```
function[record] = Transient(acc, MatData, MatState, time_step_method, algorithm_type, max_iter)
  %  Time Steping
  beta = time_step_method.beta; % Time stepping Newmark-Beta Method parameter
  alpha = time_step_method.alpha; % Time stepping Newmark-Beta Method parameter
  dt = time_step_method.time_step;% Time stepping Newmark-Beta Method time step

  % initial all recorded variables
  [record.R_total, record.R, record.C, record.M, record.num_steps, record.U, record.V, record.A] = deal(zeros(numel(acc),1));
  record.time = dt:dt:dt*numel(acc); % Record down the time
   record.acc = acc; % Record acceleration
   record.MatData = MatData; % Record material data

   % Structural Variables
   m = MatData.mass; % mass
   c = MatData.damping; % dampin coefficient
   P = -m * acc; % Convert to force
   A = MatData.A; % area
   L =MatData.L; % length
   Delta_u = 0; % Total dispalcemet from last converged displacement
   U_conv = 0; % Last converged displacement

   % Check for algorithm to use
   switch algorithm_type
      case "Newton"
         tag = 1;
      case "ModifiedNewton"
         tag = 2;
   end

   % Solving the system
   for n = 1:numel(P)-1% Loop over "time"-load steps
      conv = 0; % Convergence is false
      j = 1; % Iteration counter
      switch tag % Check for algorithm to use
         case 2
            algorithm_type = "ModifiedNewton";
            Ktan = MatData.A*MatState.Pres.Et/MatData.L;  % Tangent stiffness
      end
      m_= m* (1/beta/dt^2*record.U(n) + 1/beta/dt*record.V(n) - (1-1/2/beta)*record.A(n)); % Calculate a temporary term
      d_ = c * ( alpha/beta/dt*record.U(n) - (1-alpha/beta)*record.V(n) - dt*(1-0.5*alpha/beta)*record.A(n)); % Calculate a temporary term
      P_tilde = P(n+1) + m_ + d_;  % Known Resisting forces- Calcualte P_tilde
      %% Loop over Newton-Raphson iterations
      while ( j <= max_iter && conv == 0)
         %% Calculate Unbalance force
         % Unknown Resisting forces; These depend on the iterations, Ui
         m_u = m* (1/beta/dt^2*U_conv) ; % Mass term
         c_u = c * (alpha/beta/dt*U_conv); % Damping term
         MatState = Mate25n(MatData,MatState); R = A*MatState.Pres.sig;  % R(Ui) % Stiffness term, requires material state determination
         Unb = P_tilde - (m_u + c_u + R); % Unbalanced force
         record.Unb(n,j) = abs(Unb);  % Record unbalance force
         %% Check Convergence
         % Converged branch
         if (abs(Unb) < 1.e-5)  % Converged criteria; norm of the residual
            % Commit the next displacement, velocity, and acceleration
            record.U(n+1) = U_conv; % Set the next displacement as the current state displacement
            record.V(n+1) = dt*(1-alpha/2/beta)*record.A(n) + (1-alpha/beta)*record.V(n)+alpha/beta/dt*(record.U(n+1)-record.U(n)); % Next↙
time step velocity
```

```
57              record.A(n+1) = (1-1/beta/2)*record.A(n) - 1/beta/dt*record.V(n) +1/beta/dt^2*(record.U(n+1)-record.U(n)); % Next time step
accleration
58              record.R(n+1) = R; % Record Internal Resisting force
59              record.C(n+1) = c*record.V(n+1); % Record Damping Forces
60              record.M(n+1) = m*record.A(n+1);% Record Inertia Forces
61              record.total(n+1) = m*record.A(n+1) + c*record.V(n+1)+ R; % Record total resisting force
62              % Reset State variables
63              Delta_u = 0; % Reset ΔU for the iteration
64              MatState.eps(1,2) = 0; % Reset Δε for the iteration
65              MatState.eps(1,3) = 0;% Reset δε for the iteration
66              MatState.Past = MatState.Pres; % Saves the state
67              conv = 1; % Converged
68              record.iter(n) = j; % Record number of iterations
69          else % Has not converged
70              % Check algorithm to use
71              if algorithm_type == "Newton"
72                  Ktan = MatData.A*MatState.Pres.Et/MatData.L;  % Tangent stiffness
73              end
74              if j == max_iter
75                  disp("Could not converged using " + algorithm_type + newline + "Switching to Newton-Raphson Method");
76                  j = 1; algorithm_type = "Newton";
77              end
78              Ktan_dynamic = 1/beta/dt^2 *m + alpha/beta/dt*c + Ktan; % Dynamic tangential stiffness
79              % Update displacement variables
80              delta_u = Unb/Ktan_dynamic; % Calculate (δUi)_n+1
81              Delta_u = Delta_u + delta_u; % ΔU = ΔU + δU for the iteration
82              U_conv = U_conv + delta_u; % U = U + δU for the iteration
83              % Update strain variables
84              MatState.eps(1,1) = U_conv/L;  % Total strain
85              MatState.eps(1,2) = Delta_u/L;  % Total incremental strain from last converged state
86              MatState.eps(1,3) = delta_u/L;  % Last incremental strain
87              j = j + 1; % Increaseiteration counter
88          end
89      end
90  end
91 end
```