ON THE INTEGRATION OF EQUATIONS OF MOTION: FEM AND MOLECULAR DYNAMICS PROBLEMS

E.G. Kakouris, V.K. Koumousis

Institute of Structural Analysis & Aseismic Research

National Technical University of Athens

Zografou Campus, 15780, Athens, Greece

E-mail: manolis_kakou@hotmail.com, vkoum@central.ntua.gr

Key Words: *Equations of motion, direct time integration, Verlet Method, Verlet Leapfrog Method, Velocity Verlet Method, Analog Equation Method, structural dynamics, molecular dynamics simulation*

ABSTRACT

In this work, a comparison between direct time integration algorithms is presented for the solution of the equations of motion for linear finite element as well as molecular dynamics problems. The comparison is performed regarding the fundamental requirements of accuracy, stability, simplicity, speed and minimization of computing resources. The Verlet [1] algorithm is widely used for molecular dynamics problems due to its simplicity and robustness. There are two alternative versions of Verlet algorithm i.e. the Verlet Leapfrog and Velocity Verlet, which are algebraically equivalent. However, these two methods have an advantage of minimum storage requirements because of storing one set of velocities and positions at each time step [2]. Taking advantage of this, Verlet Leapfrog and Velocity Verlet integration algorithms are applied to simple multi-degree of freedom linear finite element examples describing their dynamic response. There is also a comparison between the previous integration algorithms and Constant-Average Acceleration Method (Newmark's trapezoidal rule) [3] as well as the direct integration method of Analog Equation Method introduced by J.T. Katsikadelis [4] [5], according to which the system of the N coupled equations of motion is treated by a set of uncoupled linear quasi-static equations involving the unknown displacements and unknown fictitious external loads. The Analog Equation Method is also applied to large molecular dynamics systems describing the trajectories of atoms. The accuracy of Analog Equation Method in molecular dynamics simulation is verified by the energy conservation law of the system. In addition, the energy drifts depending on different values of time step Δt are investigated with this method.

1. INTRODUCTION

In dynamic analysis the equation of motion are solved by considering the equilibrium at time t which satisfy the equilibrium of external, internal and inertial forces. Dynamic equilibrium may be written as:

$$\mathrm{F(t)_{Inertia\ forces}} + \mathrm{F(t)_{Damping\ forces}} + \mathrm{F(t)_{Internal\ forces}} = \mathrm{F(t)_{Exertal\ forces}} \qquad (1)$$

Where inertia forces are acceleration-dependent, damping forces are velocity-dependent and internal forces are displacement-dependent. For a structural system at time t Eq. (1) becomes:

$$\mathbf{M} \cdot \ddot{u}(t) + \mathbf{C} \cdot \dot{u}(t) + \mathbf{K} \cdot u(t) = \mathbf{P}(t) \qquad (2)$$

C is the damping matrix while **M** and **K** is the mass and stiffness matrix respectively. **M** and **K** are positive definite in contrast to **C** which is positive semi-definite. Eq. (2) represents a coupled system of second order differential equations and the problem consists of solving these in time t, where $t \in [0, \tau], \tau > 0$, satisfying the dynamic equilibrium (2) and the initial conditions $u(0) = u_o$ and $\dot{u}(0) = \dot{u}_o$ [6].

There is a significant number of direct integration methods which had developed in order to solve numerically Eq. (2). In direct integration methods the equations are integrated using a numerical step-by-step procedure and there is no transformation into a different form of these equations. However, different direct integration methods consist different accuracy, stability and computing resourses. Direct integration methods are categorized into two general classes of algorithms, implicit and explicit. If a direct computation between the dependent variables can be occurred in terms of known quantities, the algorithm is said to be explicit. If a matrix system is solved in each time step to advance the solution the algorithm is said to be implicit.

It is clear that the computation cost per time as well as the computing storage requirements for implicit methods is greater than explicit methods. However, implicit methods tend to be more stable and permit large time steps than explicit methods. Explicit methods tend to be unstable when large time steps occurs and generally require much smaller time step than implicit methods increasing the computing cost. In general, implicit algorithms tend to be more effective for structural dynamics problems because of small values of time periods T_n while explicit methods perform better in high frequencies structural modes [6] [7].

Due to their simplicity, minimum storage requirements and speed, explicit methods are widely used in molecular dynamics problems where these parameters are crucial. Verlet algorithm [1] (and its alternative versions, Leapfrog and Velocity Verlet) is well known explicit method which describe the trajectories of molecules solving the following equation of motion:

$$m_i \cdot \ddot{r}(t) = \sum_{j=1, j\neq i}^{N_m} f_{ij}(t) \qquad (3)$$

Where N_m is the total number of molecules, m_i is the mass of each molecule, $f_{ij}(t)$ is the acting force in molecule and $\ddot{r}(t)$ is the acceleration of molecule at time t. The two alternative versions of Verlet algorithms are obtained by considering the Taylor expansion:

$$\begin{aligned} \mathrm{r}(\mathrm{t}+\Delta t) &= r(t)+\Delta t\cdot\dot{r}(t)+\frac{\Delta t^2}{2}\cdot\ddot{r}(t) = r(t)+\Delta t\cdot\left(\dot{r}(t)+\frac{\Delta t}{2}\cdot\ddot{r}(t)\right) \\ &= r(t)+\Delta t\cdot\dot{r}\left(t+\frac{\Delta t}{2}\right) \end{aligned} \tag{4}$$

And

$$\begin{aligned} \mathrm{r}(\mathrm{t}-\Delta t) &= r(t)-\Delta t\cdot\dot{r}(t)+\frac{\Delta t^2}{2}\cdot\ddot{r}(t) = r(t)-\Delta t\cdot\left(\dot{r}(t)-\frac{\Delta t}{2}\cdot\ddot{r}(t)\right) \\ &= r(t)-\Delta t\cdot\dot{r}\left(t-\frac{\Delta t}{2}\right) \end{aligned} \tag{5}$$

Subtracting Eq. (5) from Eq. (4):

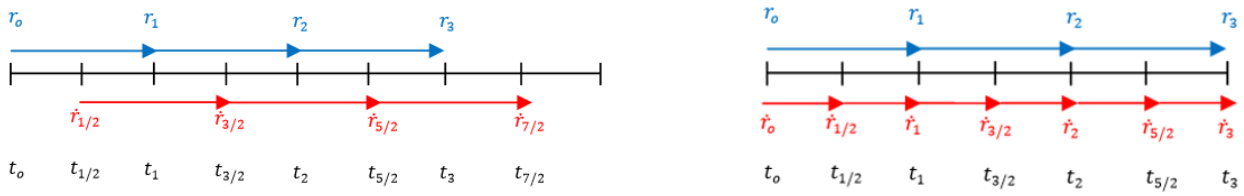
$$\dot{r}\left(t+\frac{\Delta t}{2}\right) = \dot{r}\left(t-\frac{\Delta t}{2}\right)+\Delta t\cdot\ddot{r}(t) \tag{6}$$

Or

$$\dot{r}(t) = \dot{r}\left(t+\frac{\Delta t}{2}\right)-\Delta t\cdot\ddot{r}(t) \tag{7}$$

Where Eq. (6) represents the Leapfrog Verlet and Eq. (7) the Velocity Verlet algorithm. These two alternative versions of Verlet algorithm are totally algebraically equivalent but have an advantage of minimum storage requirements because of storing one set of velocities and positions at each time step [2]. The difference between these two algorithms is that in Leapfrog the velocities are calculated in half a time step behind (or in front of), the current time step n while in Velocity Verlet both position and velocities are calculated at the same instant of time as it is shown in **Fig. 1**.

Fig. 1 Leapfrog Verlet Algorithm (Left) and Velocity Verlet (Right)



In **Table 1** is presented pseudo-codes of the referred algorithms in FEM problems.

Table 1 Verlet, Verlet Leapfrog and Velocity Verlet Algorithms in FEM problems

Verlet	**Verlet Leapfrog**	**Velocity Verlet**
Read: $\mathrm{M},\ \mathrm{C},\mathrm{K},\ \mathrm{u_o},\ \dot{\mathrm{u}}_\mathrm{o},\ \mathrm{p},\ \mathrm{dt}$		
Initial Calculations		
$\ddot{\mathrm{u}}_\mathrm{o} = \mathrm{M}^{-1}\cdot(\mathrm{p_o}-\mathrm{C}\cdot\dot{\mathrm{u}}_\mathrm{o}-\mathrm{K}\cdot\mathrm{u_o})$	$\ddot{\mathrm{u}}_\mathrm{o} = \mathrm{M}^{-1}\cdot(\mathrm{p_o}-\mathrm{C}\cdot\dot{\mathrm{u}}_\mathrm{o}-\mathrm{K}\cdot\mathrm{u_o})$	$\ddot{\mathrm{u}}_\mathrm{o} = \mathrm{M}^{-1}\cdot(\mathrm{p_o}-\mathrm{C}\cdot\dot{\mathrm{u}}_\mathrm{o}-\mathrm{K}\cdot\mathrm{u_o})$
$\dot{\mathrm{u}}_{-1} = \mathrm{u_o}-\mathrm{dt}\cdot\ddot{\mathrm{u}}_\mathrm{o}+\frac{\mathrm{dt}^2}{2}\cdot\ddot{\mathrm{u}}_\mathrm{o}$	$\dot{\mathrm{u}}_{-1/2} = \mathrm{u_o}-\frac{\mathrm{dt}}{2}\cdot\ddot{\mathrm{u}}_\mathrm{o}$ $\dot{\mathrm{u}}_{1/2} = \dot{\mathrm{u}}_{-1/2}+\mathrm{dt}\cdot\ddot{\mathrm{u}}_\mathrm{o}$	$\dot{\mathrm{u}}_{1/2} = \dot{\mathrm{u}}_\mathrm{o}+\mathrm{dt}\cdot\ddot{\mathrm{u}}_\mathrm{o}$
Start Main Loop		
$\mathrm{u_{n+1}} = \left(\frac{\mathrm{M}}{\mathrm{dt}^2}-\frac{\mathrm{C}}{\mathrm{dt}^2}\right)^{-1}\cdot\left(\mathrm{p_n}-\mathrm{u_n}\cdot\left(\mathrm{K}-\frac{2\cdot\mathrm{M}}{\mathrm{dt}^2}\right)-\mathrm{u_{n-1}}\cdot\left(\frac{\mathrm{M}}{\mathrm{dt}^2}-\frac{\mathrm{C}}{2\cdot\mathrm{dt}}\right)\right)$	$\dot{\mathrm{u}}_{\mathrm{n+1/2}} = \dot{\mathrm{u}}_{\mathrm{n-1/2}}+\mathrm{dt}\cdot\ddot{\mathrm{u}}_\mathrm{n}$	$\dot{\mathrm{u}}_{\mathrm{n+1/2}} = \dot{\mathrm{u}}_\mathrm{n}+\frac{\mathrm{dt}}{2}\cdot\ddot{\mathrm{u}}_\mathrm{n}$

$\dot{u}_n = \frac{u_{n+1} - u_{n-1}}{2 \cdot dt}$	$u_{n+1} = u_n + dt \cdot \dot{u}_{n+1/2}$	$u_{n+1} = u_n + dt \cdot \dot{u}_{n+1/2}$
$\ddot{u}_n = \frac{u_{n+1} - 2 \cdot u_n + u_{n-1}}{dt^2}$	$\ddot{u}_{n+1} = M^{-1} \cdot (p_{n+1} + C \cdot \dot{u}_{n+1/2} - K \cdot u_{n+1})$	$\ddot{u}_{n+1} = M^{-1} \cdot (p_{n+1} + C \cdot \dot{u}_{n+1/2} - K \cdot u_{n+1})$
	$\dot{u}_n = \frac{1}{2} \cdot (\dot{u}_{n+1/2} + \dot{u}_{n-1/2})$	$\dot{u}_{n+1} = \dot{u}_{n+1/2} + \frac{dt}{2} \cdot \ddot{u}_{n+1/2}$
$n = n + 1$		
End Main Loop		

Because of these algorithms are categorized as explicit methods, they become unstable for large time steps. In general, the methods applied only for time steps:

$$dt < \frac{2}{\omega_{max}} \tag{8}$$

However, the new direct explicit integration method of Analog Equation Method, introduced by J.T. Katsikadelis [4] [5], overcomes the above restriction, permitting large time steps as well as it inherits all the advantages of explicit methods. According to Analog Equation Method (AEM) the system of the N coupled equations of motion is treated by a set of uncoupled linear quasi-static equations involving the unknown displacements and unknown fictitious external loads. The pseudo-code of the Analog Equation Method in FEM problems is presented in **Table 2**.

Table 2 Analog Equation Method in FEM problems

Analog Equation Method
Read: $M,\ C, K,\ u_o,\ \dot{u}_o,\ p,\ dt$
Initial Calculations
$c_1 = \frac{dt^2}{2}$
$c_2 = dt$
$q_o = M^{-1} \cdot (p_o - C \cdot \dot{u}_o - K \cdot u_o)$
$U_o = \{q_o \quad \dot{u}_o \quad u_o\}$
$G = \begin{bmatrix} M & C & K \\ \frac{c_1}{2} \cdot I & -c2 \cdot I & I \\ -\frac{c_2}{2} \cdot I & I & 0 \end{bmatrix}$, $H = \begin{bmatrix} 0 & 0 & 0 \\ -\frac{c_1}{2} \cdot I & 0 & I \\ \frac{c_2}{2} \cdot I & I & 0 \end{bmatrix}$, $b2 = \begin{bmatrix} I \\ 0 \\ 0 \end{bmatrix}$
Solve: $[G] \cdot [A, b] = [H, b2]$ → **Find:** $[A]$ **and** $[b]$
Main Loop
$U_n = A \cdot U_{n-1} + b \cdot p_n$
$n = n + 1$
End Main Loop

In **Table 3** is summarized the properties of the above referred algorithms as well as the Constant-Average Acceleration Method (CAAM) [3].

Table 3 Properties of the referred algorithms

Integration Algorithm	Type	Stability Condition	Order of Accuracy
Verlet/Leapfrog Verlet/Velocity Verlet	Explicit	Conditional $\left(dt < \frac{2}{\omega_{max}}\right)$	2
Constant-Average Acceleration Method	Implicit	unconditional	2
Analog Equation Method	Explicit	unconditional	2

Taking advantage of the stability and accuracy of Analog Equation Method, the pseudo-code (see **Table 2**) is modified to be applied in large molecular dynamics systems. Due to the fact that **K** and **C**

matrices are zero as well as **M** matrix is diagonal, there is an analytical procedure with the MAPLE in order to minimize the computation cost.

As referred from **Table 2** $c_1, c_2, \mathbf{G}, \mathbf{H}$ and $\mathbf{b2}$ matrices are given from Eq. (9) to (13)

$$c_1 = \frac{dt^2}{2} \tag{9}$$

$$c_2 = dt \tag{10}$$

$$[G]_{(3N_m \times 3N_m)} = \begin{bmatrix} m_1 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & m_{N_m} & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0.5 \cdot c_1 & \cdots & 0 & -c_2 & \cdots & 0 & 1 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0.5 \cdot c_1 & 0 & \cdots & -c_2 & 0 & \cdots & 1 \\ -0.5 \cdot c_2 & \cdots & 0 & 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & -0.5 \cdot c_2 & 0 & \cdots & 1 & 0 & \cdots & 0 \end{bmatrix} \tag{11}$$

$$[H]_{(3N_m \times 3N_m)} = \begin{bmatrix} 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ -0.5 \cdot c_1 & \cdots & 0 & 0 & \cdots & 0 & 1 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & -0.5 \cdot c_1 & 0 & \cdots & 0 & 0 & \cdots & 1 \\ 0.5 \cdot c_2 & \cdots & 0 & 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0.5 \cdot c_2 & 0 & \cdots & 1 & 0 & \cdots & 0 \end{bmatrix} \tag{12}$$

$$[b2]_{(3N_m \times N_m)} = \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \\ 0 & \cdots & 0 \\ \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \\ 0 & \cdots & 0 \end{bmatrix} \tag{13}$$

Therefore, the inverse of **G** matrix is given by:

$$[G]^{-1}_{(3N_m \times 3N_m)} = \begin{bmatrix} m_1 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & m_{N_m} & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0.5 \cdot c_2 & \cdots & 0 & 0 & \cdots & 0 & 1 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0.5 \cdot c_2 & 0 & \cdots & 0 & 0 & \cdots & 1 \\ 0.5 \cdot (c_2^2 - c_1) & \cdots & 0 & 1 & \cdots & 0 & c_2 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0.5 \cdot (c_2^2 - c_1) & 0 & \cdots & 1 & 0 & \cdots & c_2 \end{bmatrix} \tag{14}$$

And **A** and **b** matrices become:

$$[A]_{(3N_m\times 3N_m)} = [G]^{-1}_{(3N_m\times 3N_m)} \cdot [H]_{(3N_m\times 3N_m)} = \begin{bmatrix} 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0.5\cdot c_2 & \cdots & 0 & 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0.5\cdot c_2 & 0 & \cdots & 1 & 0 & \cdots & 0 \\ 0.5\cdot (c_2^2 - c_1) & \cdots & 0 & c_2 & \cdots & 0 & 1 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0.5\cdot (c_2^2 - c_1) & 0 & \cdots & c_2 & 0 & \cdots & 1 \end{bmatrix} \quad (15)$$

$$[b]_{(3N_m\times N_m)} = [G]^{-1}_{(3N_m\times 3N_m)} \cdot [b2]_{(3N_m\times N_m)} = \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \\ 0.5\cdot c_2 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0.5\cdot c_2 \\ 0.5\cdot (c_2^2 - c_1) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0.5\cdot (c_2^2 - c_1) \end{bmatrix} \quad (16)$$

Matrices $\mathbf{U_n}$ and $\mathbf{p_{n+1}}$ are given by Eq. (17) and (18)

$$[U_n]_{(3N_m\times 1)} = \begin{bmatrix} \ddot{r}_{1_n} \\ \vdots \\ \ddot{r}_{Nm_n} \\ \dot{r}_{1_n} \\ \vdots \\ \dot{r}_{Nm_n} \\ r_{1_n} \\ \vdots \\ r_{Nm_n} \end{bmatrix} \quad (17)$$

$$[p_{n+1}]_{(N_m\times 1)} = \begin{bmatrix} \ddot{r}_{1_{n+1}} \\ \vdots \\ \ddot{r}_{Nm_{n+1}} \end{bmatrix} \quad (18)$$

Finally, the new positions, velocities and acceleration of molecules at time n+1 are given by Eq. (19)

$$[U_{n+1}]_{(3N_m\times 1)} = \begin{bmatrix} \ddot{r}_{1_{n+1}} \\ \vdots \\ \ddot{r}_{Nm_{n+1}} \\ \dot{r}_{1_{n+1}} \\ \vdots \\ \dot{r}_{Nm_{n+1}} \\ r_{1_{n+1}} \\ \vdots \\ r_{Nm_{n+1}} \end{bmatrix} = [A]_{(3N_m\times 3N_m)} \cdot [U_n]_{(3N_m\times 1)} + [b]_{(3N_m\times N_m)} \cdot [p_{n+1}]_{(N_m\times 1)} \quad (19)$$

Or more easily from Eq. (20) to (22)

$$\ddot{r}_{n+1} = p_{n+1} \quad (20)$$

$$\dot{r}_{n+1} = \dot{r}_n + 0.5\cdot c_2 \cdot \ddot{r}_n + 0.5\cdot c_2 \cdot \ddot{r}_{n+1} \quad (21)$$

$$r_{n+1} = r_n + c_2 \cdot \dot{r}_n + (-c_1 + c_2^2)\cdot \ddot{r}_n \quad (22)$$

In **Table 4** is presented the pseudo-code for both Velocity Verlet and Analog Equation Method in molecular dynamics problems. In this way the reader interested in implementing the Analog Equation Method in molecular dynamics problems can easily modify his code.

Table 4 Velocity Verlet (or Leapfrog) and Analog Equation Method in Molecular Dynamics problems

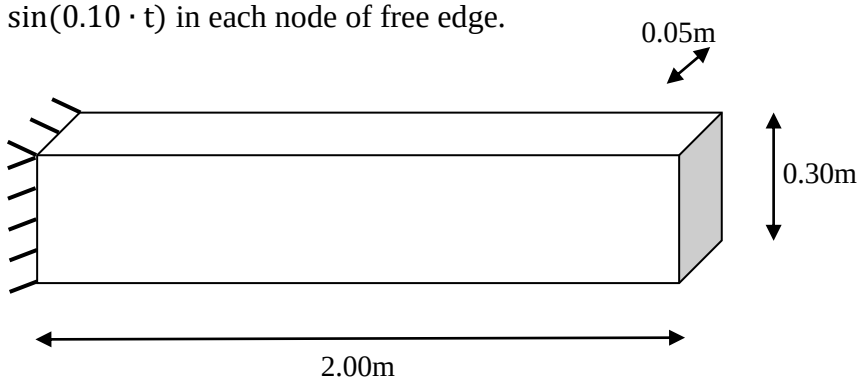
Velocity Verlet	**Analog Equation Method**
Read: $r_o,\ \dot{r}_o,\ \ddot{r}_o,\ N_m,\ V,\ dt$	
Initial Calculations	
	$c_1 = \frac{dt^2}{2}\ ,\quad c_2 = dt$
Start Main Loop	
First half step	
$\dot{r}_{n+1/2} = \dot{r}_n + \frac{dt}{2}\cdot \ddot{r}_n$	$r_{n+1} = r_n + c_2\cdot \dot{r}_n + (-c_1 + c_2^2)\cdot \ddot{r}_n$
$r_{n+1} = r_n + dt\cdot \dot{r}_{n+1/2}$	
Compute forces	
$f_{ij} = -\nabla V_{ij}$	
$\ddot{r}_{n+1} = f_i = \sum_{j=1,j\neq i}^{N_m} f_{ij}$	
Second half step	
$\dot{r}_{n+1} = \ddot{r}_{n+1/2} + \frac{dt}{2}\cdot \ddot{r}_{n+1}$	$\dot{r}_{n+1} = \dot{r}_n + 0.5\cdot c_2\cdot \ddot{r}_n + 0.5\cdot c_2\cdot \ddot{r}_{n+1}$
End Main Loop	

2. NUMERICAL EXAMPLES

Example 1: Finite Element Problem (Large multi-degree of freedom system)

* A MATLAB program has been written for the following example

There is a comparison among the above referred integration methods (see **Table 3**) in stability, accuracy, speed and computing resources. It is considered the following free-fixed beam with dynamic load $P(t) = 8\cdot \sin(0.10\cdot t)$ in each node of free edge.



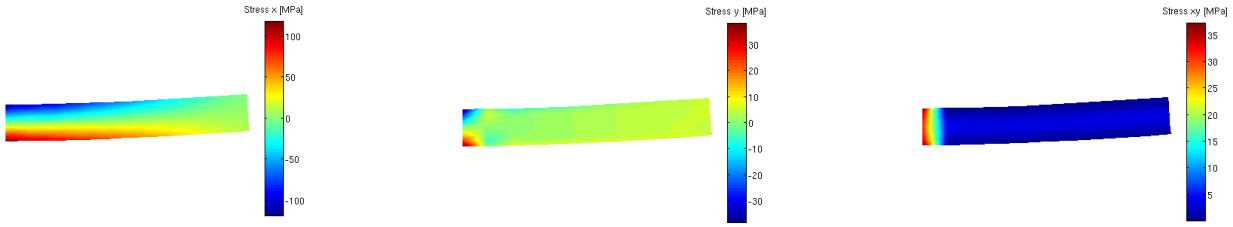
Young's modulus: $E = 200.000\text{MPa}$

Poisson's ratio: $v = 0.30$

Mass density: $\rho = 7800\,\text{kg/m}^3$

It is analyzed by 60 Plane Stress Finite Elements with 154 degrees of freedom for total time 126 [sec]. The stresses x, y and xy are shown in **Fig. 2** at time 20 [sec].

Fig. 2 Stresses x (Left), y (Center) and xy (Right) at time 20 [sec] of free-fixed beam



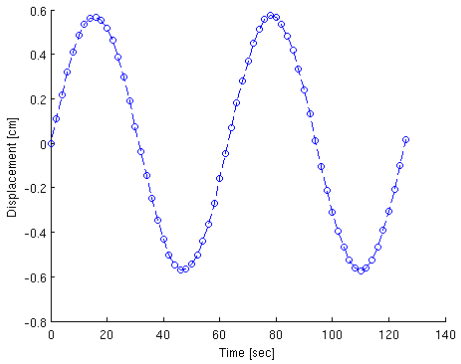
Verlet algorithms are applied only for time steps $dt < {}^{2}/_{\omega_{max}} = 0.0005$. For that time step the computation time in all integration methods are shown in **Table 5**.

Table 5 Computation time in Example 1 for time step 0.00005 and total time 126 [sec]

Method	Verlet	Constant-Average Acceleration	Analog Equation	Leapfrog	Velocity Verlet
t [sec]	318.93	711.40	329.04	273.15	273.15

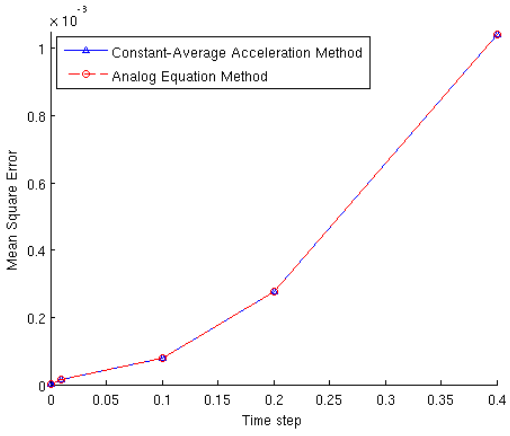
The vertical displacement in the free edge of the beam is shown in **Fig. 3**.

Fig. 3 Vertical Displacement in the free edge of the beam



It is clear from **Table 5** that Leapfrog and Velocity Verlet algorithms are faster than the others and require minimum computing resources but the time step ($dt = 0.00005$) is generally expensive in terms of computing cost. Therefore, the comparison is continued with AEM and CAAM. Because, it is impossible to be found an analytical solution of the vertical displacement, exact solution is considered the solution for $dt = 0.00005$ from Verlet integration. In **Fig. 4** is illustrated the mean square error between these two methods for the vertical displacement of free edge of the beam.

Fig. 4 Vertical displacement Mean Square Error for Constant-Average Acceleration and Analog Equation Method



Because it is hard to be noticed which method is more accurate, **Table 6** represented the above figure in numbers.

Table 6 Vertical displacement Mean Square Error for Constant-Average Acceleration and Analog Equation Method

MSE (Mean Square Error)		
Method	**CAAM**	**AEM**
0.0001	3.681358313584288e-11	3.681329129885530e-11
0.001	1.795657713985809e-08	1.795657299459770e-08
0.01	1.607258067367435e-05	1.607258020850032e-05
0.1	8.021742812878739e-05	8.021742819714664e-05
0.2	2.763579117048229e-04	2.763579120898301e-04
0.4	0.001041129446680	0.001041129446438

In terms of accuracy, the two methods have almost the same accuracy since both methods are 2^{nd} order accurate. However, due to the fact that AEM is explicit methods while CAAM is implicit, AEM is many times faster than CAAM as it is shown in **Table 7**.

Table 7 Computation time in Example 1 for Constant-Average Acceleration Method and Analog Equation Method

Δt:	**0.0001**		**0.001**		**0.01**		**0.1**		**0.20**		**0.40**	
Method	**CAAM**	**AEM**	**CAAM**	**AEM**	**CAAM**	**AEM**	**CAAM**	**AEM**	**CAAM**	**AEM**	**CAAM**	**AEM**
t [sec]	290.540	82.391	28.544	8.291	2.844	0.819	0.331	0.083	0.181	0.044	0.109	0.026

Example 2: Analog Equation Method in molecular dynamics problem (Stability and Accuracy)

* A C program has been written for the following example

**This example is a modification of D. C. Rapaport's example "pr_2_1" in book "*The Art of Molecular Dynamics Simulation.* 2nd Edition" [2]

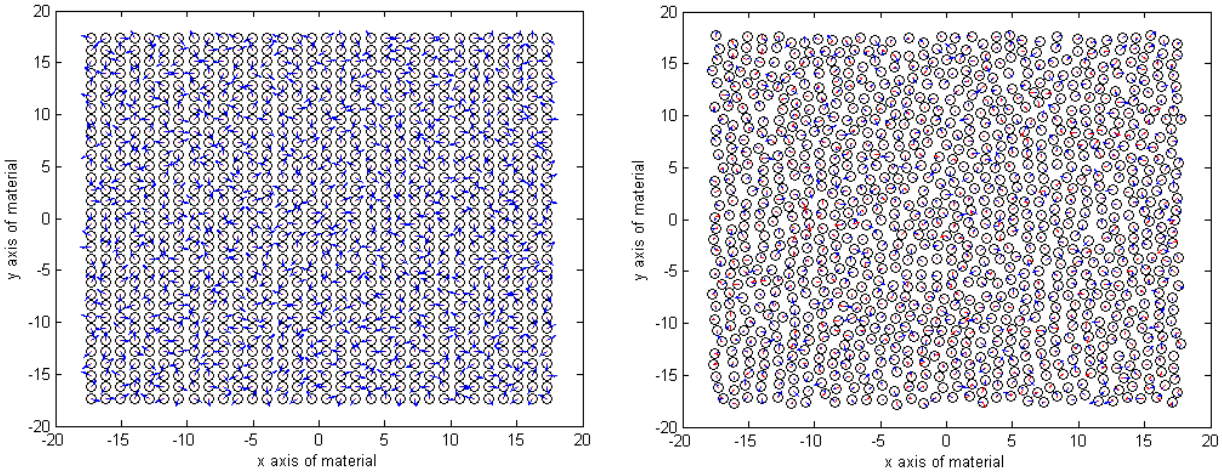
In this example, there is a comparison between Leapfrog and Analog Equation Method in a simple unit cell of 1024 molecules in square lattice. Generally, the energy drift rate depends on integration method, potential function, value of Δt and temperature. A series of measurements are carried out in order to be investigated the accuracy of AEM during the simulation. Lenard-Jones potential function is used for interaction between the molecules with parameters $\sigma^* = 1$ and $\varepsilon^* = 1$. Input data to the calculation are as follows:

Table 8 Input data to molecular dynamics problem (Unit Cell of 1024 molecules)

Mass of atoms	$m^* = {m}/{M}$	1
Density	$\rho^* = {\rho \cdot \sigma^3}/{M}$	0.8
Number of Atoms	N_m	1024
Temperature	$T^* = {k_B \cdot T}/{\varepsilon}$	1
Time step	$dt^* = dt \cdot \sigma^{-1} \cdot \sqrt{{\varepsilon}/{M}}$	0.00125, 0.0025, 0.005, 0.01, 0.02
Number of Steps	-	160000, 80000, 40000, 20000, 10000
Total Time	$t^* = t \cdot \sigma^{-1} \cdot \sqrt{{\varepsilon}/{M}}$	200
Total Energy	$E^* = E \cdot \varepsilon^{-1}$	-

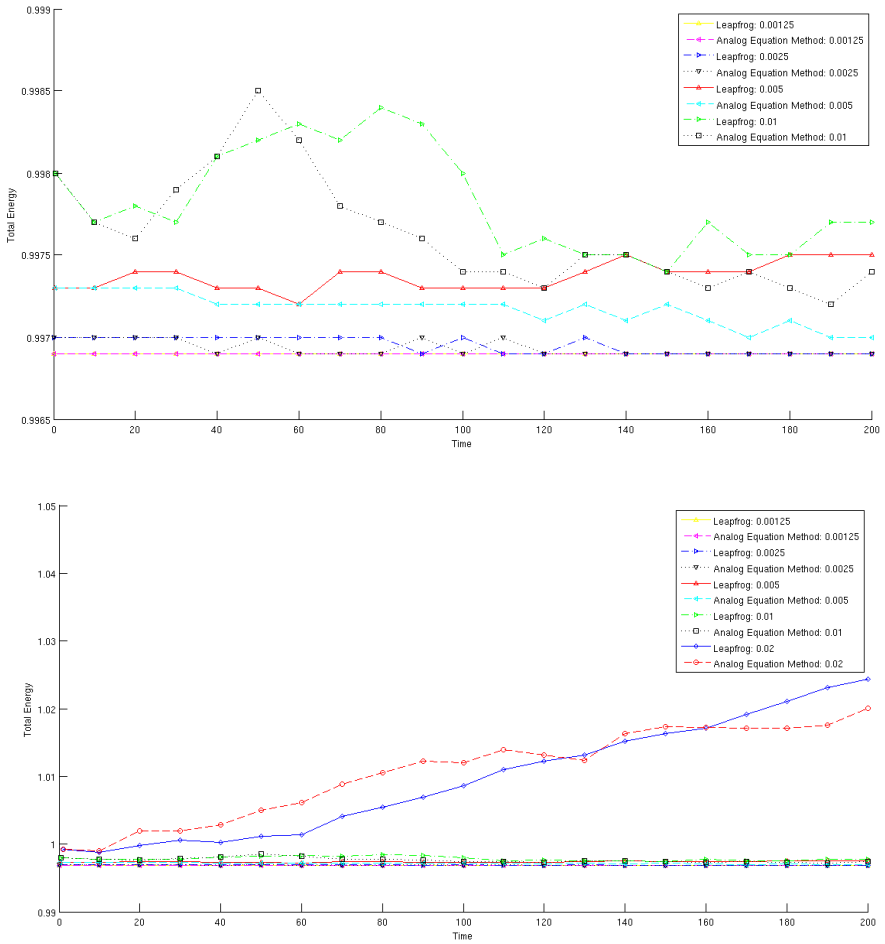
The initial state is a square lattice, so that the number of molecules is $N_m = 32 \cdot 32 = 1024$, and computations are carried out in 64-bit (double) precision. In **Fig. 5** is shown the unit cell of 1024 molecules in initial state and after a 100 time steps.

Fig. 5 Unit Cell of 1024 molecules in initial state (Left) and after a 100 time steps (Right). Velocity and acceleration vectors are represented by blue and red vectors respectively



The results are shown in **Fig. 6**, where the two methods are very close to each other and allows the same time step for a given degree of energy conservation.

Fig. 6 Total energy drifts for different values of Δt with Leapfrog and Analog Equation Method



To emphasize the accuracy of the method, from the energy point of view, most of these results are repeated in **Table 9**.

Table 9 Energy conservation for Leapfrog and Analog Equation Method

Δt:	**0.00125**		**0.0025**		**0.005**		**0.01**		**0.02**	
t	**LF**	**AEM**	**LF**	**AEM**	**LF**	**AEM**	**LF**	**AEM**	**LF**	**AEM**
10	0.9969	0.9969	0.9970	0.9970	0.9973	0.9973	0.9977	0.9977	0.9988	0.9990
50	0.9969	0.9969	0.9970	0.9969	0.9973	0.9972	0.9981	0.9981	1.0002	1.0029
100	0.9969	0.9969	0.9969	0.9970	0.9973	0.9972	0.9983	0.9976	1.0069	1.0123
150	0.9969	0.9969	0.9969	0.9971	0.9975	0.9971	0.9975	0.9975	1.0152	1.0163
200	0.9969	0.9969	0.9969	0.9969	0.9975	0.9970	0.9977	0.9972	1.0231	1.0176

In order to be investigated which method is more accurate, mean square error is obtained from the supposing exact solution of 0.9969 for time step 0.00125. As it is shown in **Table 10** AEM is slightly more accurate than Leapfrog.

Table 10 Total Energy Mean Square Error for Leapfrog and Analog Equation Method

Total Energy Exact Solution = 0.9969										
Δt:	**0.00125**		**0.0025**		**0.005**		**0.01**		**0.02**	
Method	LF	AEM	LF	AEM	LF	AEM	LF	AEM	LF	AEM
MSE (Mean Square Error)	0	0	5.2381e-09	3.3333e-09	8.2381e-08	2.2905e-08	9.5048e-07	6.4333e-07	2.2746e-04	2.2949e-04

In **Fig. 7** and **Table 11** are presented the computation time for these methods. Again, the two methods illustrate almost the same results.

Fig. 7 Computation time, until $t^* = 20$, in Example 1 for Leapfrog and Analog Equation Method

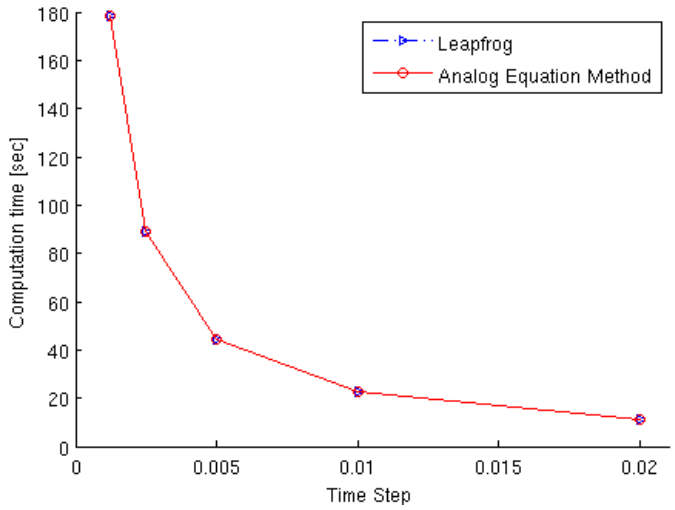


Table 11 Computation time, until $t^* = 20$, in Example 1 for Leapfrog and Analog Equation Method

Δt:	**0.00125**		**0.0025**		**0.005**		**0.01**		**0.02**	
Method	**LF**	**AEM**	**LF**	**AEM**	**LF**	**AEM**	**LF**	**AEM**	**LF**	**AEM**
t [sec]	178.349	178.237	89.125	89.050	44.583	44.574	22.346	22.345	11.234	11.234

Example 3: Analog Equation Method in molecular dynamics problem (Large molecular dynamics systems)

* A C program has been written for the following example

**This example is a modification of D. C. Rapaport's example "pr_2_1" in book "*The Art of Molecular Dynamics Simulation.* 2nd Edition" [2]

In this example (as an extension of example 2), AEM is applied to large molecular dynamics systems. In **Fig. 8** and **Table 12** is illustrated the computation time per time step increasing the number of molecules. For both small and large molecular dynamics systems AEM is slightly faster than Leapfrog.

Fig. 8 Computation time per time step in Example 2 for Leapfrog and Analog Equation Method (Δt=0.005)

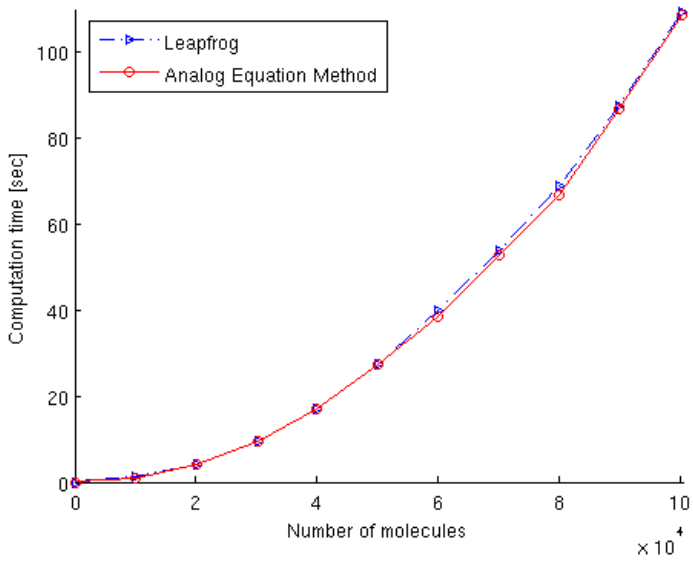


Table 12 Computation time per time step in Example 2 for Leapfrog and Analog Equation Method (Δt=0.005)

t [sec]		
N	Method	
	LF	AEM
10000	1.078	1.046
20164	4.186	4.118
30276	9.661	9.630
40000	17.022	16.950
50176	27.495	27.317

t [sec]		
N	Method	
	LF	AEM
60025	39.950	38.532
70225	53.910	53.016
80089	68.935	66.687
90000	87.651	86.893
100489	109.534	108.655

CONCLUSION

A comparison between direct time integration algorithms, both explicit and implicit, is presented for the solution of the equations of motion for linear finite element as well as molecular dynamics problems. The comparison is performed regarding the fundamental requirements of accuracy, stability, simplicity, speed and minimization of computing resources. Verlet Leapfrog and Velocity Leapfrog algorithms are applied to simple multi-degree of freedom FEM problems giving the advance of speed and minimum computing resources but they only be stable for small values of time step. However, the new explicit method, Analog Equation Method, are more accurate and faster than the other referred methods for structural dynamics problems. In addition, AEM is applied successfully in molecular dynamics problems comparing it with the Leapfrog algorithm. The two methods seems to be equivalent with regards to the accuracy and speed.

REFERENCES

[1] **L, Verlet. (1967)** *"Computer "Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules".*: Physical Review, Vol. 159, pp. 98-103.

[2] **Rapaport, D.C. (2004)** *The Art of Molecular Dynamics Simulation. 2nd Edition.* : Cambridge University Press, 978-0521825689.

[3] **Bathe, K.J. (1996)** *Finite Element Procedures.*: Prentice Hall, 978-0133014587.

[4] **Katsikadelis, J.T (1994)** *The Analog Equation Method - A Powerful BEM-based Solution Technique for Solving Linear and Nonlinear Engineering Problems.*: Boundary Element Method XVI, pp. 167-182.

[5] **Katsikadelis, J.T. (2013)** *A new direct time integration method for the equations of motion in structural dynamics.*: ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik, Vol. 94, pp. 757-774.

[6] **Dokainish, M.A. and Subbaraj, K. (1989)** *A syrvey of direct time-integration methods in computational structural dynamics-I. Explicit methods.*: Computers & Structures, Pergamon Press plc, vol. 32, pp. 1371-1386

[7] **Subbaraj, K. and Dokainish, M.A. (1989)**, *A survey of direct time integration methods in computational structural dynamics-II. Implicit methods.*: Computers & Structures, Pergamon Press plc, vol. 32, pp. 1387-1401