Structural Engineering and Mechanics, Vol. 4, No. 6 (1996) 589-600
DOI: http://dx.doi.org/10.12989/sem.1996.4.6.589

On a new fourth order self-adaptive time integration algorithm

Wanxie Zhong†

Research Institute of Engineering Mechanics, Dalian University of Technology, Dalian, P.R. China

Jianping Zhu‡

Department of Mathematics and Statistics, Mississippi State University, MS 39762, U.S.A.

Abstract. An explicit 4th order time integration scheme for solving the convection-diffusion equation is discussed in this paper. A system of ordinary differential equations are derived first by discretizing the spatial derivatives of the relevant PDE using the finite difference method. The integration of the ODEs is then carried out using a 4th order scheme and a self-adaptive technique based on the spatial grid spacing. For a non-uniform spatial grid, different time step sizes are used for the integration of the ODEs defined at different spatial points, which improves the computational efficiency significantly. A numerical example is also discussed in the paper to demonstrate the implementation and effectiveness of the method.

Key words: time integration; differential equations; self-adaptive method; numerical analysis.

1. Introduction

Traditionally, time dependent partial differential equations are solved using finite difference tim-step integration algorithms (Press, Teukolsky, Vetterling and Flannery 1992). The subdomain precise time integration method (Zhong 1995) uses semi-analytic method to integrate over the time dimension precisely to the machine precision (Zhong 1993). This algorithm not only improved solution accuracy, but also the stability property of the integration algorithms. For example the single point FTCS and leap-frog type precise time integration algorithms are unconditionally-stable when applied to the linear heat equation (Zhong and Zhu 1994), while the traditional finite difference FTCS is only conditionally stable and the finite difference leap-frog scheme is unconditionally unstable.

In this paper, we derive a 4th order time integration scheme that is based on the single point sub-domain precise time integration method. The new algorithm is explicit, but only conditionally stable. When a large time step Δt or a fine spatial grid with a small Δx is used, the algorithm will become unstable. To satisfy the stability criterion, the step size in time dimension may be too small to be computationally efficient. We propose to use adaptive grid to improve computational efficiency.

There has been active research in adaptive method for numerical solution of differential equations, particularly in spatial grid to resolve fast changing solutions in the space (Zienkiewicz

† Professor

‡ Associate Professor

and Taylor 1991). For adaptive methods in the temporal dimension, most of the methods proposed use the same time step Δt for the integration of equations defined at all spatial grid points (Zienkiewicz and Xie 1991, Zeng, Wiberg and Li 1992, Choi amd Chung 1994, Wang 1994). For non-uniform spatial grids, this method may be unnecessarily expensive since the tiny step size determined by the smallest spatial grid spacing is not necessary for most points in the spatial grid.

With explicit time integration, the equations defined at different spatial grid points are integrated independently, providing the possibility of using different time steps for the equations defined at different spatial grid points based on the spatial grid spacing at those points. The use of this type of self-adaptive time-space integration algorithm makes the precise time integration algorithm more efficient.

2. Sub-domain precise time integration of linear systems

The precise time integration is based on the semi-analytical method which discretizes the spatial dimensions of a given time dependent PDE, such as

$$u_t = \nabla \cdot (a \nabla u) \tag{1}$$

to generate

$$\dot{\boldsymbol{u}} = \boldsymbol{H}\boldsymbol{u} + \boldsymbol{f}(t) \tag{2}$$

where $\boldsymbol{u}$ is the vector containing the values of u defined on a discrete spatial grid. To solve Eq. (2) we know from the theory of ordinary differential equations that it is important to get the solution of the corresponding homogeneous equation

$$\dot{\boldsymbol{v}} = \boldsymbol{H}\boldsymbol{v}. \tag{3}$$

Assume that the coefficient a in Eq. (1) does not depend on time t, then $\boldsymbol{H}$ is a matrix whose elements are independent of t. Therefore, the general solution of Eq. (3) can be written as

$$\boldsymbol{v} = \exp(\boldsymbol{H} \cdot t) \cdot \boldsymbol{v}^{(0)} \tag{4}$$

where $\boldsymbol{v}^{(0)}$ is the initial state. Let the length of a time step be Δt and $\boldsymbol{T} = \exp(\boldsymbol{H} \cdot \Delta t)$, then at the time integration points $t_0 = 0$, $t_1 = \Delta t$, $\cdots$, $t_k = k \cdot \Delta t$, we have

$$\boldsymbol{v}(\Delta t) = \boldsymbol{v}^{(1)} = \boldsymbol{T} \cdot \boldsymbol{v}^{(0)}, \tag{5}$$

$$\vdots$$

$$\boldsymbol{v}(k\Delta t) = \boldsymbol{v}^{(k)} = \boldsymbol{T} \cdot \boldsymbol{v}^{(k-1)}, \tag{6}$$

where $\boldsymbol{v}^{(k)}$ is the solution vector at the time $t_k = k\Delta t$. It is clear that the key for the accurate computation of solution $\boldsymbol{v}^{(k)}$ is to determine a precise matrix $\boldsymbol{T}$. Once $\boldsymbol{T}$ is calculated, the solutions can be obtained by repeated matrix-vector multiplications. An efficient algorithm for computing matrix exponential function was given in (Zhong and Yang 1991). The method is based on the relation

$$\exp(\boldsymbol{H} \cdot \Delta t) = \left[\exp\left(\boldsymbol{H} \cdot \frac{\Delta t}{m}\right)\right]^m = [\exp(\boldsymbol{H} \cdot \tau)]^m \tag{7}$$

where m can be selected as an integer power of 2, i.e., $m=2^M$. If m is large enough, then $\tau=\Delta t/m$ will be too small to cause any significant truncation error. We can therefore use the truncated Taylor expansion to approximate $\exp(\boldsymbol{H}\cdot\tau)$ as

$$\exp(\boldsymbol{H}\cdot\tau)\approx \boldsymbol{I}+\boldsymbol{H}\tau+(\boldsymbol{H}\tau)^2/2\,!+(\boldsymbol{H}\tau)^3/3\,!+(\boldsymbol{H}\tau)^4/4\,!=\boldsymbol{I}+\boldsymbol{T}_a. \tag{8}$$

With a small τ, the expansion should give a very good approximation to $\exp(\boldsymbol{H}\cdot\tau)$. Since the elements in matrix $\boldsymbol{T}_a$ is very small as compared to the unity in the identity matrix $\boldsymbol{I}$, they should be stored during the computations separately for better numerical accuracy, rather than added to the identity matrix $\boldsymbol{I}$. Otherwise, the accuracy will be lost due to the round-off errors.

With Eq. (8), matrix $\boldsymbol{T}$ can be computed as

$$\boldsymbol{T}=(\boldsymbol{I}+\boldsymbol{T}_a)^{2^M}=(\boldsymbol{I}+\boldsymbol{T}_a)^{2^{M-1}}\times(\boldsymbol{I}+\boldsymbol{T}_a)^{2^{M-1}}=\cdots \tag{9}$$

using only M matrix multiplications. Note that for any matrices $\boldsymbol{T}_b$ and $\boldsymbol{T}_c$, we have

$$(\boldsymbol{I}+\boldsymbol{T}_b)(\boldsymbol{I}+\boldsymbol{T}_c)=\boldsymbol{I}+\boldsymbol{T}_b+\boldsymbol{T}_c+\boldsymbol{T}_b\times\boldsymbol{T}_c. \tag{10}$$

Therefore, the matrix computation expressed by Eq. (9) can be accomplished by the following algorithm:

```
Compute $T_a$ from Eq. (8)
do $i=1, M$
    $T_a=2T_a+T_a\times T_a$                (11)
end do
$T=I+T_a$                                   (12)
```

Eqs. (8), (11) and (12) give the algorithm for computing the exponential matrix $\boldsymbol{T}$. For the inhomogeneous system (2), assume that the inhomogeneous term $\boldsymbol{f}(t)$ is linear within a time step (t_k, t_{k+1}), then we have

$$\dot{\boldsymbol{u}}=\boldsymbol{H}\boldsymbol{u}+\boldsymbol{r}_0+\boldsymbol{r}_1(t-t_k), \tag{13}$$

where $\boldsymbol{r}_0$ and $\boldsymbol{r}_1$ are known vectors. This equation can be solved using the superposition principle. Let $\boldsymbol{\Phi}(t-t_k)$ be the fundamental solution of the corresponding homogeneous Eq. (3), i.e.,

$$\dot{\boldsymbol{\Phi}}=\boldsymbol{H}\boldsymbol{\Phi},\qquad \boldsymbol{\Phi}(0)=\boldsymbol{I},$$

then the solution of Eq. (13) can be written as

$$\boldsymbol{u}=\boldsymbol{\Phi}(t-t_k)\cdot[\boldsymbol{u}^{(k)}+\boldsymbol{H}^{-1}(\boldsymbol{r}_0+\boldsymbol{H}^{-1}\boldsymbol{r}_1)]-\boldsymbol{H}^{-1}[\boldsymbol{r}_0+\boldsymbol{H}^{-1}\boldsymbol{r}_1+\boldsymbol{r}_1\cdot(t-t_k)]. \tag{14}$$

In the numerical solution process, we do not need the analytic form of $\boldsymbol{\Phi}$, only the matrix $\boldsymbol{\Phi}(t_{k+1}-t_k)=\boldsymbol{\Phi}(\tau)=\boldsymbol{T}$ is needed to calculate $\boldsymbol{u}^{(k+1)}$ from Eq. (14). Therefore, the precise time integration formula for the inhomogeneous Eq. (13) can be given as

$$\boldsymbol{u}^{(k+1)}=\boldsymbol{T}[\boldsymbol{u}^{(k)}+\boldsymbol{H}^{-1}(\boldsymbol{r}_0+\boldsymbol{H}^{-1}\boldsymbol{r}_1)]-\boldsymbol{H}^{-1}[\boldsymbol{r}_0+\boldsymbol{H}^{-1}\boldsymbol{r}_1+\boldsymbol{r}_1\cdot\tau]. \tag{15}$$

While this precise time integration algorithm is accurate to machine precision, it becomes computationally expensive for large systems due to full matrix multiplication and inversions. The sub-domain precise time integration method was introduced to improve computational efficiency. For simplicity of discussions for the subdomain precise time integration, we consider the one-dimensional case of Eq. (1) with $a=1$ and the following boundary conditions:

$$u(0,\ t)=0, \qquad u(2,\ t)=0, \tag{16}$$

If the spatial derivative is discretized using the central finite difference scheme, we will have a set of ODEs

$$\dot{u}_i=\frac{1}{\Delta x^2}(u_{i-1}-2u_i+u_i), \qquad i=1,\ \cdots,\ N-1. \tag{17}$$

Eq. (17) can be written as a matrix equation

$$\dot{\boldsymbol{u}}=\boldsymbol{A}\boldsymbol{u} \tag{18}$$

where $u=\{u_1,\ u_2,\ \cdots,\ u_{N-1}\}^T$ and A is an $(N-1)\times(N-1)$ tri-diagonal matrix with -2 in the main diagonal and 1 in the sub- and super-diagonals. With $N=20$, Eq. (18) can be solved easily by the precise time integration method since it is a small problem.

The basic idea of the subdomain precise time integration is to apply the integration process to only a few rows of the system equations in (18) at a time, instead of the complete system. For example, the 3-point subdomain integration will treat three equations of (18) simultaneously. For an interior point i, we will have

$$\begin{Bmatrix} \dot{u}_{i-1} \\ \dot{u}_i \\ \dot{u}_{i+1} \end{Bmatrix}=\frac{1}{\Delta x^2}\begin{bmatrix} 1 & -2 & 1 & & \\ & 1 & -2 & 1 & \\ & & 1 & -2 & 1 \end{bmatrix}\begin{Bmatrix} u_{i-2} \\ u_{i-1} \\ u_i \\ u_{i+1} \\ u_{i+2} \end{Bmatrix}, \quad i=2,\ \cdots,\ N-2 \tag{19}$$

or

$$\begin{Bmatrix} \dot{u}_{i-1} \\ \dot{u}_i \\ \dot{u}_{i+1} \end{Bmatrix}=\frac{1}{\Delta x^2}\begin{bmatrix} -2 & 1 & \\ 1 & -2 & 1 \\ & 1 & -2 \end{bmatrix}\begin{Bmatrix} u_{i-1} \\ u_i \\ u_{i+1} \end{Bmatrix}+\frac{1}{\Delta x^2}\begin{Bmatrix} u_{i-2} \\ 0 \\ u_{i+2} \end{Bmatrix}, \quad i=2,\ \cdots,\ N-2. \tag{20}$$

This equation system can be considered as a system of three ODEs with an inhomogeneous term. Assume we have finished calculation at time step n, then $u_i^{(n)}$, $i=1,\ \cdots,\ N-1$, are all known. To calculate $u_i^{(n+1)}$, $i=1,\ \cdots,\ N-1$, we have to integrate Eq. (20). When the subdomain precise time integration is applied to Eq. (20), we have to approximate the inhomogeneous term $\{u_{i-2},\ 0,\ u_{i+2}\}^T$. An explicit approximation would be to use the values $u_{i-2}^{(n)}$ and $u_{i+2}^{(n)}$ from the previous time step.

Having approximated the inhomogeneous term in Eq. (20), we can write it as

$$\dot{v}_i=Hv_i+f_i^{(n)}, \tag{21}$$

$$v_i=\{u_{i-1},\ u_i,\ u_{i+1}\}^T, \qquad i=2,\ 3,\ \cdots,\ N-2.$$

Let $T=\exp(H\cdot\Delta t)$ which can be calculated accurately using the method just discussed, we will have, based on Eq. (15),

$$v_i^{(n+1)}=Tv_i^{(n)}+(T-I)H^{-1}f_i^{(n)}, \tag{22}$$

with

$$f_i^{(n)}=\frac{1}{\Delta x^2}\{u_{i-2}^{(n)},\ 0,\ u_{i+2}^{(n)}\}^T.$$

This algorithm is first order accurate in time.

To improve the accuracy in the time dimension, multi-step time integration can be easily incorporated into the subdomain precise time integration method. For example, if a leap-frog type algorithm is used in the subdomain precise time integration method, we will have

$$v_i^{(n+1)}=Tv_i^{(n-1)}+(T-I)H^{-1}f_i^n \tag{23}$$

with $T(2\Delta t)=\exp(2H\Delta t)$ which is very similar to the results given by Eq. (22). Note that for the first step when $n=0$, a one step forward integration given by Eq. (22) must be used to calculate $u_i^{(1)}$, $i=1, \cdots, N-1$, in order to use the leap-frog method at later time steps.

In the extreme case, we can use only one interior point in each subdomain to simplify the computation and analysis of the algorithms. For the i-th subdomain, with only one interior point, we have

$$\dot{u}_i=\frac{1}{\Delta x^2}(u_{i-1}-2u_i+u_{i+1})=-\frac{2}{\Delta x^2}u_i+\frac{1}{\Delta x^2}(u_{i-1}+u_{i+1}),\ i=1,\ \cdots,\ N-1. \tag{24}$$

Assume $u_i^{(n)}$, $i=0, 1, \cdots, N$, are all known, the one-point explicit subdomain precise time integration of Eq. (24) leads to

$$u_i(t)=Ce^{-2\Delta t/\Delta x^2}+\frac{u_{i-1}^{(n)}+u_{i+1}^{(n)}}{2}, \qquad t_n\le t\le t_{n+1}. \tag{25}$$

Using the conditions $u_i(t_n)=u_i^{(n)}$, $u_i(t_{n+1})=u_i(t_n+\Delta t)=u_i^{(n+1)}$, we have

$$u_i^{(n+1)}=\left(u_i^{(n)}-\frac{u_{i-1}^{(n)}+u_{i+1}^{(n)}}{2}\right)e^{-2\Delta t/\Delta x^2}+\frac{u_{i-1}^{(n)}+u_{i+1}^{(n)}}{2}. \tag{26}$$

Eq. (26) is similar to the FTCS finite difference scheme. When Δt is small, we can use a first order approximation for $e^{-2\Delta t/\Delta k^2}$ to get

$$e^{-2\Delta t/\Delta x^2}\approx 1-\frac{2\Delta t}{\Delta x^2}. \tag{27}$$

Substituting Eq. (27) back into Eq. (26), we get

$$u_i^{(n+1)}=u_i^{(n)}+\frac{\Delta t}{\Delta x^2}(u_{i-1}^{(n)}-2u_i^{(n)}+u_{i+1}^{(n)}) \tag{28}$$

which is exactly the FTCS finite difference scheme. It is well-known that for the FTCS scheme given in Eq. (28), the stability condition is

$$\frac{\Delta t}{\Delta x^2}\le\frac{1}{2} \tag{29}$$

which could be very restrictive for a fine spatial grid with small Δx.

On the other hand, for the scheme given by Eq. (26), we can use the Von Neumann stability analysis to get the stability criterion as

$$|\lambda+(1-\lambda)\cos\theta|\leq 1 \tag{30}$$

where $\lambda=e^{-2\Delta t/\Delta x^2}$ and $0<\lambda\leq 1$. It is not difficult to verify that the inequality in Eq. (30) is always satisfied for any λ and θ. Therefore, the explicit one-point subdomain precise time integration given by Eq. (26) is unconditionally stable.

For the leap-frog finite difference scheme

$$u_i^{(n+1)}=u_i^{(n-1)}+\frac{2\Delta t}{\Delta x^2}(u_{i-1}^{(n)}-2u_i^{(n)}+u_{i+1}^{(n)}), \tag{31}$$

it is also well known that the stability requirement is

$$\left|\frac{-b\pm\sqrt{b^2+4}}{2}\right|\leq 1 \tag{32}$$

where $b=\frac{8\Delta t}{\Delta x^2}\sin^2\frac{\theta}{2}$. If $\sin^2\frac{\theta}{2}\neq 0$, it is obvious that the inequality in Eq. (32) cannot be satisfied. Therefore, the method is unconditionally unstable.

With the one-point precise time integration method, a similar three level formula can be obtained from Eq. (24) by integrating from t_{n-1} to t_{n+1} and setting $\frac{1}{\Delta x^2}(u_{i-1}+u_{i+1})$ to be $\frac{1}{\Delta x^2}(u_{i-1}^{(n)}+u_{i+1}^{(n)})$:

$$u_i^{(n+1)}=\left[u_i^{(n-1)}-\frac{u_{i-1}^{(n)}+u_{i+1}^{(n)}}{2}\right]e^{-4\Delta t/\Delta x^2}+\frac{u_{i-1}^{(n)}+u_{i+1}^{(n)}}{2} \tag{33}$$

Again, using a first-order approximation for $e^{-4\Delta t/\Delta x^2}$ will lead to the leap-frog finite difference scheme of Eq. (31). The Von Neumann stability analysis of Eq. (33) shows that the convergence criterion

$$\left|\frac{1}{2}\left[(1-\lambda^2)\cos\theta\pm\sqrt{(1-\lambda^2)^2\cos^2\theta+4\lambda^2}\right]\right|\leq 1, \tag{34}$$

where $\lambda=e^{-4\Delta t/\Delta x^2}$, is satisfied unconditionally. Therefore, the one-point subdomain precise time integration algorithm given by Eq. (33) is also unconditionally stable.

3. The high order single point precise time integration method

For simplicity, we use the following one-dimensional diffusion-convection equation

$$u_t=Du_{xx}-Cu_x. \tag{35}$$

where the diffusion and convection coefficients D and C are constants. A spatial discretization of Eq. (53) can be derived as follows:

Since Eq. (35) has two obvious solutions $u=$constant and $u=x-C_t$ which is a wave propagating at a velocity C, a good spatial discretization scheme must admit these two solutions. Thus we have

$$\dot{u}_i = c_1 u_{i-1} - c_2 u_i + c_3 u_{i+1}, \qquad i = 1, \cdots, N-1 \tag{36}$$

with

$$\begin{aligned} c_1 &= \left(\frac{C}{2} + c_d\right)/\Delta x_i, \\ c_3 &= \left(-\frac{C}{2} + c_d\right)/\Delta x_{i+1}, \\ c_2 &= c_1 + c_3, \\ \Delta x_i &= x_i - x_{i-1}, \ \Delta x_{i+1} = x_{i+1} - x_i. \end{aligned} \tag{37}$$

Since Δx_i's, $i = 1, 2, \cdots, N$, are usually not equal, the coefficients c_1, c_2 and c_3 are different for different index i. The third constant c_d can be determined by requiring that the scheme Eq. (36) contain the asymptotic steady state solution of Eq. (35) which is $u = e^{Cx/D}$, thus obtaining

$$\begin{aligned} p_i &= C\Delta x_i/D, \ p_{i+1} = C\Delta x_{i+1}/D, \\ f_i &= [1 - \exp(-p_i)]/\Delta x_i, \\ g_{i+1} &= [\exp(p_{i+1}) - 1]/\Delta x_{i+1}, \\ c_d &= \frac{C}{2}(g_{i+1} + f_i)/(g_{i+1} - f_i). \end{aligned} \tag{38}$$

This spatial discretization is upwinding since $c_1 > c_3 > 0$ if $C > 0$.

The single point subdomain precise time integration can be applied to Eq. (36) easily. If we re-write Eq. (36) as

$$\dot{u}_i + c_2 u_i = c_1 u_{i-1}^n + c_3 u_{i+1}^n \tag{39}$$

where we have assumed that the right hand side terms are assigned the values from the previous time step, then Eq. (39) can be integrated explicitly using the initial condition $u_i = u_i^{n-1}$ at $t = t_{n-1}$. The single point leap-frog type algorithm will lead to

$$u_i^{(n+1)} = \left[u_i^{(n-1)} - \frac{c_3 u_{i+1}^{(n)} + c_1 u_{i-1}^{(n)}}{c_2}\right] e^{-2c_2\Delta t} + \frac{c_3 u_{i+1}^{(n)} + c_1 u_{i-1}^{(n)}}{2} \tag{40}$$

This algorithm is explicit and has second order of accuracy in time. Using the similar Von Neumann analysis as we did for the one-dimensional heat equation, we can also prove that scheme Eq. (40) is unconditionally stable.

To further improve the accuracy in time, we use a higher order approximation for the right hand side terms of Eq. (36). Using the second order Taylor expansion for u_{i-1} and u_{i+1} in Eq. (36), we obtain

$$\dot{u}_i + c_2 u_i = c_1 u_{i-1}^n + c_3 u_{i+1}^n + (c_1 \dot{u}_{i-1}^n + c_3 \dot{u}_{i+1}^n)(t - t_n) + (c_1 \ddot{u}_{i-1}^n + c_3 \ddot{u}_{i+1}^n)(t - t_n)^2/2. \tag{41}$$

The first and second order derivatives in the right hand side of Eq. (41) can be approximated by

$$\dot{u}_j^n = c_1 u_{j-1}^n - c_2 u_j^n + c_3 u_{j+1}^n, \qquad j = i-1,\ i+1, \tag{42}$$

and

$$\ddot{u}_j^n = 2[\dot{u}_j^n - (u_j^n - u_j^{n-1})/\Delta t]/\Delta t, \qquad j = i-1,\ i+1 \tag{43}$$

Integrating Eq. (41) from t_n to t_{n+1}, we obtain the following algorithm:

$$\begin{aligned} u_i^{n+1} = {} & [u_i^{n-1} - (c_1 u_{i-1}^n + c_3 u_{i+1}^n)/c_2]\,\lambda^2 + (c_1 u_{i-1}^n + c_3 u_{i+1}^n)/c_2 \\ & + [(c_1 \dot{u}_{i-1}^n + c_3 \dot{u}_{i+1}^n)/c_2]\cdot\left[\left(1 - \frac{1}{\xi}\right) + \lambda^2\left(1 + \frac{1}{\xi}\right)\right]\Delta t \\ & + [(c_1 \ddot{u}_{i-1}^n + c_3 \ddot{u}_{i+1}^n)/c_2]\cdot\left[\frac{1}{2} - \frac{1}{\xi} + \frac{1}{\xi^2} - \lambda^2\left(\frac{1}{2} - \frac{1}{\xi} + \frac{1}{\xi^2}\right)\right](\Delta t)^2 \end{aligned} \tag{44}$$

where $\xi = c_2 \Delta t$, $\lambda^2 = \exp(-2\xi)$. This is an explicit algorithm. Note that Δt always appears in the algorithm in the form of $\xi = c_2 \Delta t$ which is a non-dimensional parameter. Although in Eq. (44) Δt appears as a separate parameter, it actually still can be put in the form of $\xi = c_2 \Delta t$ if we substitute Eqs. (42) and (43) for $\dot{u}^n$ and $\ddot{u}^n$. When ξ is small, we have

$$\left(1 - \frac{1}{\xi}\right) + \lambda^2\left(1 + \frac{1}{\xi}\right) \approx \frac{2}{3}\xi^2 - \frac{2}{3}\xi^3,$$

and

$$\left(\frac{1}{2} - \frac{1}{\xi} + \frac{1}{\xi^2}\right) - \lambda^2\left(\frac{1}{2} + \frac{1}{\xi} + \frac{1}{\xi^2}\right) \approx \frac{1}{3}\xi - \frac{1}{3}\xi^2.$$

Substituting these expansions back into Eq. (44), we can show that the last two terms are of the orders of ξ^3 and ξ^4. Therefore, the leap-frog type algorithm Eq. (40) is second order accurate. To prove that algorithm Eq. (44) is 4th order accurate, we need to take more terms in both Eq. (41) and the expansion of ξ in Eq. (44). It can be shown that the truncation error in Eq. (44) is of the order of ξ^5 and higher. Therefore, it has 4th order of accuracy. The detailed proof is lengthy, though straightforward, so we omit the proof here.

Although the high precision of the algorithm given by Eq. (44) is attractive, it is no longer unconditionally stable because of the use of the expansions in the right hand side of Eq. (41). When $\xi > 1.5$, the algorithm is unstable. In a non-uniform spatial grid, the straightforward application of this algorithm is not efficient since the step size is dictated by the smallest spatial grid size. Therefore, adaptive time-space integration algorithm becomes necessary.

4. Self-adaptive time-space integration

When solving partial differential equations, the spatial grid is usually non-uniform in order to resolve the fast changing spatial variables efficiently. For explicit schemes that are conditionally stable, the step size of the time integration is dictated by the smallest spatial grid spacing (stability criterion), causing unnecessary small steps for the regions where the spatial grid spacing is large. As a result, the algorithm with uniform time step sizes for all spatial grid points is not computationally efficient.

A better idea would be to use different time step Δt at different spatial grid points. As long as $\xi = c_2 \Delta t < 1$, stability will be maintained. We can then apply the fourth-order precise time integration scheme efficiently to problems with non-uniform spatial grids.

Assume *Delt* is the standard time step size selected for the next step of time integration. It should maintain stability in most of the grid points in the spatial domain. For those spatial grid points where the stability criterion is violated, the step *Delt* can be sub-divided into $\Delta t = Delt/2^p$ ($p=1, 2, \cdots$). After the spatial grid is determined based on the resolution requirement, we can calculate coefficients c_1, c_2, and c_3 for each spatial grid point j. If $\xi = c_2 \cdot Delt < 1$ is not satisfied at point j, then the step *Delt* should be halved repeatedly until the criterion is satisfied, i.e., until $c_2 \cdot Delt/2^{p_j} < 1$ is satisfied. The information 2^{p_j} should be stored in an array for later use:

$$Astp\,[j] = 2^{p_j}. \tag{45}$$

In order to compute u_j^{n+1} based on Eqs. (42)-(44), the following values from the previous two steps are needed:

$$\begin{aligned} &u_i^n, \qquad j-2 \le i \le j+2, \\ &u_i^{n-1}, \qquad j-1 \le i \le j+1, \end{aligned} \tag{46}$$

These values from the neighboring points may not be defined at the sub-divided time steps for point j (note that *Delt* is sub-divided into 2^{p_j} smaller steps to satisfy the stability criterion at point j), or vice verse. Therefore, we need another array to record the finest subdivision of the neighboring points of j:

$$Bstp[j] = 2^{q_j}, \quad q_j = max(p_{j-2}, p_{j-1}, p_j, p_{j+1}, p_{j+2}). \tag{47}$$

Obviously we have $q_j \ge p_j$. Let dta_j and dtb_j represent the step sizes of time integration at point j and that of the neighboring points, respectively. Since Eq. (41) is integrated analytically, the solution value at any point within the step dta_j can be calculated easily. Assume that the integration process has reached $t_n = n * dta_j$ at point j, and that the values in Eq. (46) are available. To advance the time integration to $t_{n+1} = (n+1) * dta_j$, we need to calculate the solution values at the increment of dtb_j since these values will be needed for the integration of the neighboring points. Therefore, there are totally

$$Bstp[j]/Astp[j] = 2^{(q_j - p_j)}$$

substeps in each step dta_j. For the kth substep, the integration formula is

$$\begin{aligned} &dt = k * dtb_j, \quad kc = c_2 * (dta_j + dt), \quad s = 2/c_2, \\ &e = \exp(-kc), \quad dd = (c_1 * u_{j-1}^n + c_3 * u_{j+1}^n)/c_2, \\ &u_j(t_n + dt) = (u_j^{n-1} - dd) * e + dd + ((c_3 * \dot{u}_{j-1}^n + c_3 * \dot{u}_{j-1}^n)/c_2) * \\ &((c_2 * dt - 1) + (c_2 * dta_j + 1) * e)/c_2 + (c_3 * \ddot{u}_{j+1}^n + c_1 * \ddot{u}_{j-1}^n)/ \\ &(2 * c_2) * (dt * (dt - s) + s/c_2 - (dta_j * (dta_j + s) + s/c_2) * e). \end{aligned} \tag{48}$$

where $\dot{u}$ and $\ddot{u}$ can be calculated based on Eqs. (42) and (43) using the data given in Eq. (46).

5. Numerical experiment

We consider Eq. (35) with $D=1$ and $C=14$ in an interval $[0, 2]$. The boundary conditions are

$$u(0, t)=0, \; u(2, t)=20,$$

and the initial condition is

$$u(x, 0)=10x.$$

We divide the spatial interval into 20 subintervals with non-uniform lengthes:

$$\{x_0, \cdots, x_{20}\}=\{0.00, 0.50, 0.10, 0.15, 0.20, 0.30, 0.40, 0.50, 0.60, 0.80, 1.00, 1.20, 1.40, 1.50, 1.60, 1.70, 1.80, 1.85, 1.90, 1.95, 2.00\}$$

The discretization is based on Eqs. (36)-(38). The result from the precise time integration, listed as "precise" in Table 1, is taken as the accurate solution with machine precision. For the single point high order integration based on Eqs. (42)-(44), take $Delt=0.001$, the largest value of $\xi=c_2*Delt$ is 0.8324. The result of integration is given as "highp" in Table 1. When the time step size increases to $Delt=0.002$, the scheme based on Eqs. (42)-(44) becomes unstable.

With self-adaptive time integration, the step size *Delt* is divided into different number of substeps based on the local spatial grid sizes. The two arrays that record the subdivisions are as follows:

Astp[0..20]: 4,4,4,4,2,1,1,1,1,1,1,
1,1,1,1,1,2,4,4,4,4
Bstp[0..20]: 4,4,4,4,4,4,2,1,1,1,1,
1,1,1,2,4,4,4,4,4,4.

Note that the integration step size in the middle of the interval is four times that in the two ends since the grid spacing is large in the middle. The results of this adaptive integration is listed as "adap-hp" in Table 1.

Since a high order integration formula is used in the calculation, the numerical results are very close to the accurate solution. The difference is invisible from curves in a plot. For the self-adaptive formula, although the amount of computation has been reduced significantly, the accuracy was not affected very much, and the algorithm remained stable in the computation. The line of "leapfrg" in Table 1 gives the numerical results corresponding to leap-frog algorithm. The results are not as good as the self-adaptive 4th order scheme.

The computational grid (time and space) of the self-adaptive high order integration scheme for this example is given in Fig. 1, where the points represented by $\circ$ are the integration steps (recorded in the array *Astp*), and the points represented by $*$ are the auxiliary points for the transition between points in which different spatial grid spacing requires different subdivision of time step *Delt* (recorded in the array *Bstp*). The solution values at these auxiliary points should also be calculated during the integration from t_n to t_{n+1}. It is clear from Fig. 1 that the self-adaptive integration algorithm skipped many points in the subdivision which would otherwise be processed if a uniform small time step were used, thus improving the computational efficiency significantly. The results obtained by using a unified small time step for all grid points are represented in Table 1 as "highp".

Table 1 Time history of discretized convective-diffusion equation with $D=1.0$ and $C=14$

	$x=0.1$	$x=0.2$	$x=0.4$	$x=0.6$	$x=1.4$	$x=1.6$	$x=1.8$	$x=1.9$
when $t=0.004$								
precise	0.5707	1.4556	3.4402	5.4400	13.4400	15.4400	17.4408	18.4722
highp	0.5696	1.4540	3.4401	5.4400	13.4400	15.4400	17.4407	18.4719
adap-hp	0.5695	1.4540	3.4400	5.4400	13.4400	15.4400	17.4407	18.4719
Leapfrg	0.6098	1.5061	3.4528	5.4466	13.4425	15.4503	17.4747	18.5509
when $t=0.008$								
precise	0.3560	1.0209	2.8888	4.8803	12.8800	14.8800	16.8875	17.9971
highp	0.3554	1.0198	2.8883	4.8802	12.8800	14.8800	16.8875	17.9970
adap-hp	0.3554	1.0202	2.8847	4.8800	12.8800	14.8800	16.8875	17.9970
Leapfrg	0.3992	1.0913	2.9193	4.8951	12.8849	14.8985	16.9479	18.1159
when $t=0.016$								
precise	0.1608	0.5122	1.9141	3.7782	11.7600	13.7605	15.8020	17.1035
highp	0.1607	0.5119	1.9135	3.7779	11.7600	13.7605	15.8020	17.1035
adap-hp	0.1607	0.5121	1.9105	3.7749	11.7600	13.7605	15.8020	17.1035
Leapfrg	0.1920	0.5760	0.1937	3.8150	11.7693	13.7913	15.8980	17.2654
when $t=0.024$								
precise	0.0805	0.2708	1.2097	2.7721	10.6402	12.6420	14.7333	16.2380
highp	0.0805	0.2707	1.2093	2.7717	10.6402	12.6420	14.7333	16.2380
adap-hp	0.0805	0.2709	1.2095	2.7688	10.6401	12.6420	14.7333	16.2380
Leapfrg	0.0999	0.3151	1.2750	2.8291	10.6541	12.6816	14.8509	16.4232
when $t=0.032$								
precise	0.0429	0.1492	0.7506	1.9462	9.5225	11.5246	13.6726	15.3835
highp	0.0428	0.1492	0.7504	1.9458	9.5225	11.5246	13.6726	15.3835
adap-hp	0.0429	0.1494	0.7515	1.9449	9.5220	11.5245	13.6726	15.3835
Leapfrg	0.0545	0.1779	0.8071	2.0120	9.5418	11.5713	13.8043	15.5831
when $t=0.040$								
precise	0.0238	0.0848	0.4633	1.3247	8.4134	10.4100	12.6163	14.5342
highp	0.0238	0.0848	0.4632	1.3244	8.4133	10.4098	12.6163	14.5342
adap-hp	0.0238	0.0850	0.4645	1.3250	9.4123	10.4094	12.6162	14.5341
Leapfrg	0.0308	0.1029	0.5070	1.3881	8.4393	10.4630	12.7578	14.7435
when $t=0.400$, all the same as precise								
precise	0.0000	0.0000	0.0000	0.0000	0.0045	0.0740	1.2163	4.9320

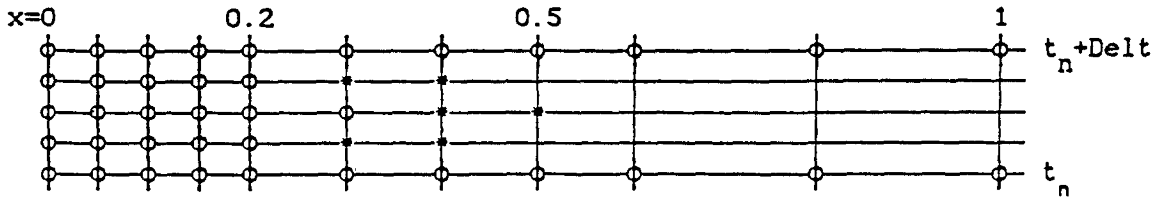

Fig. 1 A non-uniform time-space grid. Time integration points are represented by the open circles, and the auxiliary points are represented by the stars.

The self-adaptive algorithm discussed here can be extended to two- and three-dimensional cases in a straightforward manner. The array *Astp* can still be used to record the number of subdivisions of *Delt* needed to satisfy the stability criterion at each point, and the array *Bstp* is determined by the subdivisions of the neighboring points.

6. Conclusions

A fourth order explicit time integration scheme is presented in this paper. The combination of this algorithm with self-adaptive time steps can improve computational efficiency significantly by using different subdivisions at different spatial points. Application of this algorithm to more complicated practical problems is underway.

References

Choi, C.K. and Chung, H.J. (1994), "An adaptive procedure in finite element analysis of elastodynamic problems", *Proceedings of International Conferenceon Computational Methods in Structural and Geotechnical Engineering*, Hong Kong, December.

Press, W.H., Teukolsky, S.A., Vetterling, W.T. and Flannery, B.P. (1992), *Numerical Recipes*, 2nd ed. Cambridge University Press, New York.

Wang, H.Z. (1994), "A posteriori error estimator of energy and adaptive coordination of spatial-temporal discretization for FEM and direct integration in transient dynamic analysis", *Proceedings of International Conference on Computational Methods in Structural and Geotechnical Engineering*, Hong Kong, December.

Zeng, L.F., Wiberg, N.E. and Li, X.D. (1992), "A posteriori local error estimation and adaptive time-stepping for Newark integration in dynamic analysis", *Earthquake Engineering and Structural Dynamics*, **21**, 555-571.

Zhong, W. (1995), "Subdomain precise time integration and numerical solution of partial differential equations", *Computational Structural Mechanics and Its Applications*, to appear (in Chinese).

Zhong, W.X. (1993), *Computational Mechanics and Optimal Control*, Dalian University Press, Dalian, China.

Zhong, W.X. and Zhu, J.P., (1994), "Rethinking to finite difference time-step integrations", *Proceedings of International Conference on Computational Methods in Structural and Geotechnical Engineering*, Hong Kong, December.

Zienkiewicz, O.C. and Taylor, R.L. (1991), *The Finite Element Method*, 4th ed., McGraw-Hill, New York.

Zienkiewicz, O.C. and Xie, Y.M. (1991), "A simple error estimator and adaptive time stepping procedure for dynamic analysis", *Earthquake Engineering and Structural Dynamics*, **20**, 871-887.

Zhong, W.X. and Yang, Z.S. (1991), "An algorithm for computing the main eigenpairs in time continuous LQ control." *Applied Mathematics and Mechanics*, **12**, 45-50.