Nonlinear Time Integration

Claudio Perez

```
pip install sdof
```

Theory

We are looking to approximate second order differential equations of the form

$$\mathbf{M}\ddot{\boldsymbol{u}} + \mathbf{C}\dot{\boldsymbol{u}} + \boldsymbol{p}(\boldsymbol{u}, \dot{\boldsymbol{u}}) = \boldsymbol{f}$$

Newmark Equations

The Newmark-β scheme is defined by the following equations:

$$\begin{aligned} \boldsymbol{v}_{n+1} &= \boldsymbol{v}_n + (1-\gamma)\Delta t\ \boldsymbol{a}_n + \gamma\Delta t\, \boldsymbol{a}_{n+1} \\ \boldsymbol{d}_{n+1} &= \boldsymbol{d}_n + \Delta t\ \boldsymbol{v}_n + \tfrac{1}{2}\Delta t^2\left((1-2\beta)\,\boldsymbol{a}_n + 2\beta\boldsymbol{a}_{n+1}\right) \end{aligned} \tag{1}$$

for scalar parameters β and γ and the following notation is used:

$$\begin{aligned} \boldsymbol{d}_n &\approx \boldsymbol{u}(t_n) \\ \boldsymbol{v}_n &\approx \dot{\boldsymbol{u}}(t_n) \\ \boldsymbol{a}_n &\approx \ddot{\mathbf{u}}(t_n) \end{aligned}$$

Remarks

- The Newmark-β follows from Taylor's theorem with explicit remainder which states that the first time derivative can be expressed as:

  $$\begin{aligned} \dot{\boldsymbol{u}}_{n+1} &= \dot{\boldsymbol{u}}_n + \Delta t\ \ddot{\mathbf{u}}_\gamma \\ \boldsymbol{u}_{n+1} &= \boldsymbol{u}_n + \Delta t\ \dot{\boldsymbol{u}}_n + \tfrac{1}{2}\Delta t^2\ \ddot{\mathbf{u}}_\beta. \end{aligned}$$

  for some $\ddot{\mathbf{u}}_\gamma$ and $\ddot{\mathbf{u}}_\beta$ with the form:

  $$\begin{aligned} \ddot{\mathbf{u}}_\gamma &= (1-\gamma)\ddot{\boldsymbol{u}}_n + \gamma\ddot{\boldsymbol{u}}_{n+1} && 0 \le \gamma \le 1 \\ \ddot{\mathbf{u}}_\beta &= (1-2\beta)\ddot{\mathbf{u}}_n + 2\beta\ddot{\mathbf{u}}_{n+1} && 0 \le 2\beta \le 1 \end{aligned}$$

- **Explicit central difference scheme** is obtained by setting $\gamma = 0.5$ and $\beta = 0$

- **Average constant acceleration (Middle point rule)** is obtained by setting $\gamma = 0.5$ and $\beta = 0.25$

$$\frac{1}{\Delta t}(v_{n+1} - v_n) = (1-\gamma)a_n + \gamma a_{n+1}$$

Generalized α Scheme

The generalized α scheme imposes equilibrium with the following form:

$$\mathbf{M}\,\boldsymbol{a}_{\alpha_m} + \boldsymbol{p}(\boldsymbol{d}_{\alpha_f}, \boldsymbol{v}_{\alpha_f}) = \boldsymbol{f}(t_{\alpha_f})$$

where

$$\begin{aligned} t_{\alpha_f} &\triangleq (1-\alpha_f)\,t_{n+1} + \alpha_f\,t_n \\ \boldsymbol{d}_{\alpha_f} &\triangleq (1-\alpha_f)\,\boldsymbol{d}_{n+1} + \alpha_f\,\boldsymbol{d}_n \\ \boldsymbol{v}_{\alpha_f} &\triangleq (1-\alpha_f)\,\boldsymbol{v}_{n+1} + \alpha_f\,\boldsymbol{v}_n \\ \boldsymbol{a}_{\alpha_m} &\triangleq (1-\alpha_m)\,\boldsymbol{a}_{n+1} + \alpha_m\,\boldsymbol{a}_n \end{aligned}$$

Implementation

The Newmark equations allow one to express all three unknowns (i.e., a_{n+1}, v_{n+1} or d_{n+1}) in terms of one of these.

The most straight-forward way to implement the Newmark scheme, is to form a single equation in terms of unknown accelerations, $\boldsymbol{a}_{n+1}$:

$$\mathbf{A}\boldsymbol{a}_{n+1} = \boldsymbol{b}$$

for some known matrix $\mathbf{A}$ and vector $\boldsymbol{b}$. This is obtained by plugging the Newmark equations into the discrete equilibrium statement at t_{n+1}. However before deriving this, it is useful rewrite Equation (1) as follows:

$$\begin{aligned} \boldsymbol{d}_{n+1} &= \tilde{\boldsymbol{d}} + c_{da}\,\boldsymbol{a}_{n+1} \\ \boldsymbol{v}_{n+1} &= \tilde{\boldsymbol{v}} + c_{va}\,\boldsymbol{a}_{n+1} \end{aligned}$$

where the tilde variables allow us to collect everything that is known at the start of a time step. Expanding and plugging into the discretized equilibrium equation furnishes the Newmark scheme in "a-form":

$$\begin{aligned} \tilde{\boldsymbol{d}} &= \boldsymbol{d}_n + \Delta t\boldsymbol{v}_n + \tfrac{1}{2}\Delta t^2(1-2\beta)\boldsymbol{a}_n \\ \tilde{\boldsymbol{v}} &= \boldsymbol{v}_n + \Delta t(1-\gamma)\boldsymbol{a}_n \\ (\mathbf{M} + \gamma\Delta t\,\mathbf{C} + \beta\Delta t^2\,\mathbf{K})\boldsymbol{a}_{n+1} &= \boldsymbol{f}_{n+1} - \mathbf{C}\tilde{\boldsymbol{v}} - \mathbf{K}\tilde{\boldsymbol{d}} \end{aligned}$$

$$\begin{aligned} \boldsymbol{d}_{n+1} &= \tilde{\boldsymbol{d}} + \beta\Delta t^2\,\boldsymbol{a}_{n+1} \\ \boldsymbol{v}_{n+1} &= \tilde{\boldsymbol{v}} + \gamma\Delta t\,\boldsymbol{a}_{n+1} \end{aligned}$$

Generalized Unknowns

Alternatively, the Newmark equations can be manipulated to produce a problem in terms of velocity or displacement.

If $\boldsymbol{x}_{n+1}$ denotes our chosen unknown we can write the equations in the following form:

$$\begin{aligned}
\boldsymbol{d}_\alpha &= \tilde{\boldsymbol{d}}_x + c_{dx}\,\boldsymbol{x}_{n+1}\\
\boldsymbol{v}_\alpha &= \tilde{\boldsymbol{v}}_x + c_{vx}\,\boldsymbol{x}_{n+1}\\
\boldsymbol{a}_\alpha &= \tilde{\boldsymbol{a}}_x + c_{ax}\,\boldsymbol{x}_{n+1}
\end{aligned}$$

This generalizes as follows :

$$\begin{aligned}
\tilde{\boldsymbol{d}} &= \sum_i b_{di}\boldsymbol{y}_i\\
\tilde{\boldsymbol{v}} &= \sum_i b_{vi}\boldsymbol{y}_i\\
\tilde{\boldsymbol{a}} &= \sum_i b_{ai}\boldsymbol{y}_i\\
(c_a\,\mathbf{M} + c_v\,\mathbf{C} + c_d\,\mathbf{K})\boldsymbol{x}_{n+1} &= \boldsymbol{f}_{n+1} - \left(\mathbf{M}\tilde{\boldsymbol{a}} + \mathbf{C}\tilde{\boldsymbol{v}} + \mathbf{K}\tilde{\boldsymbol{d}}\right)
\end{aligned}$$

$$\begin{aligned}
\boldsymbol{d}_{n+1} &= \tilde{\boldsymbol{d}} + c_d\,\boldsymbol{x}_{n+1}\\
\boldsymbol{v}_{n+1} &= \tilde{\boldsymbol{v}} + c_v\,\boldsymbol{x}_{n+1}\\
\boldsymbol{a}_{n+1} &= \tilde{\boldsymbol{a}} + c_a\,\boldsymbol{x}_{n+1}
\end{aligned}$$

where $\boldsymbol{y}_i = (\boldsymbol{d}_n, \boldsymbol{v}_n, \boldsymbol{a}_n)$ coefficients c_{yx} and b_{ij} given below, and where $\tilde{(\cdot)}_x$ variables encapsulate the information that is known at the start of the time step.

Nonlinear Generalized - α

Applying this to a nonlinear problem yields:

$$r(\boldsymbol{x}_{n+1}) = \mathbf{M}\,[\tilde{\boldsymbol{a}} + c_{ax}\,\boldsymbol{x}_{n+1}] + \mathbf{C}\,[\tilde{\boldsymbol{v}} + c_{vx}\,\boldsymbol{x}_{n+1}] + \boldsymbol{p}\left(\tilde{\boldsymbol{d}} + c_{dx}\,\boldsymbol{x}_{n+1}, \tilde{\boldsymbol{v}} + c_{vx}\,\boldsymbol{x}_{n+1}\right) - \boldsymbol{f}(t_{n+1})$$

Linearizing for $r_\eta = r(\boldsymbol{x}^i_{n+1} + \eta\, d\boldsymbol{x})$ yields

$$\mathcal{L}\, r_\eta = r(\boldsymbol{x}^i_{n+1}) + \mathbf{A}d\boldsymbol{x}$$

where

$$\mathbf{A}d\boldsymbol{x} = \frac{d}{d\eta} r_\eta \bigg|_{\eta=0} = \left(c_d\mathbf{K}\left(\tilde{\boldsymbol{d}} + c_d\,\boldsymbol{x}_{n+1}\right) + c_v\mathbf{C} + c_a\mathbf{M}\right) d\boldsymbol{x}$$

For a chosen unknown $\boldsymbol{x}$, we can collect coefficients into a "predictor matrix" B_x, and corrector array c_{yx} so that any of the other unknowns, $\boldsymbol{y} \in (\boldsymbol{d}, \boldsymbol{v}, \boldsymbol{a})$ can be expressed as:

$$\boldsymbol{y}_{n+1} = \mathsf{B}_x\,\boldsymbol{y}_n + c_{yx}\,\boldsymbol{x}_{n+1}$$

From the Newmark equations, the coefficients c_{xy} follow as:

$$c_{xy} = \begin{pmatrix} 1 & \frac{\gamma}{\beta\Delta t} & \frac{1}{\beta\Delta t^2} \\ \frac{\Delta t\beta}{\gamma} & 1 & \frac{1}{\gamma\Delta t} \\ \beta\Delta t^2 & \gamma\Delta t & 1 \end{pmatrix}$$

The matrix B is given for the acceleration formulation ($\boldsymbol{x} \triangleq \boldsymbol{a}$) by:

$$\begin{pmatrix} \tilde{\boldsymbol{d}} \\ \tilde{\boldsymbol{v}} \\ \tilde{\boldsymbol{a}} \end{pmatrix}_a = \begin{pmatrix} 1 & \Delta t & \Delta t^2 \left(\frac{1}{2} - \beta\right) \\ & 1 & \Delta t \left(1 - \gamma\right) \\ & & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{d}_n \\ \boldsymbol{v}_n \\ \boldsymbol{a}_n \end{pmatrix}$$

In the velocity formulation ($\boldsymbol{x} \triangleq \boldsymbol{v}$):

$$\begin{pmatrix} \tilde{\boldsymbol{d}} \\ \tilde{\boldsymbol{v}} \\ \tilde{\boldsymbol{a}} \end{pmatrix}_v = \begin{pmatrix} 1 & -\Delta t \frac{\beta}{\gamma} \left(1 - \frac{\gamma}{\beta}\right) & \Delta t^2 \frac{\beta}{\gamma} \left(\frac{\gamma}{2\beta} - 1\right) \\ & 0 & \\ & \frac{-1}{\gamma\Delta t} & 1 - \frac{1}{\gamma} \end{pmatrix} \begin{pmatrix} \boldsymbol{d}_n \\ \boldsymbol{v}_n \\ \boldsymbol{a}_n \end{pmatrix}$$

Finally, for the displacement formulation ($\boldsymbol{x} \triangleq \boldsymbol{d}$):

$$\begin{pmatrix} \tilde{\boldsymbol{d}} \\ \tilde{\boldsymbol{v}} \\ \tilde{\boldsymbol{a}} \end{pmatrix}_d = \begin{pmatrix} 0 & & \\ -\frac{\gamma}{\beta\Delta t} & 1 - \frac{\gamma}{\beta} & \Delta t \left(1 - \frac{\gamma}{2\beta}\right) \\ \frac{1}{\beta\Delta t^2} & \frac{-1}{\beta\Delta t} & \left(1 + \frac{1}{2\beta}\right) \end{pmatrix} \begin{pmatrix} \boldsymbol{d}_n \\ \boldsymbol{v}_n \\ \boldsymbol{a}_n \end{pmatrix}$$

References

- J. Chung, G.M.Hubert. “A Time Integration Algorithm for Structural Dynamics with Improved Numerical Dissipation: The Generalized-α Method” ASME Journal of Applied Mechanics, 60, 371:375, 1993.

Source Code

```c
int sdof_integrate_plastic(struct sdof_alpha* conf,
    double M, double C, double K, double scale, int n, double *p, double dt,
    double *response)
{
    const double gamma   = conf->gamma;
    const double beta    = conf->beta;
    const double alpha_m = conf->alpha_m;
    const double alpha_f = conf->alpha_f;

    const double c1 = 1.0;
    const double c2 = gamma/(beta*dt);
    const double c3 = 1.0/(beta*dt*dt);

    const double b1 =     (1.0 -     gamma/beta);
    const double b2 =  dt*(1.0 - 0.5*gamma/beta);
    const double b3 = -1.0/(beta*dt);
    const double b4 =  1.0 - 0.5/beta;

    const double k0 = alpha_f*c2*C + alpha_m*c3*M;

    double  pa = 0.0, // TODO: Find pa for initial u0
           *u = &response[0], // Initialize pointers to user-supplied memory
           *v = &response[1], //
           *a = &response[2]; //

    const int past = -3, // Pointer offsets
              pres =  0; //

    // Plasticity parameters
    const int max_iter = 10;
    double up    = 0.0,
           tol   = 1e-12,
           Fy    = 7.5,
           alpha = 0.00,
           Hkin  = K*alpha/(1.0 - alpha);

    int i     = 0;                          // Time index
    double Kt = K;                          // Tangent stiffness
    a[pres]   = (p[i] - C*v[pres] - pa)/M; // Initial acceleration

    // Time loop
    for (i = 1; i < n; i++) {
      u += 3; v += 3; a += 3; // Move current state pointers forward

      // PREDICTOR
      u[pres]   = u[past];
      v[pres]   = b1*v[past] + b2*a[past];
      a[pres]   = b4*a[past] + b3*v[past];
      // values at alpha time
      double ua = (1.0 - alpha_f)*u[past] + alpha_f*u[pres];
      double va = (1.0 - alpha_f)*v[past] + alpha_f*v[pres];
      double aa = (1.0 - alpha_m)*a[past] + alpha_m*a[pres];

      double R = scale*p[i] - (M*aa + C*va + pa);
      double R0 = 1.0; //R;
      if (R0 == 0.0) R0 = 1.0;

      double du = 0.0;
      for (int iter = 0; iter <= max_iter; iter++) {

        ua += alpha_f*c1*du; // Update values at alpha time
        va += alpha_f*c2*du; //
        aa += alpha_m*c3*du; //

        // State determination for pa, Kt
        pa = K*(ua - up);
        double ftrial = fabs(pa - Hkin*up) - Fy;

        if (ftrial <= 0) {
          Kt = K;

        } else {
          double dg = ftrial/(K + Hkin);

          if (pa < 0) {
            pa += dg*K;
            up -= dg;
          } else {
            pa -= dg*K;
            up += dg;
          }

          Kt = K*Hkin/(K + Hkin);

        }

        // SOLVE
        R  = scale*p[i] - (M*aa + C*va + pa);
        du = R/(alpha_f*c1*Kt + k0);

        // UPDATE
        u[pres] += c1*du;
        v[pres] += c2*du;
        a[pres] += c3*du;

        // Check convergence
        if (fabs(R/R0) < tol)
          break;
      }
    }
    return 1;
}
```