Time-stepping Algorithms

Many of the differential equations derived on this website contain derivatives with respect to spatial coordinates. However, in plasticity, dynamics, and other problems involving temporal evolution, derivatives with respect to time appears. A basic mathematical example is

$$\dot{x}(t) = r\left(x,t\right) \tag{1}$$

where x is the unknown time-varying quantity, t is time, r is the notation for the right-hand side, and one dot means one derivative with respect to time. Eq. (1) says that r is the derivative of x. Stepping forward in time, one option is the forward Euler method,

$$x_{n+1} = x_n + \Delta t \cdot r\left(x_n\right) \tag{2}$$

Another is the backward Euler method, in which the right-hand side is evaluated at the new time-step:

$$x_{n+1} = x_n + \Delta t \cdot r\left(x_{n+1}\right) \tag{3}$$

Yet another is the midpoint rule:

$$x_{n+1} = x_n + \Delta t \cdot r\left(x_n + \frac{x_{n+1} - x_n}{2}\right) \tag{4}$$

The forward Euler approach is "explicit" because there is no equation to be solved before taking the time-step, while the others are "implicit" methods.

SDOF Problems

The equation of motion that governs single-degree-of-freedom (SDOF) problems is a second-order differential equation:

$$M \cdot \ddot{u}(t) + C \cdot \dot{u}(t) + K \cdot u(t) = F(t) \tag{5}$$

where the symbols are defined in the document on vibration of a single mass. Among the numerical "time-stepping" methods to solve this problem are

- Central difference method
- Houbolt's method
- Constant initial acceleration method
- Constant average acceleration method
- Newmark's β methods
- Generalized Newmark methods
  - Wilson's θ method
  - Hilber's α method
  - Bossak-Newmark method
- Runge-Kutta methods

The *stability* and *accuracy* of such methods are important characteristics. Depending on the size of the time-step, i.e., Δ*t*, a method can be unstable, meaning that the errors grow without bounds. Even in the stable response regime, the accuracy may be poor if Δ*t* is selected too large. Time-stepping algorithms are often written with subscripts that indicate the time-step at which the quantities are evaluated. For example, the notation

$$M \cdot \ddot{u}_n + C \cdot \dot{u}_n + K \cdot u_n = F_n \tag{6}$$

gives the equation of motion evaluated at time *t*, while

$$M \cdot \ddot{u}_{n+1} + C \cdot \dot{u}_{n+1} + K \cdot u_{n+1} = F_{n+1} \tag{7}$$

is the equation of motion at time *t*+Δ*t*. A few algorithms employ Eq. (6) and thus formulate equilibrium at time *t* when solving for the displacement at time *t*+Δ*t*. However, the algorithms presented in this document and implemented in the Python code posted on this website utilize the common approach; employ Eq. (7) and use equilibrium at time *t*+Δ*t* when solving for the displacement at time *t*+Δ*t*. In general, it is possible that information from the time-steps *t*–2Δ*t*, *t*–Δ*t*, *t*, and *t*+Δ*t* is employed to obtain the solution at *t*+Δ*t*. However, the algorithms presented in this document and implemented in the Python code posted on this website utilize only the solution at the previous step. Because the displacement is the integral of the velocity, and the velocity is the integral of the acceleration, it is natural to write time-stepping algorithms as

$$\dot{u}_{n+1} = f(u_n, \dot{u}_n, \ddot{u}_n, \ddot{u}_{n+1}) \tag{8}$$

$$u_{n+1} = f(u_n, \dot{u}_n, \ddot{u}_n, \ddot{u}_{n+1}) \tag{9}$$

Then Eqs. (8) and (9) are solved for $\ddot{u}_{n+1}$ and $\dot{u}_{n+1}$ to facilitate substitution into Eq. (7). In this document, that rearranged time-stepping algorithm is written in the generic format

$$\ddot{u}_{n+1} = a_1 \cdot u_{n+1} + a_2 \cdot u_n + a_3 \cdot \dot{u}_n + a_4 \cdot \ddot{u}_n \tag{10}$$

$$\dot{u}_{n+1} = a_5 \cdot u_{n+1} + a_6 \cdot u_n + a_7 \cdot \dot{u}_n + a_8 \cdot \ddot{u}_n \tag{11}$$

Substitution of Eqs. (10) and (11) into Eq. (7) yields the equilibrium equation

$$\begin{aligned} & M \cdot a_1 \cdot u_{n+1} + M \cdot a_2 \cdot u_n + M \cdot a_3 \cdot \dot{u}_n + M \cdot a_4 \cdot \ddot{u}_n \\ & + C \cdot a_5 \cdot u_{n+1} + C \cdot a_6 \cdot u_n + C \cdot a_7 \cdot \dot{u}_n + C \cdot a_8 \cdot \ddot{u}_n \\ & + K \cdot u_{n+1} = F \end{aligned} \tag{12}$$

Rearranging yields a linear equation for u_{n+1}:

$$\begin{aligned} \left(a_1 \cdot M + a_5 \cdot C + K\right) \cdot u_{n+1} = F & - \left(a_4 \cdot M + a_8 \cdot C\right) \cdot \ddot{u}_n \\ & - \left(a_3 \cdot M + a_7 \cdot C\right) \cdot \dot{u}_n \\ & - \left(a_2 \cdot M + a_6 \cdot C\right) \cdot u_n \end{aligned} \tag{13}$$

After the solution u_{n+1} is determined, Eqs. (10) and (11) provide the velocity and acceleration.

Constant Average Acceleration

Consider the popular method of constant average acceleration, in which the velocity is based on the average acceleration:

$$\dot{u}_{n+1} = \dot{u}_n + \Delta t \cdot \left(\frac{\ddot{u}_{n+1} + \ddot{u}_n}{2} \right) \tag{14}$$

and as a result, the displacement is

$$u_{n+1} = u_n + \Delta t \cdot \dot{u}_n + \Delta t^2 \cdot \left(\frac{\ddot{u}_{n+1} + \ddot{u}_n}{4} \right) \tag{15}$$

This time-stepping algorithm, written in the generic form of Eqs. (10) and (11), is

$$\begin{aligned} a_1 &= \frac{4}{\Delta t^2} \quad a_2 = -\frac{4}{\Delta t^2} \quad a_3 = -\frac{4}{\Delta t} \quad a_4 = -1 \\ a_5 &= \frac{2}{\Delta t} \quad a_6 = -\frac{2}{\Delta t} \quad a_7 = -1 \quad a_8 = 0 \end{aligned} \tag{16}$$

Having those constants, the solution procedure follows the discussion from Eq. (10) to Eq. (13).

Newmark's β Method

This is a family of methods is defined by two parameters γ and β:

$$\dot{u}_{n+1} = \dot{u}_n + (1-\gamma) \cdot \Delta t \cdot \ddot{u}_n + \gamma \cdot \Delta t \cdot \ddot{u}_{n+1} \tag{17}$$

$$u_{n+1} = u_n + \Delta t \cdot \dot{u}_n + \left(\frac{1}{2} - \beta \right) \cdot \Delta t^2 \cdot \ddot{u}_n + \beta \cdot \Delta t^2 \cdot \ddot{u}_{n+1} \tag{18}$$

Written in the generic form of Eqs. (10) and (11), this algorithm reads

$$\begin{aligned} a_1 &= \frac{1}{\beta \cdot \Delta t^2} \quad a_2 = -\frac{1}{\beta \cdot \Delta t^2} \quad a_3 = -\frac{1}{\beta \cdot \Delta t} \quad a_4 = \left(1 - \frac{1}{2\beta} \right) \\ a_5 &= \frac{\gamma}{\beta \cdot \Delta t} \quad a_6 = -\frac{\gamma}{\beta \cdot \Delta t} \quad a_7 = \left(1 - \frac{\gamma}{\beta} \right) \quad a_8 = \Delta t \cdot \left(1 - \frac{\gamma}{2\beta} \right) \end{aligned} \tag{19}$$

Having those constants, the solution procedure follows the discussion from Eq. (10) to Eq. (13).

MDOF Problems: Implicit vs. Explicit

The time-stepping algorithms for multi-degree-of-freedom (MDOF) are straightforward extensions of the algorithms above for SDOF problem. The difference is that the equation of motion now is a system of equations:

$$\mathbf{M}\ddot{\mathbf{u}}_{n+1} + \mathbf{C}\dot{\mathbf{u}}_{n+1} + \mathbf{K}\mathbf{u}_{n+1} = \mathbf{F}_{n+1} \tag{20}$$

where the displacements, velocities, and accelerations are

$$\begin{aligned}\ddot{\mathbf{u}}_{n+1} &= a_1 \cdot \mathbf{u}_{n+1} + a_2 \cdot \mathbf{u}_n + a_3 \cdot \dot{\mathbf{u}}_n + a_4 \cdot \ddot{\mathbf{u}}_n \\ \dot{\mathbf{u}}_{n+1} &= a_5 \cdot \mathbf{u}_{n+1} + a_6 \cdot \mathbf{u}_n + a_7 \cdot \dot{\mathbf{u}}_n + a_8 \cdot \ddot{\mathbf{u}}_n\end{aligned} \tag{21}$$

Time-stepping algorithms are referred to as implicit or explicit. Whether an algorithm is implicit or explicit has little bearing on the computational efficiency for SDOF problems. It is, however, meaningful to discuss those concepts when addressing MDOF problems. With an explicit algorithm, the solution at the next time-step is obtained without solving a system of equations of the generic form **Ku**=**F**. Conversely, implicit methods require the solution of such a system at every time-step. For a time-stepping algorithm to be explicit, it must enforce equilibrium at time *t*, using Eq. (6). Furthermore, must only use quantities at time *t* in the right-hand side of Eqs. (8) and (9). It seems computationally appealing to formulate explicit time-stepping algorithms, but they are not addressed here. One reason is problems with accuracy; another reason is that in nonlinear structural analysis it is necessary to solve the system of equilibrium equations at each time-step anyway, as part of the Newton-Raphson scheme.