RESEARCH ARTICLE OPEN ACCESS

Integration of Finite Element Method with Runge – Kuta Solution Algorithm

[1]Olawale Simon, [1]Ogunbiyi Moses A, [2]Alabi Olusegun and [3]Ofuyatan Olatokunbo

[1] *Department of Civil Engineering, Faculty of Engineering and Environmental Sciences, Osun State University, Osogbo, Osun State, Nigeria*
[2] *Department of Mathematical and Physical Sciences, Faculty of Basic and Applied Sciences, Osun State University, Osogbo, Osun State,Nigeria*
[3] *Department of Civil Engineering,college of Engineering, Covenant University, Otta, Ogun State, Nigeria*

ABSTRACT
Runge – Kuta (RK) method is reasonably simple and robust for numerical solution of differential equations but it requires an intelligent adaptive step-size routine; to achieve this, there is need to develop a good logical computer code. This study develops a finite element code in Java using Runge-Kuta method as a solution algorithm to predict dynamic time response of structural beam under impulse load. The solution obtained using direct integration and the present work is comparable.

I. INTRODUCTION

In numerical analysis, the Runge-Kuta method is a family of implicit and explicit iterative methods, which includes the well – known routine called Euler methods, used in temporal discretization for the approximate solution of Ordinary Differential Equation (ODE) (Devries and Hasbun, 2011). Runge-Kuta method is reasonably simple and robust and is a good candidate for numerical solution of differential equations when combined with an intelligent adaptive step-size routine (Abramowitz and Stegun, 1972).The Runge-Kuta Algorithm is known to be very accurate and well – behaved for a wide range of problems but to describe it precisely we need to develop some notation and a good logical computer code; which this study endeavored to achieve.

II. THEORETICAL BACKGROUND

Finite Element Analysis (FEA) is a branch of solid mechanics which can be applied to solve multi-physics problems. Its applications include structural analyses, solid mechanics, dynamics, thermal analysis, electrical analysis and biomaterials (Hughes, 1987 and Logan, 2002). The major purpose of FEA is to determine the values of the displacements, stresses and strains at each material point if a force is applied on a solid (Jerry, 2006).

The Runge-Kuta algorithm works over time step increment to implicitly calculate the responses over time domain, starting from the initial time t_0 to the time limit t_{max}.

Methodology: Study Solution Development

The equation of motion in single degree of freedom (SDF) is given by

$$m^*\ddot{u} + \beta m^*\dot{u} + k^*u = f^*(t) \quad (1)$$

and the displacement equation in terms of shape functions and time is given by

$$u(x,t) = [\varphi_1(x) \quad \varphi_2(x) \quad \varphi_3(x) \quad \varphi_4(x)\ ]\ u(t) \quad (2)$$

or u(x,t) = [A] u(t) and the shape functions are defined as follows :

$$\begin{aligned} \varphi_1(x) &= 1 - \frac{3x^2}{L^2} + \frac{2x^3}{L^3} \\ \varphi_2(x) &= x - \frac{2x^2}{L} + \frac{x^3}{L^2} \\ \varphi_3(x) &= \frac{3x^2}{L^2} - \frac{2x^3}{L^3} \\ \varphi(x) &= -\frac{x^2}{L} + \frac{x^3}{L^2} \end{aligned} \quad (3)$$

And u(t) is the nodal displacement at time t

External forces: g(x,t) = $-m\ddot{u}\,(x,t)$4

and

f (x,t) is the applied force

By the principle of virtual works:

$$\delta w_i = \delta w_e \quad 5$$

$$\delta w_e = \int_0^l \{g(x,t) \,.\, \delta u\}\, dx + \int_0^l \{f(x,t) \,.\, \delta u\}\, dx$$
$$- \int_0^l \{m\ddot{u}(x,t) \,.\, \delta u\}\, dx + \int_0^l \{f(x,t) \,.\, \delta u\}\, dx \quad 6$$

$$\delta w_i = \int_0^l \{EI\, u''(x,t) \,.\, \delta u''\}\, dx + \int_0^l \{\beta m\dot{u}(x,t) \,.\, \delta u\}\, dx \quad 7$$

where

$$u'' = \frac{d^2[A]}{dx^2} U(t)$$
$$\ddot{u} = [A]\, \ddot{u}(t)$$
$$\delta u = [A]\, \delta u(t) \quad 8$$
$$\delta u'' = \frac{d^2[A]}{dx^2} \delta u\,(t)$$

$$\delta w_e = -\int_0^l \{m[A]\,\ddot{u}[A]^T\, \delta u\}\, dx + \int_0^l \{f[A]\, \delta u\}\, dx \quad 9$$

$$\delta w_i = \delta u\{-\ddot{u}\int_0^l m[A]\,[A]^T\, dx + \int_0^l f[A]\, dx\} \quad 10$$

$$\delta w_i = \int_0^1 \{EI[\frac{d^2[A]}{dx^2}][\frac{d^2[A]}{dx^2}]^T \delta u\}\, dx + \int_0^1 \{\beta m[A][A]^T\, \delta u\, \dot{u}\}\, dx$$

$$= \delta u \left\{ \int_0^1 \{EI[\frac{d^2[A]}{dx^2}][\frac{d^2[A]}{dx^2}]^T_{\dot{u}}\}\, dx + \int_0^1 \{\beta m[A][A]^T \dot{u}\}\, dx \right\}$$

Equating 9 and 10

$$u \int_0^1 EI[\frac{d^2[A]}{dx^2}][\frac{d^2[A]}{dx^2}]^T\, dx + \dot{u} \int_0^1 \beta m[A][A]^T\, dx$$

$$-\ddot{u}\int_0^l m[A][A]^T\, dx + \int_0^l f[A]\, dx \quad 11$$

$$m^*\ddot{u} + c^*\dot{u} + k^*u = f^*$$

Where the consistent matrices of mass, stiffness, damping and force are given below

$$m^* = \int_0^1 m[A][A]^T\, dx$$

$$k^* = \int_0^1 EI[\frac{d^2[A]}{dx^2}][\frac{d^2[A]}{dx^2}]^T\, dx$$

$$c^* = \int_0^1 \beta m[A][A]^T\, dx$$

$$f^* = \int_0^1 f[A]\, dx$$

Runge-Kuta Method of Solution

The solution to the equation of motion can be obtained using Runge-Kuta (RK) method which very suited to initial condition system. However, the integration of Finite Element Method with RK method requires some careful of considerations because the overall global U vector is a combination of displacement and velocity vectors. The RK solution decomposes the equation of motion into two equations $U_1 = U$ and $U_2 = dU_1/dt$. Thus the initial conditions to start the solution procedure are given below. Please note that U is the combination of global displacement and velocity and is different from u.

$U_1 = 0$ (U_{1pre}), $U_2 = 0$ (U_{2pre}) at t = 0

$\overrightarrow{U_0} = \binom{0}{0} \binom{U_{1\,pre}}{U_{2\,pre}}$

$\frac{d}{dt}\binom{U_1}{U_2} = \left(\begin{matrix} U_2 \\ \frac{f(t)}{m} - \frac{c}{m}U_2 - \frac{k}{m}U_1 \end{matrix}\right)$ (2,8) matrix 12

$\vec{K}_1 = \Delta t f(t, \overrightarrow{U})$

$\vec{K}_2 = \Delta t f(t + \frac{\Delta t}{2}, \frac{\vec{K}_1}{2} + \overrightarrow{U})$

$\vec{K}_3 = \Delta t f(t + \frac{\Delta t}{2}, \frac{\vec{K}_2}{2} + \overrightarrow{U})$

$\vec{K}_3 = \Delta t f(t + \Delta t, \vec{K}_3 + \overrightarrow{U})$

$U_{i+1} = u_i + \frac{1}{6}(\vec{K}_1 + 2\vec{K}_1 + 2\vec{K}_3 + \vec{K}_4)$

$\vec{K}_1, \vec{K}_2, \vec{K}_3, \vec{K}_4$ and $\vec{U}_i$ are vector of $2N \times 1$ size.

Infact, where N x N is the size of global consistent stiffness, damp and mass matrices

Pseudo Code

Step 1: Calculate the member stiffness matrix $[K]_{4x4}$, mass matrix $[M]_{4x4}$and damping matrix $[C]_{4x4} = \beta\ [M]_{4x4}$

Step 2: Set start time t[0] = tini

Calculate the time step dt = $\frac{t_{max} - t_{ini}}{n}$, n being the total steps

Step 2: Set up $[U]_{initial}$ and set $[U]_{i-1} = [U]_{initial}$

Step 3: Set time t[i] = t [i-1] + dt

Step 4: Assemble the global stiffness matrix $[K]_{NxN}$, mass matrix $[M]_{NxN}$ and damping matrix $[C]_{NxN} = \beta\ [M]_{NxN}$

Step 5: Compute $\left(\begin{matrix} [U_2] \\ [M^{-1}][F - CU_2 - KU_1] \end{matrix}\right)$ $2N \times 1$

Step 6: Compute

$[K_1]_i = dt * fun\ (t[i-1], [U]_i)$

$[K_2]_i = dt * fun\ (t[i-1] + \frac{dt}{2}, [U]_i + \frac{[K_1]_i}{2})$

$[K_3]_i = dt * fun\ ([i-1] + \frac{dt}{2}, [U]i + \frac{[K_2]_i}{2})$

$[K_4]_i = dt * fun\ (t[i-1]) + dt, [U]_i + [K_3]_i)$

$[U]_i = [U]_{i-1} + ([K_1]_i) + 2 * [K_2]_i + 2 * [K_3]_i + [K_4]_i) / 6.0$

Step 7: Extract global displacement, velocities and compute acceleration which are N x 1 size.

Step 8: Increase time to t [i] = t [i -1] + dt and repeat Step 5 to Step 7.

III. RESULTS AND DISCUSSION

This study tests the present solution of the equation of motion by analyzing a prismatic concrete beam of 200 x 200 mm cross section by 3.0m length. The study used material characteristic of Young's modulus of 48.39 MPa and yield stress of 65.00 MPa. A triangular force excitation of maximum value of 500KN, decaying to zero on the positive phase of 0.015 ms was applied over a time domain. The same problem was analyzed using Direct Integration and Runge-Kuta methods for both damped and un-damped situations. The results of the comparison of the two methods are shown below in figures 1 and 2 respectively.

The agreement between the two methods is reasonable and indicates that Runge-Kuta method integrated with Finite Element Method can result in accurate prediction of the time response of structural elements over the period of excitation. With more attention paid to details, the two methods can seamlessly converge to the same solution with practically no difference.

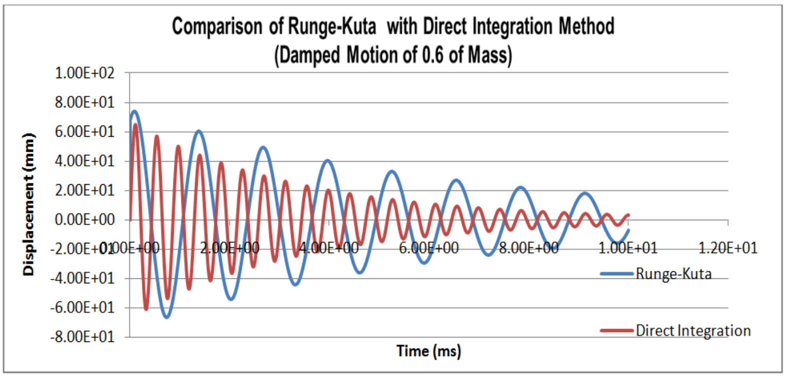

Figure 1: Comparison of Runge-Kuta Method with Direct Integration Method for damped motion.

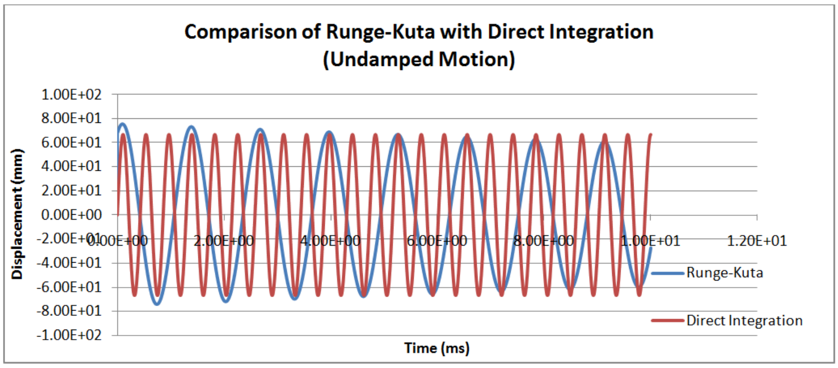

Figure 2: Comparison of Runge-Kuta Method with Direct Integration Method for un-damped motion.

IV. CONCLUSION

The agreement between the two methods is reasonable and indicates that Runge-Kuta method integrated with finite element method can result in accurate prediction of the time response of structural elements over the period of excitation.

REFERENCES

[1]. Abramowtiz, M. and Stegun, I.A. (1972) Handbook of Mathematical foundation with formulas, Graphs and Mathematical tables 9th Edition, New York: Dover pp. 896-897.
[2]. Barthe K.J. (1996) The finite element procedures. Prentice Hall.
[3]. Devries, P.L. and Hasbun, J.E. (2011) A first course in Computational Physics (2nd Edition) Jones and Bartlett Publishers, pg. 215.
[4]. Huges, T.J.R. (1987) The finite element methods: Linear static and dynamic finite element analysis. Dove publication.
[5]. Jerry, H.Q. (2006) Finite element analysis note book.
[6]. Logen, D.L.(2002) A first course in finite element (3rd Edition)

Appendix

Although the detailed listing of the Java code may be required by some inquisitive readers, effort is made to provide the Javadoc listings below to assist in recreating the code quickly.

Java Code Definitions:
Class DynaBeamRK

- java.lang.Object
  -
    - DynaBeamRK
-

---

public class DynaBeamRK
extends java.lang.Object

- **Constructor Summary**

  Constructors

  Constructor and Description

  DynaBeamRK(int numberElemen, float timeLimit, int numberOfTimeStep)

- **Method Summary**

  Methods

Modifier and Type	**Method and Description**
static void	calcK1(int step, float deltaTime)

static void	**calcK2**(int step, float deltaTime)
static void	**calcK3**(int step, float deltaTime)
static void	**calcK4**(int step, float deltaTime)
static void	**calcU**(int step)
static void	**calcU0**()
static void	**calcU01**()
static void	**calcU02**()
static void	**cofactor**(float[][] num)
static void	**computeAcceleration**(int step)
static void	**computeElementMatrix**()
static void	**computeElemForces**(int t)
static void	**computeForce**(int step, float addedT)
static void	**computeNodalAccel**(int t)
static void	**computeNodalDisp**(int t)
static void	**computeNodalVel**(int t)
static void	**computeTimeDispHistory**()
static void	**computeTimeRespHistRK**()
static float	**determinant**(float[][] num, int s)
static void	**initialise**()
static void	**initialiseIntermediate**()
static void	**main**(java.lang.String[] args)
static void	**readBasicInput**()
static void	**readInputData**()
static void	**transpose**(float[][] num)

- **Methods inherited from class java.lang.Object**
  clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

○ **Constructor Detail**

- **DynaBeamRK**
- public DynaBeamRK(int numberElemen,
- float timeLimit,
  int numberOfTimeStep)

○ **Method Detail**

- **calcU01**
  public static void calcU01()
- **calcU02**
  public static void calcU02()
- **calcU0**
  public static void calcU0()
- **initialiseIntermediate**
  public static void initialiseIntermediate()
- **calcK1**
- public static void calcK1(int step,
  float deltaTime)
- **calcK2**
- public static void calcK2(int step,
  float deltaTime)
- **calcK3**
- public static void calcK3(int step,
  float deltaTime)
- **calcK4**
- public static void calcK4(int step,
  float deltaTime)

- **calcU**
  public
  static void calcU(int step
)
- **readInputData**
  public
  static void readInputDat
  a()
- **readBasicInput**
  public
  static void readBasicInp
  ut()
- **initialise**
  public
  static void initialise()
- **computeElementMatri
  x**
  public
  static void computeElem
  entMatrix()
- **computeTimeDispHist
  ory**
  public
  static void computeTime
  DispHistory()
- **computeTimeRespHist
  RK**
  public
  static void computeTime
  RespHistRK()
- **computeNodalDisp**
  public
  static void computeNod
  alDisp(int t)
- **computeNodalVel**
  public
  static void computeNod
  alVel(int t)
- **computeNodalAccel**
  public
  static void computeNod
  alAccel(int t)
- **computeElemForces**
  public
  static void computeElem
  Forces(int t)
- **computeForce**
- public
  static void computeForc
  e(int step,
  float addedT)
- **computeAcceleration**
  public
  static void computeAcce
  leration(int step)
- **determinant**
- public
  static float determinant(f
  loat[][] num,
  int s)
- **cofactor**
  public
  static void cofactor(float
  [][] num)
- **transpose**
  public
  static void transpose(flo
  at[][] num)
- **main**
  public
  static void main(java.lan
  g.String[] args)