_{i+1} - Kx_i - [C + hK]\dot{x}_i - \left[\frac{h}{2}C + \frac{h^2}{3}K\right]\ddot{x}_i \ . \tag{16}$$

and re-arranging,

$$\left[M + \frac{h}{2}C + \frac{h^2}{6}K\right]\ddot{x}_{i+1} = \left[-\frac{h}{2}C - \frac{h^2}{3}K\right]\ddot{x}_i - Kx_i - C\dot{x}_i - hK\dot{x}_i + f^{\text{ext}}_{i+1} \ . \tag{17}$$

Recall from the equations of motion, that

$$M\ddot{x}_i - f^{\text{ext}}_i = -Kx_i - C\dot{x}_i \ . \tag{18}$$

Substituting equation (18) into (17), we obtain the closed-form linear acceleration recurrence relations for structural dynamics simulation.

$$\left[M + \frac{h}{2}C + \frac{h^2}{6}K\right]\ddot{x}_{i+1} = \left[M - \frac{h}{2}C - \frac{h^2}{3}K\right]\ddot{x}_i - hK\dot{x}_i + f^{\text{ext}}_{i+1} - f^{\text{ext}}_i \ . \tag{19}$$

$$\dot{x}_{i+1} = \dot{x}_i + \frac{h}{2}\left[\ddot{x}_{i+1} + \ddot{x}_i\right]. \tag{20}$$

and

$$x_{i+1} = x_i + h\dot{x}_i + \frac{h^2}{6}\left[\ddot{x}_{i+1} + 2\ddot{x}_i\right]. \tag{21}$$

These relationships are now *explicit* because $\ddot{x}_{i+1}$ can be determined from the current response values (x_i and $\dot{x}_i$), the current dynamic load f^{ext}_i, and the next dynamic load f^{ext}_{i+1}. Note that within each time step, the dynamic equations of equilibrium are satisfied both at time t_i and at time t_{i+1}.

The Newmark-β method — incremental formulation

The finite difference approximations for the Newmark-β method are

$$x_{i+1} \approx x_i + h\dot{x}_i + h^2\left[\left(\frac{1}{2}-\beta\right)\ddot{x}_i + \beta\ddot{x}_{i+1}\right], \tag{22}$$

and

$$\dot{x}_{i+1} \approx \dot{x}_i + h\left[(1-\gamma)\ddot{x}_i + \gamma\ddot{x}_{i+1}\right]. \tag{23}$$

If $\beta = 1/4$ and $\gamma = 1/2$ the Newmark-β method is implicit and unconditionally stable. In this case the acceleration within the time interval $t \in [t_i\ t_{i+1})$ is presumed to be constant. If $\beta = 1/6$ and $\gamma = 1/2$ the Newmark-β method is identical to the linear acceleration method. If $\beta = 0$ and $\gamma = 1/2$ the Newmark-β method is identical to the central difference method. For linear structural dynamics, if $2\beta \geq \gamma \geq 1/2$, then the Newmark-$\beta$ method is stable regardless of the size of the time-step, h. The Newmark-β method is conditionally stable if $\gamma < 1/2$. For $\gamma = 1/2$ the Newmark-β method is at least second-order accurate; it is first order accurate for all other values of γ.

Formulating the finite difference relationships in terms of the increments of displacement ($\delta x_i = x_{i+1} - x_i$), velocity ($\delta\dot{x}_i = \dot{x}_{i+1} - \dot{x}_i$), acceleration ($\delta\ddot{x}_i = \ddot{x}_{i+1} - \ddot{x}_i$), and nonlinear restoring force ($\delta R_i = R(x_{i+1}, \dot{x}_{i+1}) - R(x_i, \dot{x}_i)$)

$$\delta\ddot{x}_i = \frac{1}{\beta h^2}\delta x_i - \frac{1}{\beta h}\dot{x}_i - \frac{1}{2\beta}\ddot{x}_i, \tag{24}$$

and

$$\delta\dot{x}_i = \frac{\gamma}{\beta h}\delta x_i - \frac{\gamma}{\beta}\dot{x}_i + h\left(1 - \frac{\gamma}{2\beta}\right)\ddot{x}_i. \tag{25}$$

Now, satisfying incremental equilibrium over the time step, h,

$$M\delta\ddot{x}_i + C\delta\dot{x}_i + K\delta x_i + \delta R_i = f_{i+1}^{\text{ext}} - f_i^{\text{ext}} = \delta f_i^{\text{ext}}, \tag{26}$$

re-grouping terms, and solving for the increment in displacements,

$$\left[\frac{6}{h^2}M + \frac{3}{h}C + K\right]\delta x_i = \delta f_i^{\text{ext}} - \delta R_i + \left[3M + \frac{h}{2}C\right]\ddot{x}_i + \left[\frac{6}{h}M + 3C\right]\dot{x}_i, \tag{27}$$

where $\beta = 1/6$ and $\gamma = 1/2$. Solving this linear system of equations[4] for δx_i, the displacements are updated with,

$$x_{i+1} = x_i + \delta x_i, \tag{28}$$

the velocities are updated with

$$\dot{x}_{i+1} = -2\dot{x}_i - \frac{h}{2}\ddot{x}_i + \frac{3}{h}\delta x_i, \tag{29}$$

and the accelerations satisfy the equations of motion,

$$\ddot{x}_{i+1} = -M^{-1}[C\dot{x}_{i+1} + Kx_{i+1} + R(x_{i+1}, \dot{x}_{i+1}) - f_{i+1}^{\text{ext}}]. \tag{30}$$

Similar relations may be found for other values of β and γ. Note that in this incremental formulation one needs to obtain the inverse of the mass matrix, M, and the inverse of the matrix $[6M/h^2 + 3C/h + K]$.

[4] If the incremental nonlinear restoring forces δR_i are not negligible, equation (27) is solved using the Newton-Raphson method, equation (48).

The HHT-α method

The HHT-α method[5] is a generalization of the Newmark-β method and reduces to the Newmark-β method for $\alpha = 0$. The HHT-α method adopts the finite difference equations of the Newmark-β method, (equations (22) and (23)). The equations of motion are modified, however, using a parameter α, which represents a numerical lag in the damping, stiffness, nonlinear, and external forces.

$$M\ddot{x}_{i+1} + (1-\alpha)C\dot{x}_{i+1} + \alpha C\dot{x}_i + (1-\alpha)Kx_{i+1} + \alpha Kx_i = (1-\alpha)f^{\text{ext}}_{i+1} + \alpha f^{\text{ext}}_i. \tag{31}$$

Equation (31) is analogous to equation (26) with the added condition that the acceleration is constant within the interval $t \in [t_i \; t_{i+1})$. If

$$\begin{aligned} 0 \;\le\; \alpha \;&\le\; 1/3 && (32)\\ \beta \;&=\; (1+\alpha)^2/4\,, \quad \text{and} && (33)\\ \gamma \;&=\; 1/2+\alpha\,, && (34) \end{aligned}$$

the HHT-α method is at least second-order accurate and unconditionally stable. The HHT-α method is useful in structural dynamics simulations incorporating many degrees of freedom, and in which it is desirable to numerically attenuate (or dampen-out) the response at high frequencies. Increasing α decreases the response at frequencies above $1/(2h)$, provided that β and γ are defined as above.

Substituting the Newmark-β finite difference relationships into equation (31),

$$\begin{aligned} M\ddot{x}_{i+1} \;&+\; (1-\alpha)C\{\dot{x}_i + h[(1-\gamma)\ddot{x}_i + \gamma\ddot{x}_{i+1}]\} + \alpha C\dot{x}_i \\ &+\; (1-\alpha)K\{x_i + h\dot{x}_i + h^2[(1/2-\beta)\ddot{x}_i + \beta\ddot{x}_{i+1}]\} + \alpha Kx_i \\ &=\; (1-\alpha)f^{\text{ext}}_{i+1} + \alpha f^{\text{ext}}_i, \end{aligned} \tag{35}$$

and grouping terms, we obtain

$$\begin{aligned} [M + h(1-\alpha)\gamma C + h^2(1-\alpha)\beta K]\ddot{x}_{i+1} \;&+\; [h(1-\alpha)(1-\gamma)C + h^2(1-\alpha)(1/2-\beta)K]\ddot{x}_i \\ &+\; [C + h(1-\alpha)K]\dot{x}_i \\ &+\; Kx_i \\ &=\; (1-\alpha)f^{\text{ext}}_{i+1} + \alpha f^{\text{ext}}_i. \end{aligned} \tag{36}$$

These are a set of linear equations for $\ddot{x}_{i+1}$ in terms of $\ddot{x}_i$, $\dot{x}_i$, x_i, the external dynamic load, and the structure's mass, damping, and stiffness. Equations (22), (23), and (36) provide the recurrence relationship for the HHT-α method.

[5] Hughes, T.J.R., Analysis of Transient Algorithms with Particular Reference to Stability Behavior. in *Computational Methods for Transient Analysis,* North-Holland, 1983, pp. 67–155.

The HHT-α method — incremental formulation

The HHT-α method may be formulated in an incremental form, as was done with the Newmark-β method in the previous section. Again, formulating the finite difference relationships in terms of the increments of displacement ($\delta x_i = x_{i+1} - x_i$), velocity ($\delta \dot{x}_i = \dot{x}_{i+1} - \dot{x}_i$), acceleration ($\delta \ddot{x}_i = \ddot{x}_{i+1} - \ddot{x}_i$), and nonlinear restoring force, and difference between equation (31) (evaluated at time t_{i+1}) and the same expression evaluated at time t_i, results in

$$M\delta\ddot{x}_i + (1-\alpha)C\delta\dot{x}_i + \alpha C\delta\dot{x}_{i-1} + (1-\alpha)K\delta x_i + \alpha K\delta x_{i-1} + (1-\alpha)\delta R_i + \alpha\delta R_{i-1} = (1-\alpha)\delta f_i^{\text{ext}} + \alpha\delta f_{i-1}^{\text{ext}}. \tag{37}$$

Now, substituting in expressions for $\delta\dot{x}$ and $\delta\ddot{x}$ at times t_i and t_{i-1}, and grouping terms results in

$$\begin{aligned}
\left[\frac{1}{\beta h^2}M + (1-\alpha)\frac{\gamma}{\beta h}C + (1-\alpha)K\right]\delta x_i &= \left[\frac{1}{\beta h}M + (1-\alpha)\frac{\gamma}{\beta}C\right]\dot{x}_i \\
&+ \left[\frac{1}{2\beta}M - (1-\alpha)h\left(1-\frac{\gamma}{2\beta}\right)C\right]\ddot{x}_i \\
&- \alpha\left[\frac{\gamma}{\beta h}C + K\right]\delta x_{i-1} \\
&+ \alpha\frac{\gamma}{\beta}C\dot{x}_{i-1} - \alpha h\left(1-\frac{\gamma}{2\beta}\right)C\ddot{x}_{i-1} \\
&- (1-\alpha)\delta R_i - \alpha\delta R_{i-1} \\
&+ (1-\alpha)\delta f_i + \alpha\delta f_{i-1}\ .
\end{aligned} \tag{38}$$

If $R(x,\dot{x}) = 0$, these are a set of linear equations for δx_i in terms of $\dot{x}_i$, $\ddot{x}_i$, δx_{i-1}, $\dot{x}_{i-1}$, $\ddot{x}_{i-1}$, δx_{i-1}, the external dynamic load, and the structure's mass, damping, and stiffness. Equations (22), (23), and (38) provide the recurrence relationship for the HHT-α method in incremental form. Solving this linear system of equations[6] for δx_i, the displacements are updated with,

$$x_{i+1} = x_i + \delta x_i, \tag{39}$$

the velocities are updated with

$$\dot{x}_{i+1} = \left(1-\frac{\gamma}{\beta}\right)\dot{x}_i + \frac{\gamma}{\beta h}\delta x_i + h\left(1-\frac{\gamma}{2\beta}\right)\ddot{x}_i. \tag{40}$$

and the accelerations satisfy the HHT-α form of the equations of motion,

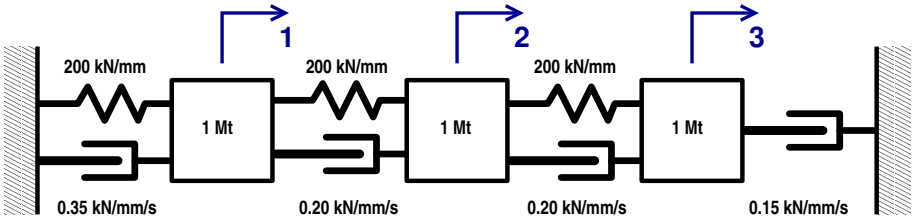
$$\ddot{x}_{i+1} = -M^{-1}\left[(1-\alpha)C\dot{x}_{i+1} + \alpha C\dot{x}_i + (1-\alpha)Kx_{i+1} + \alpha Kx_i - (1-\alpha)f_{i+1}^{\text{ext}} - \alpha f_i^{\text{ext}}\right]. \tag{41}$$

This algorithm is explicit *regardless* of the values of α, β, or γ.

[6] If the incremental nonlinear restoring forces δR_i are not negligible, equation (38) is solved using the Newton-Raphson method, equation (48).

Numerical Example

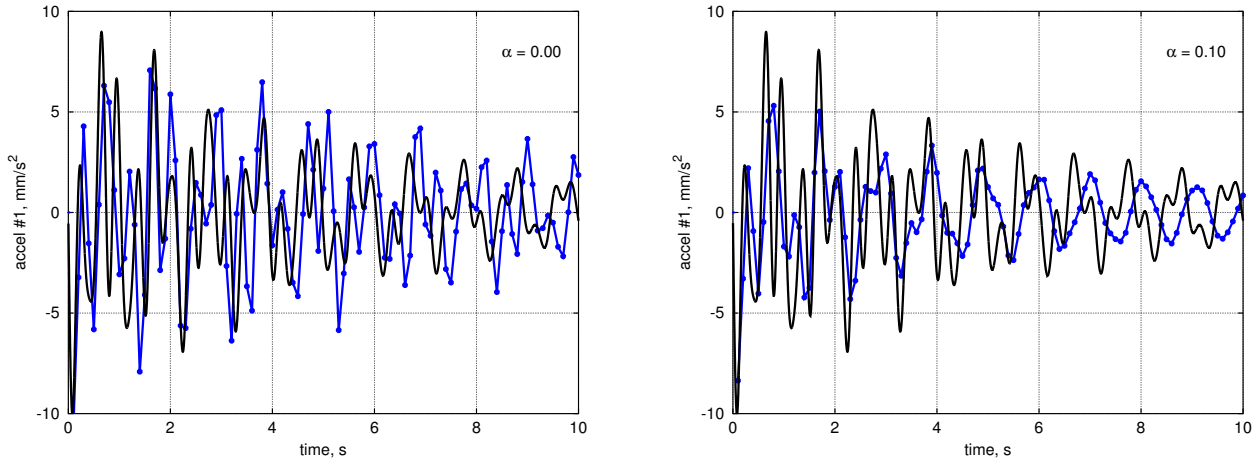
The simple linear structural model shown below



is described by the following stiffness, damping, and mass matrices (units kN,mm,s):

$$K = \begin{bmatrix} 400 & -200 & 0 \\ -200 & 400 & -200 \\ 0 & -200 & 200 \end{bmatrix} \quad C = \begin{bmatrix} 0.55 & -0.20 & 0 \\ -0.20 & 0.55 & -0.20 \\ 0 & -0.20 & 0.35 \end{bmatrix} \quad M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (42)$$

This system has natural periods of 1.00 s, 0.36 s, and 0.25 s, with about 1.5% critical damping in each mode. Free acceleration responses to initial conditions of $x(0) = [0, 0, 0]^T$ and $\dot{x}(0) = [1, 1, 1]^T$ using the HHT-α method with time steps of 0.001 s (black solid) and 0.1 s (blue dash-dot), and values of α of 0.0 (left) and 0.1 (right) are shown below.



For the $\alpha = 0$ cases, the response computed with $h = 0.1$ s has a magnified short-period response as compared to the $h = 0.001$ s case. For $\alpha = 0.1$ and $h = 0.1$ s, the response is more heavily damped especially the response at the shorter periods. Also, the responses simulated with $\alpha = 0.1$ and $h = 0.1$ s has a longer period than the system's true fundamental period of 1.0 s. Note that the linear acceleration method ($\alpha = 0, \beta = 1/6, \gamma = 1/2$) would be numerically unstable for this system with a time step of 0.1 s.

A more formal analysis of numerical damping, stability, and period lengthening effects in explicit numerical integration methods can be carried out by examining the eigenvalues of the discrete time dynamics matrix, A, in equation (3).

Simulation of SDOF Bilinear Hysteretic Behavior

Bilinear hysteretic behavior may be simulated using linear dynamic models. The basic idea is that each branch of the hysteresis may be described by an equation of the form $R = K(x - d)$, where the restoring force is zero when $x = d$ and K will take the value of the pre-yield stiffness or the strain hardening stiffness. The procedure is as follows:

1. Initialize K using the pre-yield stiffness K_{hi}. Initialize x_0, $\dot{x}_0$, $\ddot{x}_0$, f_0^{ext}, and d to be zero at time $t_0 = 0$. The value d is the equilibrium displacement of the bilinear hysteretic system.

2. At time $t = t_{i+1}$, read the value of the external forcing f_{i+1}^{ext}, solve equation (27) for δx_i, and calculate x_{i+1} and $\dot{x}_{i+1}$ using equations (28) and (29).

3. Compute the next acceleration $\ddot{x}_{i+1}$ from equilibrium
$$\ddot{x}_{i+1} = -M^{-1}[C\dot{x}_{i+1} + K(x_{i+1} - d) - f_{i+1}^{\text{ext}}]. \tag{43}$$

4. Compute the yielding force level from
$$f^{\text{yield}}(x_{i+1}, \dot{x}_{i+1}) = K_{\text{lo}} x_{i+1} + (K_{\text{hi}} - K_{\text{lo}})\ x_y\ \text{sgn}(\dot{x}_{i+1}) \tag{44}$$

5. Check for yielding:
   (a) If $K = K_{\text{hi}}$ and $\dot{x}_{i+1} > 0$ and $K(x_{i+1} - d) > f^{\text{yield}}$,
   then the system has just exceeded its positive yield force level.
   Set $K = K_{\text{lo}}$ and $d = (1 - K_{\text{hi}}/K_{\text{lo}})x_y$.
   (b) If $K = K_{\text{hi}}$ and $\dot{x}_{i+1} < 0$ and $K(x_{i+1} - d) < f^{\text{yield}}$,
   then the system has just exceeded it's negative yield force level.
   Set $K = K_{\text{lo}}$ and $d = (K_{\text{hi}}/K_{\text{lo}} - 1)x_y$.

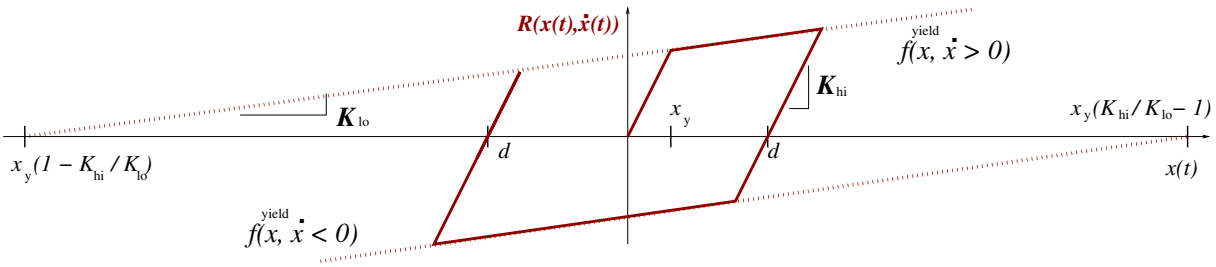
6. Check for load reversal:

   If $K = K_{\text{lo}}$ and $\dot{x}_{i+1}\dot{x}_i < 0$,
   Set $K = K_{\text{hi}}$ and $d = x_{i+1} - (K_{\text{lo}}/K_{\text{hi}})(x_{i+1} - d)$

7. If any of the checks in steps 5 or 6 were true, then correct the velocity and acceleration ($\dot{x}_{i+1}$, $\ddot{x}_{i+1}$) using equations (27), (29), and (43) with a very small time step ($h/10^4$), over which the external forcing is assumed to be constant ($\delta f_i^{\text{ext}} = 0$).

8. Record x_{i+1}, $\dot{x}_{i+1}$, $\ddot{x}_{i+1}$, and $f_{i+1} - M\ddot{x}_{i+1}$. Increment the time step. $i := i + 1$.

9. Go back to step 2 and repeat until the end of the time history.



Newton-Raphson Iterative Solution for Nonlinear Behavior

Equations (27) or (38) may be written $\tilde{K}\ \delta x_i = \tilde{f}_i$. If the nonlinear restoring force increment δR_i is not negligible, then $\tilde{f}$ depends on δx and $\delta\dot{x}$. For the case of the linear acceleration method ($\alpha = 0$, $\beta = 1/6$, $\gamma = 1/2$),

$$\tilde{K} = \left[\frac{6}{h^2}M + \frac{3}{h}C + K\right], \tag{45}$$

and

$$\tilde{f}_i(\delta x_i) = \delta f_i^{\text{ext}} - R(x_i + \delta x_i, \dot{x}_i + \delta\dot{x}_i) + R(x_i, \dot{x}_i) + \left[3M + \frac{h}{2}C\right]\ddot{x}_i + \left[\frac{6}{h}M + 3C\right]\dot{x}_i. \tag{46}$$

The Newton-Raphson algorithm[7] is an efficient method to solve the nonlinear equations $\tilde{K}\ \delta x_i = \tilde{f}_i(\delta x_i)$ for the incremental displacement δx_i. The algorithm is iterative and proceeds as follows:

1. The initial value for δx_i is denoted $\delta x_i^{(0)}$ and is arbitrarily set to zero. The corresponding value for $\tilde{f}_i$ is

$$\tilde{f}_i(\delta x_i^{(0)}) = \delta f_i^{\text{ext}} + \left[3M + \frac{h}{2}C\right]\ddot{x}_i + \left[\frac{6}{h}M + 3C\right]\dot{x}_i \tag{47}$$

2. The Newton-Raphson recurrence relation is simply

$$\delta x_i^{(n+1)} = \tilde{K}^{-1}\ \tilde{f}_i(\delta x_i^{(n)}) \tag{48}$$

where

$$\tilde{f}_i(\delta x_i^{(n)}) = \tilde{f}_i(\delta x_i^{(0)}) - R(x_i + \delta x_i^{(n)}, \dot{x}_i + \delta\dot{x}_i^{(n)}) + R(x_i, \dot{x}_i), \tag{49}$$

and

$$\delta\dot{x}_i^{(n)} = \frac{3}{h}\delta x_i^{(n)} - 3\dot{x}_i - \frac{h}{2}\ddot{x}_i. \tag{50}$$

3. Equation (48) is iterated upon until the incremental displacement δx_i converges, i.e., $||\delta x_i^{(n+1)} - \delta x_i^{(n)}|| < \epsilon$.

The convergence of this form of the Newton-Raphson method depends on the local smoothness of $R(x, \dot{x})$. Convergence can be improved, for a particular time step, by making h smaller.

Note that $\tilde{K}$ is strongly positive definite; it does not depend on δx_i; and it is inverted or factorized only once at the beginning of the simulation. If convergence is problematic, then a set of $\tilde{K}$ matrices may be formed, some with small time steps, and factorized prior to the simulation.

As an alternative to Newton-Raphson iterations, the approximation $\delta R_i \approx \delta R_{i-1}$ results in an explicit formulation for the simulation of the response of nonlinear systems, and can significantly reduce computational time, but with some loss of accuracy.

[7] Press, W.H., *et al.*, *Numerical Recipes,* Cambridge Univ. Press, 1993. http://www.nr.com/

State Variable Formulations

The second order ordinary differential equations of motion for a structural system are

$$M\ddot{r}(t) + C\dot{r}(t) + Kr(t) + R(r(t), \dot{r}(t)) = f^{\text{ext}}(t), \tag{51}$$

where we have re-named the displacement vector $x(t)$ of equation (1) with $r(t)$. This set of n second order differential equations may be written as a set of $2n$ first order differential equations, as follows:

$$\frac{d}{dt}\begin{bmatrix} r \\ \dot{r} \end{bmatrix} = \begin{bmatrix} 0_{N\times N} & I_{N\times N} \\ -M^{-1}K & -M^{-1}C \end{bmatrix}\begin{bmatrix} r \\ \dot{r} \end{bmatrix} + \begin{bmatrix} 0_{N\times 1} \\ -M^{-1}R(r,\dot{r}) \end{bmatrix} + \begin{bmatrix} 0_{N\times N} \\ M^{-1} \end{bmatrix} f^{\text{ext}}(t). \tag{52}$$

The vector $[r^T \;\; \dot{r}^T]^T$ is called the *state vector*; along with the external forces, it can completely describe the state of the system at any point in time. State-space models are conventionally written as

$$\dot{x} = Ax + g(x) + Bu, \tag{53}$$

where x is the *state vector*, A is the *dynamics matrix*, $g(x)$ contains any nonlinear terms in the equations of state, B is the *input matrix*, and u is the *input* to the system. For a linear structure, $g(x) = 0$ and the eigenvalues of A contain the complex *poles* of the system. For under-damped dynamic systems the poles are complex-valued. The imaginary part of the pole is the damped natural frequency, and the real part of the pole is the negative of the natural frequency times the damping ratio.

A convenient feature of state-variable formulations is that first order systems can be easily combined. If components of the structural system have first-order dynamics (for example, visco-elasticity or Bouc-Wen hysteresis), then the first order dynamics of those components can be appended to the state vector, and the simulation of the second order structural system coupled with the first order structural components can proceed in parallel.

The state variable formulation is the most general description of a dynamic system, and many methods exist for the simulation of state-variable models. A set of these methods has been implemented in MATLAB. See, for example, the MATLAB command `ode45` .

For large second-order systems ($n > 1000$), however, the HHT-α method is typically more efficient in terms of memory and speed.

A Fixed-Step Fourth-Order Runge-Kutta Solver

In the computation of transient response analyses of linear or nonlinear systems, it is common for the external forcing to be sampled at uniformly-spaced increments in time, and to desire the response to be computed at these same points in time. If the sample interval $h = \Delta t$ is less than one-tenth of the shortest natural period involved in the system (over the course of the simulation) then a fixed time-step Runge-Kutta solver can compute a solution quickly, and with reasonable accuracy.[8] The most famous of these solvers is fourth-order accurate. The goal of the solver is to advance the solution, x, of a general system of

[8] Press, W.H., *et al.*, *Numerical Recipes,* Cambridge Univ. Press, 1993. Section 16.1 http://www.nr.com/

non-homogeneous first-order ordinary differential equations

$$\dot{x}(t) = f(t, x(t), u(t)) \ , \quad x(0) = x_o \ , \tag{54}$$

from time t to time $t+\Delta t$. Given the state vector $x(t)$, we endeavor to find the state $x(t+\Delta t)$. The solution scheme is illustrated in Figure 1. Within each time-step, the algorithm evaluates

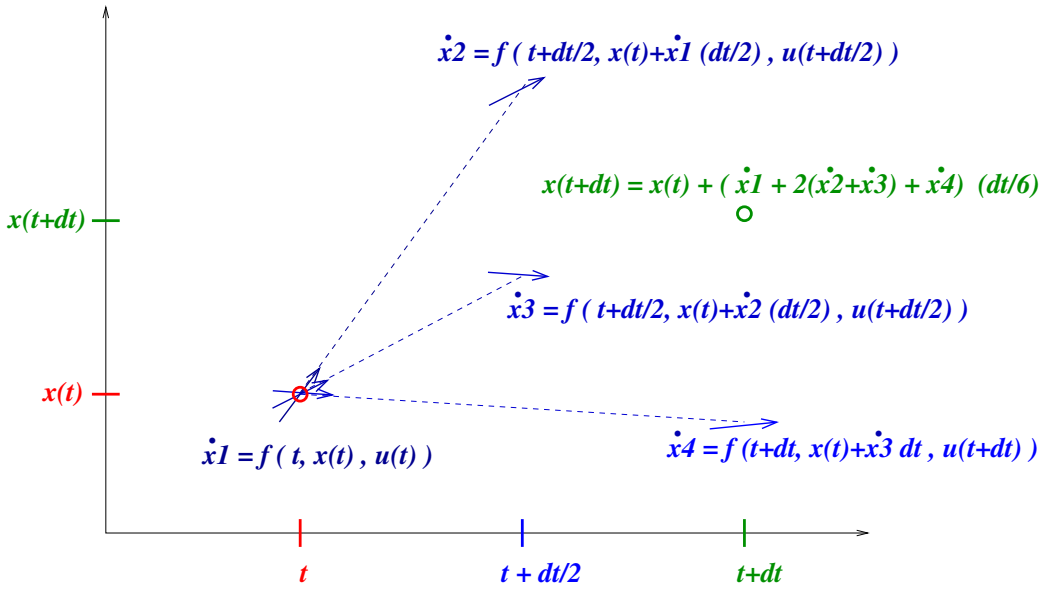


Figure 1. Solution Scheme of the Fourth Order Runge-Kutta Method.

$\dot{x}$ four times.

$$\dot{x}_1 = f\left(t \ , \ x(t) \ , \ u(t)\right) \tag{55}$$

$$\dot{x}_2 = f\left(t + \frac{\Delta t}{2}, \ x(t) + \dot{x}_1 \frac{\Delta t}{2}, u\left(t + \frac{\Delta t}{2}\right)\right) \tag{56}$$

$$\dot{x}_3 = f\left(t + \frac{\Delta t}{2}, \ x(t) + \dot{x}_2 \frac{\Delta t}{2}, u\left(t + \frac{\Delta t}{2}\right)\right) \tag{57}$$

$$\dot{x}_4 = f\left(t + \Delta t \ , \ x(t) + \dot{x}_3 \Delta t \ , \ u(t + \Delta t)\right) \tag{58}$$

The desired solution, $x(t + \Delta t)$, is computed from the four state derivatives

$$x(t + \Delta t) = x(t) + (\dot{x}_1 + 2(\dot{x}_2 + \dot{x}_3) + \dot{x}_4)\frac{\Delta t}{6} \ , \tag{59}$$

time is incremented, $t = t + \Delta t$, and the solution marches on. The solution scheme is implemented in the MATLAB function `ode4u.m`[9]. As long as Δt is shorter than $0.45T_n$ where T_n is the shortest natural period in the system, this constant time-step Runge-Kutta method is stable.

[9] http://www.duke.edu/~hpgavin/ode4u.m

```
function [time, x_sol, x_drv, y_sol] = ode4u( dxdt, time, x0, u, params )
%  [time, x_sol, x_drv, y_sol] = ode4u( dxdt, time, x0, u, params )
%
% Solve a system of nonhomogeneous ordinary differential equations using the
% 4th order Runge-Kutta method.
%
%  Input Variable       Description
%  --------------       -----------
%     dxdt            : a function of the form [x_dot,y] = dxdt(t,x,u,params)
%                       which provides the state derivative given the time, t,
%                       the state, x, external forcing, u, and parameters;
%                       a vector system outputs, y, may also be returned.
%     time            : 1-by-p vector of time values, uniformly increasing
%                       (or decreasing), at which the solution is computed.
%     x0              : n-by-1 vector of initial state values, at time(1).
%     u               : m-by-p matrix of system forcing data for the ode,
%                       sampled at points in the 1-by-p vector time.
%     params          : optional parameters for the function dxdt.
%
% Output Variable       Description
% ---------------       -----------
%     time            : is returned, un-changed.
%     x_sol           : the solution to the differential equation at
%                       times given in the 1-by-p vector time.
%     x_drv           : the derivitive of the solution
%     y_sol           : additional system outputs
%
% Henri Gavin, Civil and Environmental Engineering, Duke Univsersity, Jun. 2014
% WH Press, et al, Numerical Recipes in C, 1992, Section 16.1

 points = length(time);
 if nargin < 5, params = 0;             end
 if nargin < 4, u = zeros(1,points);    end

 [dxdt0,y0]  = feval( dxdt, time(1), x0, u(:,1), params );  % compute initial output

 n = length(x0(:));
 m = length(y0(:));              % number of outputs

 x_sol  = nan(n,points);         % memory allocation for state history
 x_drv  = nan(n,points);         % memory allocation for state derivative history
 y_sol  = nan(m,points);         % memory allocation for output history
 x_sol(:,1) = x0(:);             % the initial conditions for the states
 y_sol(:,1) = y0(:);             % the initial output

 [ru,cu] = size(u);
% pad u with zeros if it is not long enough
 if ( cu < points ), u(:,cu+1:points) = 0; end

 for p = 1:points-1              % advance the solution from p to p+1
     t      = time(p);
     dt     = time(p+1)-t;       % the time step for this interval
     dt2    = dt/2;              % half of the time step
     [dxdt1,y1] = feval( dxdt, t,     x0, u(:,p), params );
     [dxdt2,y2] = feval( dxdt, t+dt2, x0+dxdt1*dt2, (u(:,p)+u(:,p+1))/2.0, params );
     [dxdt3,y3] = feval( dxdt, t+dt2, x0+dxdt2*dt2, (u(:,p)+u(:,p+1))/2.0, params );
     [dxdt4,y4] = feval( dxdt, t+dt,  x0+dxdt3*dt, u(:,p+1), params );
     x0 = x0  +  ( dxdt1 + 2*(dxdt2 + dxdt3) + dxdt4 ) * dt/6.0;
     x_sol(:,p+1) = x0(:);                                % next state
     x_drv(:,p)   = dxdt1(:);                             % current rate
     y_sol(:,p)   = y1(:);                                % current output
     if ~all(isfinite(x0)), break; end                    % floating point error
 end
 [dxdt1,y1]  = feval( dxdt, time(p), x0, u(:,p), params ); % final rate & output
 x_drv(:,p+1) = dxdt1(:);                                 % final rate
 y_sol(:,p+1) = y1(:);                                    % final output
% ------------------------------------------------------------------------ ODE4U
```

A Variable-Step Fourth-Fifth-Order Runge-Kutta Solver

For problems in which the time increments are not much shorter than the shortest natural periods involved in the system, and for problems involving hard nonlinearities, such as problems involving impact or friction, fixed-step Runge-Kutta solvers may be numerically unstable. In such cases one may attempt a re-analysis using a shorter time step for every time step, or one may use a solver in which the large fixed time-steps are automatically sub-divided only when necessary. A feature of such variable time-step solvers is that a desired level of accuracy may be enforced throughout the simulation, by comparing a fourth-order accurate solution with a fifth-order accurate solution.

The implementation of the fourth-fifth order solver discussed here was proposed by Cash and Karp in 1990.[10] To advance the solution from a time t to a time $t+\Delta t$, six state derivatives are computed

$$\dot{x}_1 = f(t+a_1\Delta t,\ x(t),\ u(t+a_1\Delta t)) \tag{60}$$
$$\dot{x}_2 = f(t+a_2\Delta t,\ x(t)+\dot{x}_1 b_{21}\Delta t,\ u(t+a_2\Delta t)) \tag{61}$$
$$\dot{x}_3 = f(t+a_3\Delta t,\ x(t)+(\dot{x}_1 b_{31}+\dot{x}_2 b_{32})\Delta t,\ u(t+a_3\Delta t)) \tag{62}$$
$$\dot{x}_4 = f(t+a_4\Delta t,\ x(t)+(\dot{x}_1 b_{41}+\dot{x}_2 b_{42}+\dot{x}_3 b_{43})\Delta t,\ u(t+a_4\Delta t)) \tag{63}$$
$$\dot{x}_5 = f(t+a_5\Delta t,\ x(t)+(\dot{x}_1 b_{51}+\dot{x}_2 b_{52}+\dot{x}_3 b_{53}+\dot{x}_4 b_{54})\Delta t,\ u(t+a_5\Delta t)) \tag{64}$$
$$\dot{x}_6 = f(t+a_6\Delta t,\ x(t)+(\dot{x}_1 b_{61}+\dot{x}_2 b_{62}+\dot{x}_3 b_{63}+\dot{x}_4 b_{64}+\dot{x}_5 b_{65})\Delta t,\ u(t+a_6\Delta t)) \tag{65}$$

The fourth order predictor at time $t+\Delta t$, is $x_4(t+\Delta t)$, and the fifth order predictor at time $t+\Delta t$, is $x_5(t+\Delta t)$.

$$x_4(t+\Delta t) = x(t)+(\dot{x}_1 c_{41}+\dot{x}_3 c_{43}+\dot{x}_4 c_{44}+\dot{x}_5 c_{45}+\dot{x}_6 c_{46})\Delta t \tag{66}$$
$$x_5(t+\Delta t) = x(t)+(\dot{x}_1 c_{51}+\dot{x}_3 c_{53}+\dot{x}_4 c_{54}+\dot{x}_6 c_{56})\Delta t \tag{67}$$

The coefficients a_i, b_{ij}, and c_{ij} are provided in Cash and Karp's paper. If every term of the truncation error vector, $\epsilon = |x_4 - x_5|/|x_5|$ is less than the required tolerance, $\epsilon_{\rm tol}$, then the solution x_5 is considered sufficiently accurate. In this case, $x(t+\Delta t)$ is assigned to x_5, t is assigned to $t+\Delta t$, and the solution marches on. Otherwise, the time step is divided into $N_{\rm ss}$ sub-steps,[11] where

$$N_{\rm ss} = \left\lceil N_{\rm ss} \max\left[\max\left((\epsilon/\epsilon_{\rm tol})^{1/4}\right), 1.1\right]\right\rceil \ , \tag{68}$$

and $N_{\rm ss}$ is initialized to 1, and $\lceil\cdot\rceil$ is the round-up operator.

These concepts are implemented in the MATLAB function `ode45u.m`[12]

[10] Cash, J.R. and Karp, A.H., "A Variable Order Runge-Kutta Method for Initial Value Problems with Rapidly Varying Right-Hand Sides," *ACM Transactions on Mathematical Software*, vol 16, no 3, pp 201-222 (1990). http://www.duke.edu/~hpgavin/cee541/Cash-90.pdf

[11] Press, W.H., *et al.*, *Numerical Recipes,* Cambridge Univ. Press, 1993. Section 16.2 http://www.nr.com/

[12] http://www.duke.edu/~hpgavin/ode45u.m

The usage of `ode45u.m` is very similar to the usage of `ode4u.m`

ode45u.m

```
% [time, x, x_dot, y] = ode45u( dxdt, time, x0, u, params, tolerance, display )
%
% Solve a system of nonhomogeneous ordinary differential equations
% using the embedded Runge-Kutta formulas of J.R. Cash and A.H. Karp, and
% using linear interpolation of external forcing within time-steps.
%
%  Input Variable       Description
%  ---------------      -----------
%   dxdt            : a function of the form [x_dot,y] = dxdt(t,x,u,params)
%                     which provides the state derivative given the time, t,
%                     the state, x, external forcing, u, and parameters
%                     a vector system outputs, y, may also be returned.
%   time            : 1-by-p vector of time values, uniformly increasing
%                     (or decreasing), at which the solution is computed.
%   x0              : n-by-1 vector of initial state values, at time(1).
%   u               : m-by-p matrix of system forcing data for the ode,
%                     sampled at points in the 1-by-p vector time.
%   params          : optional parameters used in the function dxdt
%   tolerance       : the desired tolerance constant (default 0.001)
%                     may be a vector of the length of x0, or a scalar
%   display         : 3= display lots of results; 2= display less; 1= even less
%
% Output Variable       Description
% ---------------       -----------
%   time            : is returned, un-changed
%   x               : the solution to the differential equation at times
%   x_dot           : the derivative to the differential equation at times
%   y               : the output of the differential equation at times
%
```

Numerical Example

In this example ode4u.m is used to solve the transient response of an inelastic system described by the equations:

$$\ddot{r}(t) + 2\zeta\omega_{\rm n}\dot{r}(t) + \mu z(t) = -\ddot{w}(t) \tag{69}$$

$$\dot{z}(t) = \left(1 - z^3\ \mathrm{sgn}(\dot{r}(t))\right)\ \dot{r}(t)/d_{\rm y}. \tag{70}$$

with initial conditions $r(0) = 0, \dot{r}(0) = 0, z(0) = 0$. The paramter μ is a plastic force level divided by the mass ($\mu = f_{\rm p}/m$), and the paramter $d_{\rm y}$ is a yield displacement. The external forcing is ground acceleration, $-\ddot{w}(t)$, given by

$$\ddot{w}(t) = \begin{cases} A_o \sin(2\pi t/T_{\rm p}) & 0 \le t \le T_{\rm p} \\ 0 & \text{otherwise} \end{cases} \tag{71}$$

In state space, these equations may be expressed as $\dot{x} = f(t, x, u, p)$

$$\frac{d}{dt}\begin{bmatrix} r(t) \\ \dot{r}(t) \\ z(t) \end{bmatrix} = \begin{bmatrix} \dot{r}(t) \\ -2\zeta\omega_{\rm n}\dot{r}(t) - \mu z(t) - \ddot{w}(t) \\ (1 - z^3\ \mathrm{sgn}(\dot{r}(t)))\ \dot{r}(t)/d_{\rm y}. \end{bmatrix} \tag{72}$$

These ordinary differential equations are computed in the short `.m`-file inelasSys.m

```
function [dxdt,fi] = inelasSys(t,x,a_g,p)
% [dxdt,fi] = inelasSys(t,x,a_g,p)
% dynamics of an SDOF inelastic system

% state vector ...
r     = x(1);           % displacement response, m
r_dot = x(2);           % velocity response, m/s
z     = x(3);           % hysteretic variable

% parameters ...
wn    = p(1);           % natural frequency, rad/s
zeta  = p(2);           % viscous damping ratio
mu    = p(3);           % yield force divided by mass, m/s^2
dy    = p(4);           % yield displacement, m

fi = mu*z;              % inelastic force

% state equations
dxdt = [ r_dot  ;                                  % velocity response
         -2*zeta*wn*r_dot - fi - a_g  ;            % acceleration response
         (1 - z^3 * sign(r_dot)) * r_dot/dy];      % inelastic force derivitive
```

And the simulation may be executed using the m-file inelasSim.m

```
% inelasSim.m - simulate the response of a SDOF inelastic system using ode4u.m

% parameters ...
g     =    9.81;        % gravitational acceleration, m/s^2
k     = 5000;           % stiffness, N/m
m     =  100;           % mass, kg
zeta  =    0.02;        % viscous damping ratio
fp    =  95.0;          % plastic force level, N
Tp    =   1.0;          % pulse period, s
Ao    =   1.0;          % pulse amplitude, m/s^2

mu    = fp/m;           % yield force divided by mass, m/s^2
wn    = sqrt(k/m);      % natural frequency, rad/s
dy    = fp/k;           % yield displacement

p  = [ wn , zeta , mu , dy ];   % vector of system parameters

% time axis data ...
T  = 5;                 % duration of simulation, sec
dt = 0.005;             % time step, s
N  = floor(T / dt);     % number of data points
t  = [1:N]*dt;          % time, s

% ground acceleration input disturbance, m/s^2 ...
a_g = Ao*sin(2*pi*t/Tp);  a_g( floor(Tp/dt):end ) = 0;

x0 = [ 0; 0; 0];        % initial states : [ displ; veloc; inelastic force ]
[t,x,dxdt,fi] = ode4u('inelasSys', t, x0, a_g, p); % simulate response using ode4u

% Plots ------------------------------------------------------------------
%epsPlots = 0; if epsPlots, formatPlot(14,7,4); else formatPlot(0); end
figure(1)
  plot( t,x(1,:), t,x(2,:) );
    ylabel('response')
    xlabel('time, s')
    legend('displacement response, m','velocity response, m/s')
%if epsPlots, print('inelasSim1.eps','-color','-solid','-FHelvetica:14'); end
figure(2)
  plot( x(1,:),fi );
    xlabel('displacement, m')
    ylabel('inelastic force/mass, m/s^2')
%if epsPlots, print('inelasSim2.eps','-color','-solid','-FHelvetica:14'); end
```