In the following function, we construct the residual needed for time integration of

second-order system

$\mathbf{M}\ddot{\mathbf{q}} + \mathbf{C}\dot{\mathbf{q}} + \mathbf{F}(\mathbf{q}) = \mathbf{F}_{ext}(t)$,

where we use the residual is defined as

$\mathbf{r}(\ddot{\mathbf{q}}, \dot{\mathbf{q}}, \mathbf{q}) = \mathbf{M}\ddot{\mathbf{q}} + \mathbf{C}\dot{\mathbf{q}} + \mathbf{F}(\mathbf{q}) - \mathbf{F}_{ext}(t)$.

A generic Residual function, whose *handle* is passed for performing Implicit Newmark and Generalized- α nonlinear time integration schemes in this code has the following syntax:

```
[r, drdqdd, drdqd, drdq, c0] = Residual(q,qd,qdd,t);
```

where

$$\begin{aligned}
&\texttt{r} = \mathbf{r},\\
&\texttt{drdqdd} = \frac{\partial \mathbf{r}}{\partial \ddot{\mathbf{q}}},\\
&\texttt{drdqd} = \frac{\partial \mathbf{r}}{\partial \dot{\mathbf{q}}},\\
&\texttt{drdq} = \frac{\partial \mathbf{r}}{\partial \mathbf{q}},\\
&\texttt{c0} = \text{a scalar measure for comparing the residual norm while checking for convergence}
\end{aligned}$$

The extra arguments:

1. `Assembly`, which is an instance of Assembly class
2. `Fext`, which is a function handle for the external forcing,

are required for computing the residual in this case.

New residual functions that follow the above-mentioned syntax can be written according to user preference. This way, the same time integration class can be used to solve a variety of problems.

Please refer to the Mechanical directory in the examples folder to understand applications and usage.

```
function [ r, drdqdd,drdqd,drdq, c0] = residual_nonlinear( q, qd, qdd, t, Assembly, Fext)
```

In this function, it is assumed that the matrices $\mathbf{M}, \mathbf{C}$ for the finite element mesh were precomputed and stored in the `DATA` property of the `Assembly` object to avoid unnecessary assembly during each time-step.

```
M = Assembly.DATA.M;
C = Assembly.DATA.C;
```

The tangent stiffness $\mathbf{K} = \frac{\partial \mathbf{F}}{\partial \mathbf{q}}$ and the the internal force $\mathbf{F}$, however, need to be assmbled at each iteration depending on the current state. To perform this assembly, we first need the constrained vector of displacements $\mathbf{q}$ to be converted to its counterpart $\mathbf{u}$ that contains all the degrees of freedom.

```
u = Assembly.unconstrain_vector(q);
[K, F] = Assembly.tangent_stiffness_and_force(u);
```

These matrices and the external forcing vector are appropriately constrained according to the boundary conditions:

```
M_red = Assembly.constrain_matrix(M);
C_red = Assembly.constrain_matrix(C);
K_red = Assembly.constrain_matrix(K);
F_elastic = Assembly.constrain_vector(F);
F_external =  Assembly.constrain_vector(Fext(t));
```

Residual is computed according to the formula above:

```
F_inertial = M_red * qdd;
F_damping = C_red * qd;

r = F_inertial + F_damping + F_elastic - F_external ;

drdqdd = M_red;
drdqd = C_red;
drdq = K_red;
```

We use the following measure to comapre the norm of the residual $\mathbf{r}$

$$\texttt{c0} = \|\mathbf{M}\ddot{\mathbf{q}}\| + \|\mathbf{C}\dot{\mathbf{q}}\| + \|\mathbf{F}(\mathbf{q})\| + \|\mathbf{F}_{ext}(t)\|$$

```
c0 = norm(F_inertial) + norm(F_damping) + norm(F_elastic) + norm(F_external);

end
```