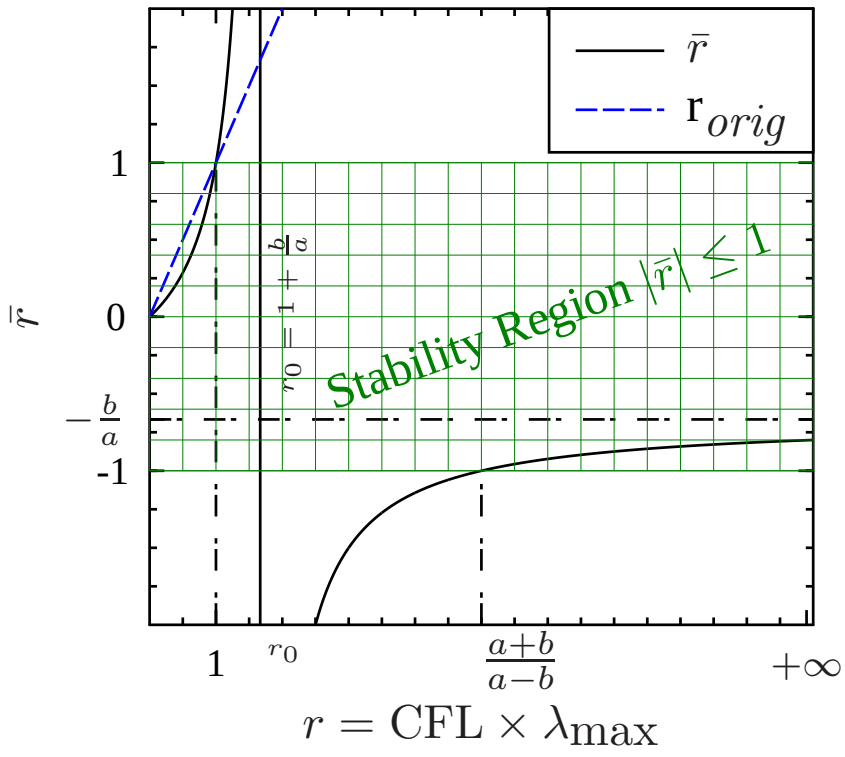
On the Theory of a Novel High-Order Time-Marching Algorithm Based on Picard Iteration

Arash Ghasemi
Kidambi Sreenivas
National Center for Computational Engineering,
University of Tennessee at Chattanooga
701 E. M.L. King Blvd. Chattanooga, TN 37403, USA

First Edition



A mathematician is a person who can find analogies between theorems; a better mathematician is one who can see analogies between proofs and the best mathematician can notice analogies between theories. One can imagine that the ultimate mathematician is one who can see analogies between analogies.

Stefan Banach (1892-1945)

Prologue

In 1893 French mathematician Émile Picard proposed a comprehensive proof for the existence and uniqueness of a solution to $y'(t) = f(t, y(t)) \quad y(t_0) = y_0$ in $t \in [t_0-\varepsilon, t_0+\varepsilon]$[1]. He used iterative integral equation $y_{n+1}(t) = y_0+\int_{t_0}^{t} f(s, y_n(s))\, ds$ in his theorem. Later in 1922, Stefan Banach introduced his famous fixed-point theorem[2] which states the criteria in which the general sequence $y_{n+1} = F(y_n)$ converges to a fixed point in a metric space. Obviously Picard Iteration $(y_{n+1}(t) = y_0 + \int_{t_0}^{t} f(s, y_n(s))\, ds)$, Newton iteration $(y_{n+1} = y_n - f(y_n)/f'(y_n))$ and Mann Iteration $(y_{n+1}(t) = y_0 + \alpha_n y_n(t) + \int_{t_0}^{t} f(s, y_n(s))\, ds)$ are special cases of Banach fixed-point theorem for $F(\square) = y_0 + \int_{t_0}^{t} f(s, \square)ds$, $F(\square) = \square - f(\square)/f'(\square)$, $F(\square) = y_0 + \alpha_n\square + \int_{t_0}^{t} f(s, \square)ds$ respectively. Therefore they are generally classified as fixed point iterations and their convergence is usually evaluated using Banach fixed point theorem.

In this work, we focus on the numerical representation[3] of the analytical iterations mentioned before to find the criteria in which a discretized (numerical) counterpart of fixed-point iteration (mainly Picard iteration) converges to a *numerical fixed-point*[4]. In other word, for the integral equation $y_{n+1}(t) = y_0 + \int_{t_0}^{t} f(s, y_n(s))\, ds$, we replace the analytical integration operator $\int$ with a matrix operator $\mathbb{S}$ containing p^{th}-order numerical approximation (quadrature), yielding the matrix equation $\mathbf{y}_{n+1} = \mathbf{y}_0 + \Delta t\ \mathbb{S}\ \mathbf{f}(\mathbf{y}_n) + \mathcal{O}\left(\Delta t^p\right)$ where $\Delta t = t - 0 = t$. This particular matrix algebraic iteration is studied in the finest detail in Sec.(1.1). This study leads to Theorem (1) where the stability and convergence of such iteration is related to a new parameter which we call it the *stability number* $r = c\lambda\Delta t$ where λ is the eigen-value of the integration operator $\mathbb{S}$ and c is the rate of change of function $df/dy(y(t_0))$ (eq. (1.19)). We will show in Sec.(2.1) that the stability number is expressed in the form of generalized CFL number $r = \lambda$ CFL when Picard iteration is extended to the case of

[1] Sur lapplication des mthodes dapproximations successives ltude de certaines quations diffrentielles ordinaires, Journal de mathmatiques pures et appliques 4 e srie, tome 9 (1893), p. 217-272

[2] Banach, S. "Sur les oprations dans les ensembles abstraits et leur application aux quations intgrales." Fund. Math. 3(1922), 133-181

[3] Because the analytical Picard iteration is studied and applied in the discretized form we call these family of numerical methods Discrete Picard Iteration (DPI).

[4] The numerical fixed point is actually the fixed point obtained using analytical (exact) operator plus a tolerance which we call it truncation error which depends on the order of accuracy of the discretized integration operator(See eq. (1.39) showing a numerical fixed point containing truncation error.)

general partial differential equation[5]. Thus one of the important result of this paper is that:

> The convergence of Picard iteration to the fixed point doesn't depend on the criteria given in Banach fixed point theorem *when iteration is done numerically*. Instead it depends on more limiting conditions including the generalized CFL number in the way that is described in Theorem (2)- Sec.(2.1). Also, the rearrangement operator developed in this paper in Sec.(2.1) and used in Theorem (2) makes this theorem so general that can be applied to scalar/vector ODE and/or PDE in the same time with the same formulation.

In addition, a convergence analysis is performed in Sec.(1.1.1) for basic Discrete Picard Iteration which reveals another important result as follows:

> The Discrete Picard Iteration is exponentially convergent. The base is the stability number and the exponent is the number of iterations. (see eq.(1.46))

Some preliminary implementation of original DPI[6] are brought in Sec.(1.3) and Sec.(2.2) for ODEs and PDEs respectively which prove the feasibility of application of DPI in numerical solution of differential equations. In these numerical solutions, a fairly large CFL number is chosen although the DPI scheme used in these examples is explicit. Thus we conclude another important result:

> The explicit DPI is conditionally stable with CFL limit given by
>
> $$\text{CFL} \leq \frac{1}{\lambda_{\max}}$$
>
> Where $\lambda_{\max}$ is the maximum eigen-value of $\mathbb{S}$. The ratio of $\frac{1}{\lambda_{\max}}$ goes up to 59 (see fig.(1.2.2)) for the integration operators that we designed in Sec.(1.2). Thus explicit DPI remains stable up to CFL ≈ 59. This is an improvement compared to traditional explicit schemes where CFL ≤ 1.

To overcome stability limit of explicit DPI, we investigate some alternative approaches to the original Discrete Picard iteration. In particular, in Sec.(3.2.1) we modify the original DPI to obtain a new implicit scheme. The developed scheme has the following interesting properties:

[5] See eq.(TheoremII)

[6] The original DPI is defined as the discretization of analytical Picard Iteration $y_{n+1}(t) = y_0 + \int_{t_0}^{t} f(s, y_n(s))\, ds$ in the explicit form of $\mathbf{y}_{n+1} = \mathbf{y}_0 + \Delta t\, \mathbb{S}\, \mathbf{f}(\mathbf{y}_n) + \mathcal{O}(\Delta t^p)$ without any modification like changing to implicit form as we do in Sec.(3.2.1).

- It is always more convergent than original DPI.
- It is stable for arbitrarily large and small CFL numbers. However it has a instability hole at asymptote $r = \lambda_{\max} \times CFL = 1 + b/a$ (see cover figure) which can be avoided by changing user-defined coefficients a and b during numerical solution.
- For $a = 1$ and $b = 0$ it becomes Newton iteration with faster convergence.
- For $a = 0$ and $b = 1$ it becomes Explicit Picard Iteration with slower convergence. Between these two values, it is a blend of Picard and Newton Iteration thus it is possible to avoid divergence problem usually encountered in Newton iteration by changing coefficients a and b during solution.
- If applied to time dependent differential equations, implicit DPI is theoretically arbitrary order accurate in time and changing the time accuracy doesn't change the formulation. This is while other implicit methods are usually limited to second-order accuracy in time or improving the accuracy requires reconsidering the formulation.

As an unusual result of this paper, in Sec.(3.4.1) the following postulate is proven,

The Discrete Picard Iteration doesn't have dissipation error in time when applied to time-dependent ODEs and PDEs if the residual is computed *exactly* and Jacobian of nonlinear terms remains negative as $t \to \infty$.

Also the numerical solution for a linear scalar ODE (Sec.(3.4.1)), nonlinear ODE. (Sec.(3.4.2)), system of linear ODEs.(Sec.(3.4.3)) and a three-dimensional convection PDE (Sec.(4.3.2)) proves this spectacular property in practice.

In the final chapters, we apply the developed DPI method to multidimensional Partial Differential Equations. In Chapter (4) the Rearrangement Operator is extended to arbitrarily rearrange multidimensional discrete spaces in the desired directions. Using this robust tool, the discrete form of spatial operator (like Divergence and Laplacian) are obtained in discrete space in the easiest form. These matured concepts are then used to discretize a general PDE with arbitrary boundary/initial conditions using DPI.

For the first time a generic *numerical closed-form solution* for arbitrary *linear* conservative PDE $\frac{\partial \mathbf{u}}{\partial t} + \sum_i \sum_j \frac{\partial^i}{\partial x_j^i} \mathbf{F}_{ij} = \mathbf{G}$ is proposed in Sec.(4.3).

A special case, i.e. spectral solution of three-dimensional scalar linear convection with periodic BCs is presented in this chapter.

In Chapter (5) we concentrate on a general class of conservative form of time-evolutionary PDEs know as Conservation Laws. These have numerous applications in science and engineering and include phenomena related to Fluid Dynamics, Electromagnetic Wave Propagation, Financial Modeling, Thermal Science, Elasticity and Plasticity, and etc. Two popular spatial discretization approaches namely Finite-Volume method and Finite Element methods are coupled with DPI scheme and a general formulation and algorithm of solution is brought therein for practitioners.

This editions of the report might have typesetting and/or minor notational errors like indices and duplications. All suggestions regarding any improvement to this documentation are acknowledged.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction and Preliminary Concepts

Let us consider the general time-evolution equation,

$$\frac{\partial u}{\partial t} = R\left(u, t\right), \tag{1.1}$$

Where $R\left(u, t\right)$ is the generalized residuals which means that it depends on the time as well as 'u' and any partial derivative of the 'u' which is discretized over a multidimensional space to gives only time evolution for each spatial position. Here we consider (1.1) in the integral form below.

$$u = u_0 + \int_{t_0}^{t_0+\Delta t} R\left(u\left(\xi\right), \xi\right) d\xi, \tag{1.2}$$

Where $u = [u_0, u]$ and $t = [t_0, t_0 + \Delta t]$. The Picard-Lindelof theorem (Coddington & Norman (1955)) shows the uniqueness of the solution to (1.2). This is also known as Picard's existence theorem or CauchyLipschitz theorem (Coddington & Norman (1955)). However, the iterative approach used in the proof of theorem, known as Picard iteration, is much more interesting from the practical point of view as will be discussed in this work. The Picard iteration can be described as below. Suppose that u_n is an initial guess to the solution (1.2) and assume that $R(u, t)$ is continuous over t and u and Lipschitz continuous over t. Then the following iteration,

$$u_{n+1} = u_0 + \int_{t_0}^{t_0+\Delta t} R\left(u_n\left(\xi\right), \xi\right) d\xi, \tag{1.3}$$

Converges to a unique solution for *analytical* function u. Here we move forward to find an equivalent theorem for the *numerical form* of (1.3).
Let us discretize arbitrary function $g\left(t\right)$ over t_i on a non-uniform stencil with n_S nodes in it. The temporal stencil with the corresponding discretized solution is shown in fig.(1.0.1).

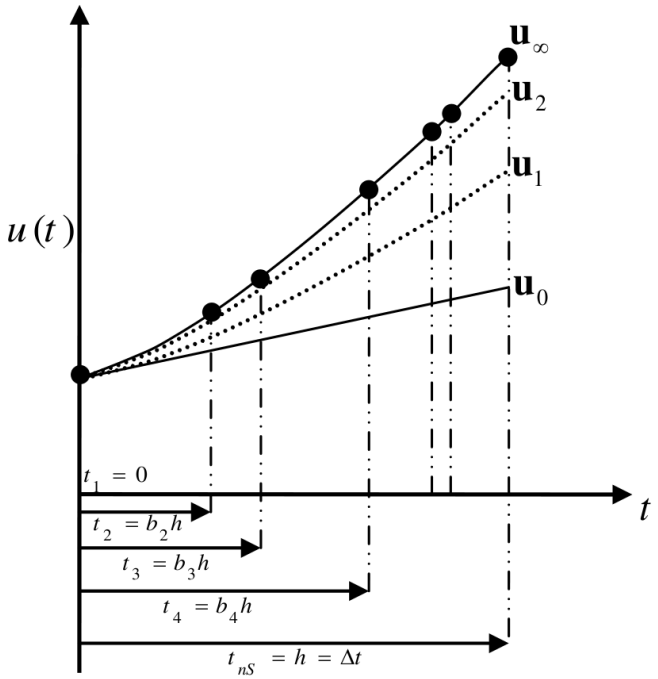


Figure 1.0.1: The schematics of variable spacing grid used for temporal discretization of the integration operator. As shown, the initial vector of nodal values $\mathbf{u}_0$ (the straight line at the bottom) converges to the unique solution $\mathbf{u}_\infty$ as $n \to \infty$.

Evidently the numerical initial function of $g(t_i)$, represented by $\tilde{g}(t_i)$, can be computed by a numerical quadrature scheme over $[t_1, t_{n_S}]$. Let say,

$$\tilde{g}(t_i) = \Delta t \sum_{j=1}^{i} s_j g(t_j) + \mathcal{O}\left(\Delta t^p\right), \tag{1.4}$$

is the quadrature scheme with weight s_j. The order of truncation error p depends on the number of nodes and the type of quadrature(Quarteroni *et al.* (2000)). We note that (1.4) can be written in the matrix form,

$$\tilde{\mathbf{g}} = \Delta t \, \mathbb{S} \, \mathbf{g} + \mathcal{O}\left(\Delta t^p\right), \tag{1.5}$$

Where $\tilde{\mathbf{g}} = [\tilde{g}(t_1) \; \tilde{g}(t_2) \; \ldots \; \tilde{g}(t_{n_S})]^{\mathrm{T}}$ is the vector of *numerical initial function* and

$$\mathbb{S} = \begin{bmatrix} s_0 & & & & \\ s_1 & s_2 & & & \\ s_1 & s_2 & s_3 & & \\ \vdots & \vdots & \vdots & \ddots & \\ s_1 & s_2 & s_3 & \cdots & s_{n_S} \end{bmatrix}, \quad s_0 = 0. \tag{1.6}$$

is the lower triangular matrix of quadrature weights. Note that s_0 must be necessarily zero to ensure $\tilde{g}(t_1) = \int_{t_1}^{t_1} g(\xi)d\xi = 0$. A recent study shows that (1.6) is not necessarily restricted to lower triangular form (Ghasemi (2010)). As an example, consider the special case of (1.4) where the grid spacing is fixed and the scheme is first order $p = 1$.

$$\tilde{g}(t_i) = \Delta t \sum_{j=1}^{i} g(t_j) + \mathcal{O}(\Delta t), \tag{1.7}$$

Obviously this is the Riemann series. When written in the matrix form (1.5)

$$\begin{bmatrix} \tilde{g}(t_1) \\ \tilde{g}(t_2) \\ \vdots \\ \tilde{g}(t_{n_S}) \end{bmatrix} = \Delta t \begin{bmatrix} 0 & & & & \\ 0 & 1 & & & \\ 0 & 1 & 1 & & \\ \vdots & \vdots & \vdots & \ddots & \\ 0 & 1 & 1 & \cdots & 1 \end{bmatrix} + \mathcal{O}(\Delta t) \tag{1.8}$$

Therefore the integration operator for Riemann series can be obtained as follows.

$$\mathbb{S}_{\text{Riemann}} = \begin{bmatrix} 0 & & & & \\ 0 & 1 & & & \\ 0 & 1 & 1 & & \\ \vdots & \vdots & \vdots & \ddots & \\ 0 & 1 & 1 & \cdots & 1 \end{bmatrix} \tag{1.9}$$

Similarly for Newton-Cotes operator we obtain

$$\mathbb{S}_{\text{NC}} = \begin{bmatrix} 0 & & & & & \\ .5 & .5 & & & & \\ .5 & 1 & .5 & & & \\ .5 & 1 & 1 & .5 & & \\ \vdots & \vdots & \vdots & \vdots & \ddots & \\ .5 & 1 & 1 & \cdots & 1 & .5 \end{bmatrix} \tag{1.10}$$

We use (1.5) to discretize the recursive integral equation (1.3) as follows.

$$\begin{aligned} \mathbf{u}_{n+1} &= \mathbf{u}_0 + \Delta t\, \mathbb{S}\, \mathbf{R}_n + \mathcal{O}(\Delta t^p) \\ \mathbf{R}_n &= f(\mathbf{u}_n, t), \end{aligned} \tag{1.11}$$

Equation (1.11) is an iterative matrix equation. For example, using Riemann operator (1.9), (1.11) is fully expanded as below.

$$\underbrace{\begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{nS} \end{bmatrix}_{n+1}}_{\mathbf{u}_{n+1}} = \underbrace{\begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{nS} \end{bmatrix}_{0}}_{\mathbf{u}_0} + \Delta t \underbrace{\begin{bmatrix} 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 1 & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 1 & 1 & 1 & 1 \end{bmatrix}}_{\mathbb{S}} \underbrace{\begin{bmatrix} R_1 \\ R_2 \\ \vdots \\ R_{nS} \end{bmatrix}_{n}}_{\mathbf{R}_n} + \begin{bmatrix} \mathcal{O}\left(\Delta t^p\right) \\ \mathcal{O}\left(\Delta t^p\right) \\ \vdots \\ \mathcal{O}\left(\Delta t^p\right) \end{bmatrix}$$

$$\begin{bmatrix} R_1 \\ R_2 \\ \vdots \\ R_{nS} \end{bmatrix}_{n} = \underbrace{\begin{bmatrix} f\left(u_1, t_1\right) \\ f\left(u_2, t_2\right) \\ \vdots \\ f\left(u_{nS}, t_{nS}\right) \end{bmatrix}_{n}}_{f(\mathbf{u}_n, t)} \tag{1.12}$$

1.1 The Convergence of Discrete Picard Iterations for Ordinary Differential Equations

To study the exsistence of any solution to (1.11), we consider the short interval $[u_1, u_{n_S}] \times [t_1, t_{n_S}]$ such that the residual can be expanded using Taylor series,

$$\mathbf{R}_n = \left(f\left(u_1, t_1\right) + \left.\frac{\partial f}{\partial t}\right|_{(u_1,t_1)} (t - t_1) \right) \mathbf{e} + \left.\frac{\partial f}{\partial u}\right|_{(u_1,t_1)} (\mathbf{u}_n - \mathbf{u}_0) \tag{1.13}$$

Where $e = [1\ 1\ 1\ \ldots\ 1]^{\mathrm{T}}$ is a column vector with length n_S. Equation (1.13) is rearranged to give

$$\mathbf{R}_n = \mathbf{c}_0 + c\mathbf{u}_n \tag{1.14}$$

Where

$$c = \left.\frac{\partial f}{\partial u}\right|_{(u_1,t_1)} \tag{1.15}$$

and

$$\mathbf{c}_0 = \left(f\left(u_1, t_1\right) + \left.\frac{\partial f}{\partial t}\right|_{(u_1,t_1)} (t - t_1) \right) \mathbf{e} - \left.\frac{\partial f}{\partial u}\right|_{(u_1,t_1)} \mathbf{u}_0 \tag{1.16}$$

Substituting (1.14) into (1.11) we have,

$$\mathbf{u}_{n+1} = \mathbf{u}_0 + \Delta t\, \mathbb{S}\ (\mathbf{c}_0 + c\mathbf{u}_n) + \mathcal{O}\left(\Delta t^p\right) \tag{1.17}$$

Or in the simplified form

$$\mathbf{u}_{n+1} = \mathbf{u}_0 + (c\,\Delta t\,\mathbb{S}\,)\,\mathbf{u}_n + \Delta t\,\mathbb{S}\,\mathbf{c}_0 + \mathcal{O}\,(\Delta t^p) \tag{1.18}$$

Which is defined in this paper as the *discrete Picard Iteration*. In the following we study the convergence of (1.18) in detail. A similar but limited theorem can be found in (Ghasemi (2010)).

Theorem I - Discrete Picard Iteration Convergence Theorem for Ordinary Differential Equations
Suppose that $f(u,t)$ in (1.11) is continuous over $u = [u_1, u_{nS}]$ over $t = [t_1, t_1 + \Delta t]$ and Lipschitz continuous over t and the Jacobian $\partial f/\partial u$ exists and is bounded over t. Also assume that the discretization of the integration operator $\mathbb{S}$ is consistent, i.e. $p \geq 1$. Then *for the given operator* $\mathbb{S}$, the discrete Picard Iteration (1.18) converges to a unique solution *if and only if*,

$$\|r_i\| \leq 1 \tag{1.19}$$

for all r_i where $r_i = c\Delta t\lambda_i$ and λ_i is the eigen-value of $\mathbb{S}$.

Proof
We prove this theorem by using mathematical induction and elementary inequality theorems. We start with the first iteration of (1.18). For $n = 0$ we obtain,

$$\begin{aligned} \mathbf{u}_1 &= \mathbf{u}_0 + (c\,\Delta t\,\mathbb{S}\,)\,\mathbf{u}_0 + \Delta t\,\mathbb{S}\,\mathbf{c}_0 + \mathcal{O}\,(\Delta t^p) \\ &= (\mathbb{I} + (c\,\Delta t\,\mathbb{S}\,))\,\mathbf{u}_0 + \Delta t\,\mathbb{S}\,\mathbf{c}_0 + \mathcal{O}\,(\Delta t^p) \end{aligned} \tag{1.20}$$

Where $\mathbb{I}$ is the *identity* matrix. For $n = 1$, we will have,

$$\begin{aligned} \mathbf{u}_2 &= \mathbf{u}_0 + (c\,\Delta t\,\mathbb{S}\,)\,\mathbf{u}_1 + \Delta t\,\mathbb{S}\,\mathbf{c}_0 + \mathcal{O}\,(\Delta t^p) \\ &= \left(\mathbb{I} + (c\,\Delta t\,\mathbb{S}\,) + (c\,\Delta t\,\mathbb{S}\,)^2\right)\mathbf{u}_0 + \left(\Delta t\,\mathbb{S}\, + c(\,\Delta t\,\mathbb{S}\,)^2\right)\mathbf{c}_0 \\ &+ (\mathbb{I} + c\,\Delta t\,\mathbb{S}\,)\,\mathcal{O}\,(\Delta t^p) \end{aligned} \tag{1.21}$$

Following the same procedure, for arbitrary n we obtain,

$$\begin{aligned} \mathbf{u}_n &= \left(\mathbb{I} + \sum_{i=1}^{n} (c\,\Delta t\,\mathbb{S}\,)^i\right)\mathbf{u}_o + \sum_{i=1}^{n} (c\,\Delta t\,\mathbb{S}\,)^i \frac{\mathbf{c}_0}{c} \\ &+ \left(\mathbb{I} + \sum_{i=1}^{n-1} (c\,\Delta t\,\mathbb{S}\,)^i\right)\mathcal{O}\,(\Delta t^p) \end{aligned} \tag{1.22}$$

Using eigen-value decomposition

$$\mathbb{S} = \mathbb{L}^{-1}\mathbf{\Lambda}\mathbb{L}, \tag{1.23}$$

Equation (1.22) is written as the following.

$$\begin{aligned}\mathbf{u}_n &= \left(\mathbb{I} + \sum_{i=1}^{n}\left(c\,\Delta t\,\mathbb{L}^{-1}\mathbf{\Lambda}\mathbb{L}\,\right)^i\right)\mathbf{u}_o + \sum_{i=1}^{n}\left(c\,\Delta t\,\mathbb{L}^{-1}\mathbf{\Lambda}\mathbb{L}\,\right)^i\frac{\mathbf{c}_0}{c} \\ &+ \left(\mathbb{I} + \sum_{i=1}^{n-1}\left(c\,\Delta t\,\mathbb{L}^{-1}\mathbf{\Lambda}\mathbb{L}\,\right)^i\right)\mathcal{O}\left(\Delta t^p\right)\end{aligned} \tag{1.24}$$

Which is further simplified to

$$\begin{aligned}\mathbf{u}_n &= \left(\mathbb{I} + \mathbb{L}^{-1}\sum_{i=1}^{n}\left(c\,\Delta t\,\mathbf{\Lambda}\,\right)^i\mathbb{L}\right)\mathbf{u}_o + \mathbb{L}^{-1}\sum_{i=1}^{n}\left(c\,\Delta t\,\mathbf{\Lambda}\,\right)^i\mathbb{L}\frac{\mathbf{c}_0}{c} \\ &+ \left(\mathbb{I} + \mathbb{L}^{-1}\sum_{i=1}^{n-1}\left(c\,\Delta t\,\mathbf{\Lambda}\,\right)^i\mathbb{L}\right)\mathcal{O}\left(\Delta t^p\right)\end{aligned} \tag{1.25}$$

Choosing the following matrix variable

$$\mathbf{r} = c\,\Delta t\,\mathbf{\Lambda} \tag{1.26}$$

Equation (1.25) is rewritten as follows,

$$\begin{aligned}\mathbf{u}_n &= \left(\mathbb{I} + \mathbb{L}^{-1}\left(\sum_{i=1}^{n}\mathbf{r}^i\right)\mathbb{L}\right)\mathbf{u}_o + \mathbb{L}^{-1}\left(\sum_{i=1}^{n}\mathbf{r}^i\right)\mathbb{L}\frac{\mathbf{c}_0}{c} \\ &+ \left(\mathbb{I} + \mathbb{L}^{-1}\left(\sum_{i=1}^{n-1}\mathbf{r}^i\right)\mathbb{L}\right)\mathcal{O}\left(\Delta t^p\right)\end{aligned} \tag{1.27}$$

We notice from elementary algebra that the geometric series $\sum_{i=1}^{n}\mathbf{r}^i$ has the partial sum

$$\sum_{i=1}^{n}\mathbf{r}^i = \mathbf{r}\left(\mathbf{r}^n - \mathbb{I}\right)\left(\mathbf{r} - \mathbb{I}\right)^{-1} \tag{1.28}$$

Substituting (1.28) into (1.27) we obtain a *closed form solution* for $\mathbf{u}_n$ as below.

$$\begin{aligned}\mathbf{u}_n &= \left(\mathbb{I} + \mathbb{L}^{-1}\mathbf{r}\left(\mathbf{r}^n - \mathbb{I}\right)\left(\mathbf{r} - \mathbb{I}\right)^{-1}\mathbb{L}\right)\mathbf{u}_o + \mathbb{L}^{-1}\mathbf{r}\left(\mathbf{r}^n - \mathbb{I}\right)\left(\mathbf{r} - \mathbb{I}\right)^{-1}\mathbb{L}\frac{\mathbf{c}_0}{c} \\ &+ \left(\mathbb{I} + \mathbb{L}^{-1}\mathbf{r}\left(\mathbf{r}^{n-1} - \mathbb{I}\right)\left(\mathbf{r} - \mathbb{I}\right)^{-1}\mathbb{L}\right)\mathcal{O}\left(\Delta t^p\right)\end{aligned} \tag{1.29}$$

Since $p \geq 1$ then the truncation error vanishes as time interval Δt becomes smaller. In particular, from (1.29) we have.

$$\begin{aligned} \lim_{\Delta t \to \epsilon} \mathbf{u}_n &= \left(\mathbb{I} + \mathbb{L}^{-1} \mathbf{r} \left(\mathbf{r}^n - \mathbb{I} \right) \left(\mathbf{r} - \mathbb{I} \right)^{-1} \mathbb{L} \right) \mathbf{u}_o \\ &+ \mathbb{L}^{-1} \mathbf{r} \left(\mathbf{r}^n - \mathbb{I} \right) \left(\mathbf{r} - \mathbb{I} \right)^{-1} \mathbb{L} \frac{\mathbf{c}_0}{c} + K_0 \left(\epsilon \right), \end{aligned} \tag{1.30}$$

Where the upper bound for truncation error $K_0 \left(\epsilon \right) = \left(\mathbb{I} + \mathbb{L}^{-1} \mathbf{r} \left(\mathbf{r}^n - \mathbb{I} \right) \left(\mathbf{r} - \mathbb{I} \right)^{-1} \mathbb{L} \right) \mathcal{O} \left(\epsilon^p \right)$ vanishes for $p \geq 1$ as $\epsilon \to 0$. Therefore, from (1.30) it is evident that $\mathbf{u}_n$ is bounded *if and only if* $\| r_i \| \leq 1$ and $\mathbf{c}_0$ is bounded. That means we *must* have

$$\| r_i = c \lambda_i \Delta t \| \leq 1 \tag{1.31}$$

To show $\mathbf{c}_0$ is bounded we note from (1.16) that

$$| \mathbf{c}_0 | \leq | f \left(u_1, t_1 \right) | + \left| \left. \frac{\partial f}{\partial t} \right|_{(u_1, t_1)} \right| \Delta t + \left| \left. \frac{\partial f}{\partial u} \right|_{(u_1, t_1)} u_0 \right| \tag{1.32}$$

Now we need to find a bound for time derivative term in the RHS of (1.32). Since $f(u, t)$ is Lipschitz continuous over time, then by definition, there exist a K such that,

$$D_f \left(f \left(u, t_1 \right), f \left(u, t_2 \right) \right) \leq K D_t \left(t_1, t_2 \right) \tag{1.33}$$

For all t_1 and t_2 in the interval t. For $| t_2 - t_1 | \leq \epsilon$ the distance operator D in Lipschitz condition (1.33) converges to the differential operator. Therefore we obtain the following,

$$\begin{aligned} \lim_{t_2 \to t_1} \frac{D_f \left(f \left(u, t_1 \right), f \left(u, t_2 \right) \right)}{D_t \left(t_1, t_2 \right)} &\leq K \\ \to \left| \frac{\partial f}{\partial t} \left(u, t \right) \right| &\leq K \end{aligned} \tag{1.34}$$

That means the time derivative of f is bounded for all $t \in [t_1, t_{nS}]$ and $u \in [u_1, u_{nS}]$. Therefore (1.34) is valid for point (t_1, u_1), i.e., $\left| \frac{\partial f}{\partial t} \left(u_1, t_1 \right) \right| \leq K$. Since f is continuous over $[t_1, t_{nS}] \times [u_1, u_{nS}]$, then it is *bounded*. Therefore we obtain another bound for f itself as follows.

$$| f \left(u_1, t_1 \right) | \leq K' \tag{1.35}$$

Also, since Jacobian $\partial f/\partial u$ exists over t, we will have

$$\left| \left. \frac{\partial f}{\partial u} \right|_{(u_1,t_1)} u_0 \right| \leq K'' \tag{1.36}$$

Substituting (1.35) and (1.34) and (1.36) into (1.32) we obtain

$$|\mathbf{c}_0| \leq K' + \Delta t\ K + K'' \tag{1.37}$$

So far we showed that $\mathbf{u}_n$ in (1.29) remains bounded over infinitesimal Δt for arbitrarily large n if and only if (1.31) is satisfied. We need to show that $\mathbf{u}_n$ converges to a *unique* function as $n \to \infty$. Using (1.28), for sufficiently large n, (1.29) can be written as the following.

$$\begin{aligned} \lim_{n\to\infty} \mathbf{u}_n &= \left(\mathbb{I} + \mathbb{L}^{-1}\mathbf{r}(\mathbb{I}-\mathbf{r})^{-1}\mathbb{L}\right)\mathbf{u}_o + \mathbb{L}^{-1}\mathbf{r}(\mathbb{I}-\mathbf{r})^{-1}\mathbb{L}\frac{\mathbf{c}_0}{c} \\ &+ \left(\mathbb{I} + \mathbb{L}^{-1}\mathbf{r}(\mathbb{I}-\mathbf{r})^{-1}\mathbb{L}\right)\mathcal{O}\left(\Delta t^p\right) \end{aligned} \tag{1.38}$$

We assume that $\mathbf{u}_n$ converges to $\bar{\mathbf{u}}_n$ which is not unique. We show that this is a false assumption. Since $\bar{\mathbf{u}}_n$ is a solution to (1.18) then it can be cast down to (1.38) as follows.

$$\begin{aligned} \bar{\mathbf{u}}_n &= \left(\mathbb{I} + \mathbb{L}^{-1}\mathbf{r}(\mathbb{I}-\mathbf{r})^{-1}\mathbb{L}\right)\mathbf{u}_o + \mathbb{L}^{-1}\mathbf{r}(\mathbb{I}-\mathbf{r})^{-1}\mathbb{L}\frac{\mathbf{c}_0}{c} \\ &+ \left(\mathbb{I} + \mathbb{L}^{-1}\mathbf{r}(\mathbb{I}-\mathbf{r})^{-1}\mathbb{L}\right)\bar{\mathcal{O}}\left(\Delta t^p\right) \end{aligned} \tag{1.39}$$

Note that $\mathbf{u}_0$ and $\mathbf{c}_0$ only depend on the initial values (according to 1.16) so they are same for any solution to (1.18) as shown in (1.39). However, we note that $\mathbf{r} = c\Delta t\mathbf{\Lambda}$ depends on integrator $\mathbb{S}$. Therefore we conclude that for different operators the solution is different. From the assumptions of the theorem, $\mathbb{S}$ is unique and hence $\mathbf{r}$ doesn't change between $\mathbf{u}_n$ and $\bar{\mathbf{u}}_n$ in (1.39) and (1.38). The truncation error might be different due to different time interval used. We represented the new truncation error with $\bar{\mathcal{O}}\left(\Delta t^p\right)$ in (1.39). Subtracting (1.38) from (1.39), we obtain,

$$\lim_{\Delta t\to 0} \bar{\mathbf{u}}_n - \mathbf{u}_n = \left(\mathbb{I} + \mathbb{L}^{-1}\mathbf{r}(\mathbb{I}-\mathbf{r})^{-1}\mathbb{L}\right)\left(\bar{\mathcal{O}}\left(\Delta t^p\right) - \mathcal{O}\left(\Delta t^p\right)\right) = 0. \tag{1.40}$$

That means $\mathbf{u}_n$ is unique for $\Delta t \to 0$ and the proof is complete. ▲

Corollary I-I For Riemann operator (1.9), we have $\lambda_{\max} = 1$. Therefore, the maximum allowable time step is obtained using Theorem I,

$$\Delta t \leq \left(\frac{1}{\lambda_{\max} c} = \frac{1}{c} \right) \tag{1.41}$$

Corollary I-II For Newton-Cotes operator (1.10), we have $\lambda_{\max} = 1/2$. Therefore,

$$\Delta t \leq \frac{2}{c} \tag{1.42}$$

Comparing (1.41) and (1.42), we observe that the ratio of maximum allowable time-steps of Newton-Cotes operator to the maximum time step of Riemann operator is 2. Therefore the former is twice more efficient that the latter.

Conclusions from corollaries (I-I) and (I-II)
We noticed that decrease in the maximum eigen-value of the operator resulted in increase in allowable time step. Therefore we might think about designing integration operators that have small eigen-values. This is the motivation for Sec. (1.2).

1.1.1 Convergence Mechanism

In this section we drive an interesting lemma which describes the convergence of discrete Picard iterations in the linear case.

Lemma I- the discrete Picard iteration is exponentially convergent with exponent r.

Proof:
We need to find an expression for *total error*. Subtracting the converged solution (1.38) from (1.29) we will have,

$$\begin{aligned}
\delta_n &= \mathbf{u}_n - \mathbf{u}_{n\to\infty} = \mathbb{L}^{-1}\left(\mathbf{r}\left(\mathbf{r}^n - \mathbb{I}\right)\left(\mathbf{r} - \mathbb{I}\right)^{-1} - \mathbf{r}(\mathbb{I} - \mathbf{r})^{-1}\right)\mathbb{L}\mathbf{u}_o \\
&+ \mathbb{L}^{-1}\left(\mathbf{r}\left(\mathbf{r}^n - \mathbb{I}\right)\left(\mathbf{r} - \mathbb{I}\right)^{-1} - \mathbf{r}(\mathbb{I} - \mathbf{r})^{-1}\right)\mathbb{L}\frac{\mathbf{c}_0}{c} \\
&+ \mathbb{L}^{-1}\underbrace{\left(\mathbf{r}\left(\mathbf{r}^{n-1} - \mathbb{I}\right)\left(\mathbf{r} - \mathbb{I}\right)^{-1} - \mathbf{r}(\mathbb{I} - \mathbf{r})^{-1}\right)}_{\mathbf{r}^n(\mathbf{r}-\mathbb{I})^{-1}}\mathbb{L}\mathcal{O}\left(\Delta t^p\right)
\end{aligned} \tag{1.43}$$

Where δ_n is the *total error* of iteration n. Defining new variables $\mathbf{E}_n = \mathbb{L}\delta_n$, $\mathbf{U}_0 = \mathbb{L}\mathbf{u}_0$, $\mathbf{C}_0 = \mathbb{L}\mathbf{c}_0$, and $\mathbf{O} = \mathbb{L}\mathcal{O}$, eq.(1.43) can be written as,

$$\mathbf{E}_n = \mathbf{r}^{n+1}(\mathbf{r} - \mathbb{I})^{-1}\left(\mathbf{U}_0 + \frac{\mathbf{C}_0}{c}\right) + \mathbf{r}^n(\mathbf{r} - \mathbb{I})^{-1}\mathbf{O}\left(\Delta t^p\right), \tag{1.44}$$

For the first iteration we have $\mathbf{E}_0 = \mathbf{r}(\mathbf{r} - \mathbb{I})^{-1}\left(\mathbf{U}_0 + \frac{\mathbf{C}_0}{c}\right) + (\mathbf{r} - \mathbb{I})^{-1}\mathbf{O}\left(\Delta t^p\right)$. Therefore from (1.44) we have,

$$\mathbf{E}_n = \mathbf{r}^n \mathbf{E}_0 \tag{1.45}$$

Since $\|r_i\| \leq 1$ then (1.45) is exponentially convergent and proof is complete. ▲

Assume that the error ratio $\mathbf{E}_n/\mathbf{E}_0$ reaches to machine zero. Then form (1.45), we will have

$$\frac{\mathbf{E}_n}{\mathbf{E}_0} = \mathbf{r}^n = \frac{b^{-P}}{2} \tag{1.46}$$

Where b is called based or radix and P is the number of fractional digits(Goldberg (1991)). These values are shown in Table (1.1.1) for various decimal number representation.

IEEE 754-2008	**Common name**	**C++ data type**	**b**	**P**	**Machine epsilon**
binary16	half precision	NA	2	10	4.88e-04
binary32	single precision	float	2	23	5.96e-08
binary64	double precision	double	2	52	1.11e-16
binary128	quad(ruple) precision	long double	2	112	9.63e-35

Table 1.1: The values of P and b according to IEEE 754-2008.

We can easily obtain the number of Picard iterations required for convergence on a *binary* machine ($b = 2$) using eq.(1.46).

$$\mathbf{N} = \left\lceil \frac{(P+1)}{\log_2^{\mathbf{R}}} \right\rceil \tag{1.47}$$

Where $\mathbf{R} = \mathbf{r}^{-1} = \mathbf{diag}\left(1/\left(c\lambda_i \Delta t\right)\right)$ contains N_i which is the number of iterations required for the solution at the ith node, i.e., $u\left(x_i\right)$ to converge to machine zero. According to (1.47), since $\mathbf{N}$ is a descending function of $\mathbf{R}$, it maximizes when $\mathbf{R}$ minimizes. On the other hand, $\mathbf{R} = \mathbf{r}^{-1}$ minimizes when $\mathbf{r}$ maximizes. Therefore the number of Picard iterations required so that all $u\left(x_i\right)$ converges to machine zero is

$$N = \left\lceil \frac{(P+1)}{-\log_2^{\max(\mathbf{r})}} \right\rceil = \left\lceil \frac{(P+1)}{-\log_2^{(c\lambda \max \Delta t)}} \right\rceil \tag{1.48}$$

For double precision accuracy $P = 53$ (1). Therefore,

$$\mathbf{N} = \left\lceil \frac{53}{-\log_2^{\max(\mathbf{r})}} \right\rceil \tag{1.49}$$

The variation of number of iterations versus $R_i = 1/r_i$ (eq.(1.47)) is shown in fig.(1.1.1) for different data types. As we see, the number of iterations increases as data type becomes more elaborated and/or R_i decreases. This observation proves the important fact that if R_i decreases or equivalently r_i increases. then according to Theorem I-eq.(1.31) for fixed c and λ_i, the time step increaes and hence for higher time steps, the number of Picard iterations increases. In the other words, the efficiency of the discrete Picard iterations decreases for higher time steps.

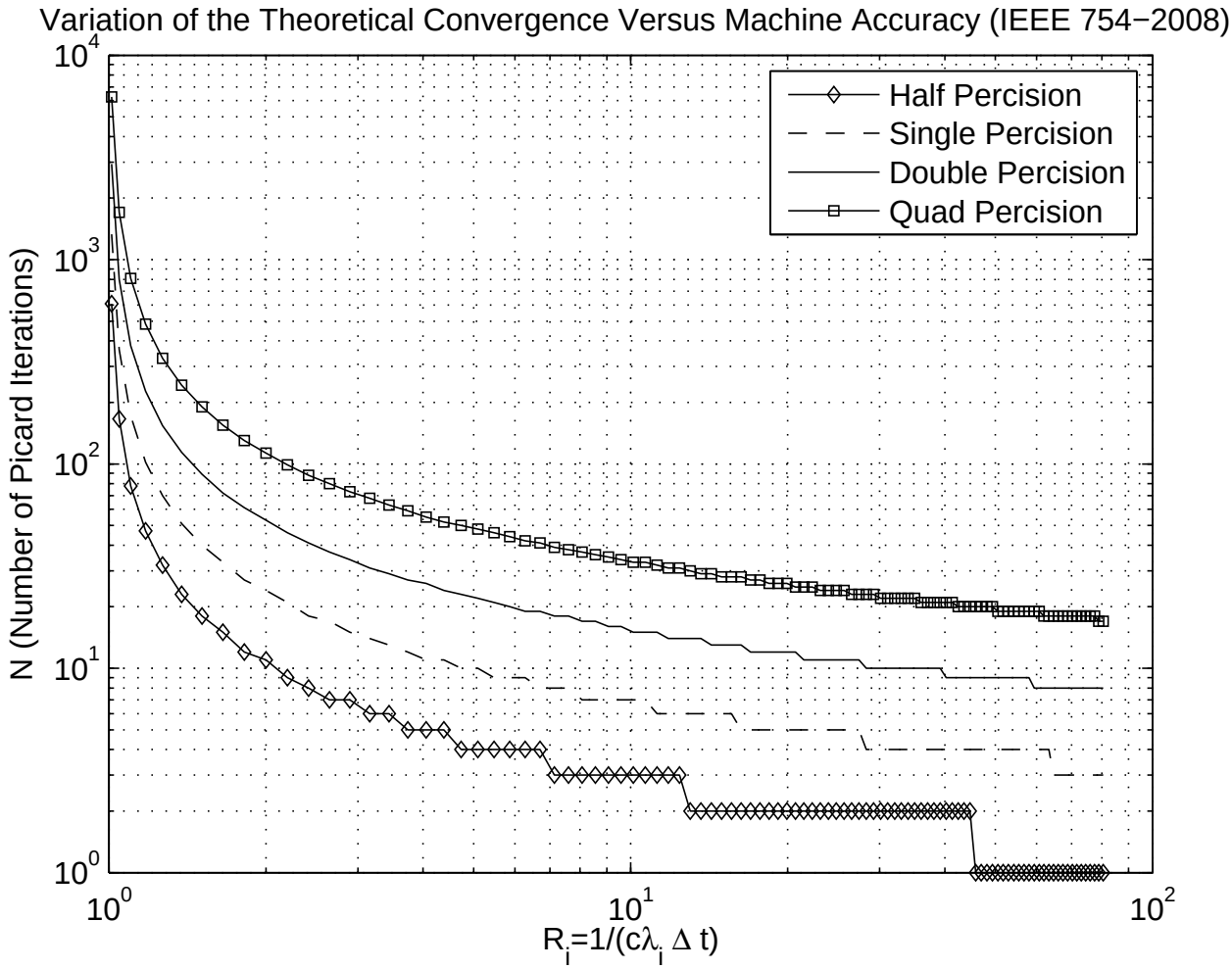


Figure 1.1.1: The number of Picard iterations required for convergence to machine zero.

1.2 Integration Operator Design

In the collary I-II of Theorem I, we observed that when the maximum eigen values of the integrator operator decrease, the maximum allowable time step increases significantly. This interesting property makes numerical solution to be highly efficient by using larger time steps. Here is the question: "Can we design an integrator operator

which has a very small eigen-value while is high-order accurate?".
To answer this question, let us consider fig.(1.0.1) for the second time. To find a quadrature scheme for the composite grid spacing shown in fig.(1.0.1) we use the following high-order interpolant.

$$g(t) = \begin{bmatrix} 1 & t & t^2 & \cdots & t^{N-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{N-1} \end{bmatrix} \tag{1.50}$$

Where the coefficients a_i are to be determined. If we evaluate (1.50) for all times t_i we obtain,

$$\begin{bmatrix} g_1 \\ g_1 \\ \vdots \\ g_N \end{bmatrix} = \begin{bmatrix} 1 & t_1 & t_1^2 & \cdots & t_1^{N-1} \\ 1 & t_2 & t_2^2 & \cdots & t_2^{N-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & t_N & t_N^2 & \cdots & t_N^{N-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{N-1} \end{bmatrix} \tag{1.51}$$

In general, we can define the stretching function $0 \leq b \leq 1$ such that the time t_i can be written in the term of interval width $\Delta t = h$ as the following.

$$t = bh \tag{1.52}$$

We note that (1.52) is evaluated in discrete points t_i shown in fig.(1.0.1) as following

$$t_i = b_i h, \quad b_1 = 0, \quad (i = 1 \ldots N) \tag{1.53}$$

Where b_1 is always zero. Obviously, for $b_i = (i-1)\,/\,(N-1)$ we obtain the *uniform* spacing. In this work we use the following polynomial stretching function to minimize the eigen values of the integration operator.

$$b_i = \left(\frac{i-1}{N-1} \right)^{\alpha} \tag{1.54}$$

Where α is the stretching strength. Using (1.53), eq.(1.51) is written as,

$$\underbrace{\begin{bmatrix} g_1 \\ g_1 \\ \vdots \\ g_N \end{bmatrix}}_{\mathbf{g}} = \underbrace{\begin{bmatrix} 1 & b_1 & b_1^2 & \cdots & b_1^{N-1} \\ 1 & b_2 & b_2^2 & \cdots & b_2^{N-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & b_N & b_N^2 & \cdots & b_N^{N-1} \end{bmatrix}}_{\mathbb{B}} \underbrace{\begin{bmatrix} 1 & & & & \\ & h & & & \\ & & h^2 & & \\ & & & \ddots & \\ & & & & h^{N-1} \end{bmatrix}}_{\mathbf{H}} \underbrace{\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{N-1} \end{bmatrix}}_{\mathbf{a}} \tag{1.55}$$

The above equation can be solved to find the coefficients of the interpolation function.

$$\mathbf{a} = \mathbf{H}^{-1}\mathbb{B}^{-1}\mathbf{g} \tag{1.56}$$

The analytical integration of (1.50) leads to

$$\tilde{g}(t) = \int g(t) = \begin{bmatrix} t & \frac{1}{2}t^2 & \frac{1}{3}t^3 & \cdots & \frac{1}{N}t^N \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{N-1} \end{bmatrix} \tag{1.57}$$

Evaluating (1.57) at unequally spaced points (1.53) gives us,

$$\tilde{g}(t_i) = \begin{bmatrix} b_i h & \frac{1}{2}b_i^2 h^2 & \frac{1}{3}b_i^3 h^3 & \cdots & \frac{1}{N}b_i^N h^N \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{N-1} \end{bmatrix} \tag{1.58}$$

Or in the matrix form below.

$$\begin{aligned} \tilde{g}(t_i) &= \underbrace{\begin{bmatrix} b_i & \frac{1}{2}b_i^2 & \frac{1}{3}b_i^3 & \cdots & \frac{1}{N}b_i^N \end{bmatrix}}_{\mathbf{v}(i)} \underbrace{\begin{bmatrix} h & & & & \\ & h^2 & & & \\ & & h^3 & & \\ & & & \ddots & \\ & & & & h^N \end{bmatrix}}_{h\mathbf{H}} \underbrace{\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{N-1} \end{bmatrix}}_{\mathbf{a}} \\ &= h\, \mathbf{v}(i)\, \mathbf{H}\, \mathbf{a} \end{aligned} \tag{1.59}$$

We note that $\mathbf{v}(i)$ is a row vector whose length is equal to the number of nodes used to evaluate the quadrature. If we evaluate (1.59) for all nodes $i = 1..n_S$, we obtain the matrix form of quadrature scheme for complete integration over Δt. In other words

$$\underbrace{\begin{bmatrix} \tilde{g}(t_1) \\ \tilde{g}(t_2) \\ \tilde{g}(t_3) \\ \vdots \\ \tilde{g}(t_{n_S}) \end{bmatrix}}_{\tilde{\mathbf{g}}} = \underbrace{\begin{bmatrix} 0 & & & & \\ b_2 & \frac{1}{2}b_2^2 & & & \\ b_3 & \frac{1}{2}b_3^2 & \frac{1}{3}b_3^3 & & \\ \vdots & \vdots & \vdots & \ddots & \\ b_{n_S} & \frac{1}{2}b_{n_S}^2 & \frac{1}{3}b_{n_S}^3 & \cdots & \frac{1}{n_S}b_{n_S}^{n_S} \end{bmatrix}}_{\mathbb{V}} \underbrace{\begin{bmatrix} h & & & & \\ & h^2 & & & \\ & & h^3 & & \\ & & & \ddots & \\ & & & & h^{n_S} \end{bmatrix}}_{h\mathbf{H}} \underbrace{\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{(n_S-1)} \end{bmatrix}}_{\mathbf{a}}$$

$$= h\,\mathbb{V}\,\mathbf{H}\,\mathbf{a} \tag{1.60}$$

Substituting the vector '$\mathbf{a}$' from (1.56) into (1.60), we will obtain the final form of high-order variable spacing quadrature scheme

$$\tilde{\mathbf{g}} = h\,\mathbb{V}\,\mathbf{H}\,\mathbf{a} = h\,\mathbb{V}\,\mathbf{H}\,\left(\mathbf{H}^{-1}\mathbb{B}^{-1}\mathbf{g}\right) = h\,\mathbb{V}\,\mathbb{B}^{-1}\,\mathbf{g} \tag{1.61}$$

Comparing (1.61) with (1.5) we find that $\mathbb{V}\,\mathbb{B}^{-1}$ is the integration operator.

$$\mathbb{S} = \mathbb{V}\mathbb{B}^{-1} \tag{1.62}$$

A comparison between analytical integration and the one obtained by integration operator is presented in fig.(1.2.1). While the numerical integration obtained by applying integration operator (1.62) is in excellent agreement with analytical solution for lower values of stretching strength α, it differs significantly as α increases.

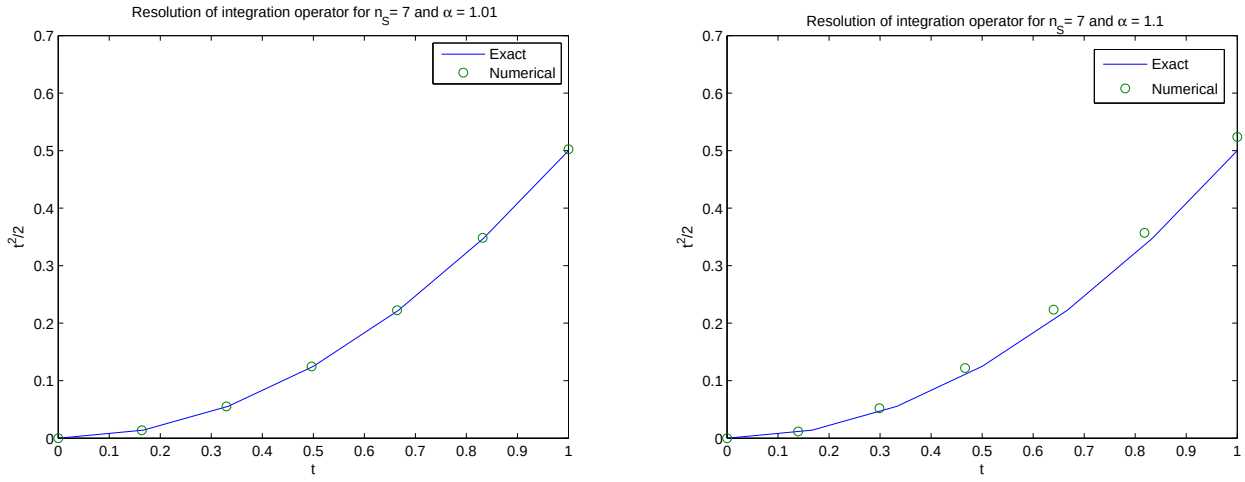


Figure 1.2.1: The result of $\int_0^t \xi dxi = t^2/2$ obtained by using integration operator (1.62) for $n_S = 7$ and Left) $\alpha = 1.01$. Right) $\alpha = 1.1$.

According to theorem (I), the maximum allowable time step is equal to the inverse of $c\lambda_{max}$.

$$\Delta t_{\text{allowable}} = \frac{1}{c\lambda_{\text{max}}}, \tag{1.63}$$

Therefore the maximum eigen-value of the operator plays vital rule in stability of the Picard method. To analyze the maximum eigen-value of $\mathbb{S}$, we changed the values of stretching strength α and the number of nodes in the integration operator to numerically calculate the maximum eigen-value. The result is plotted in fig.(1.2.2) where the ratio of maximum values of $\lambda_{\text{Riemann}}/\lambda_{\text{new}} = 1/\lambda_{\text{new}}$ is plotted versus stretching strength for different sizes of integration operator. We will prove in theorem (II) that this ratio is equal to $CFL_{\text{Picard}}/CFL_{\text{Euler}}$ or simply CFL/CFL_0 for general partial differential equations. Interestingly, this ratio goes to near 60 for stretching $\alpha \simeq 1.2$ without solving implicit system of equations. In the case of ODEs, this ratio resembles the size of time step that we can take so that the numerical integration remains stable compared to explicit Euler stepping.

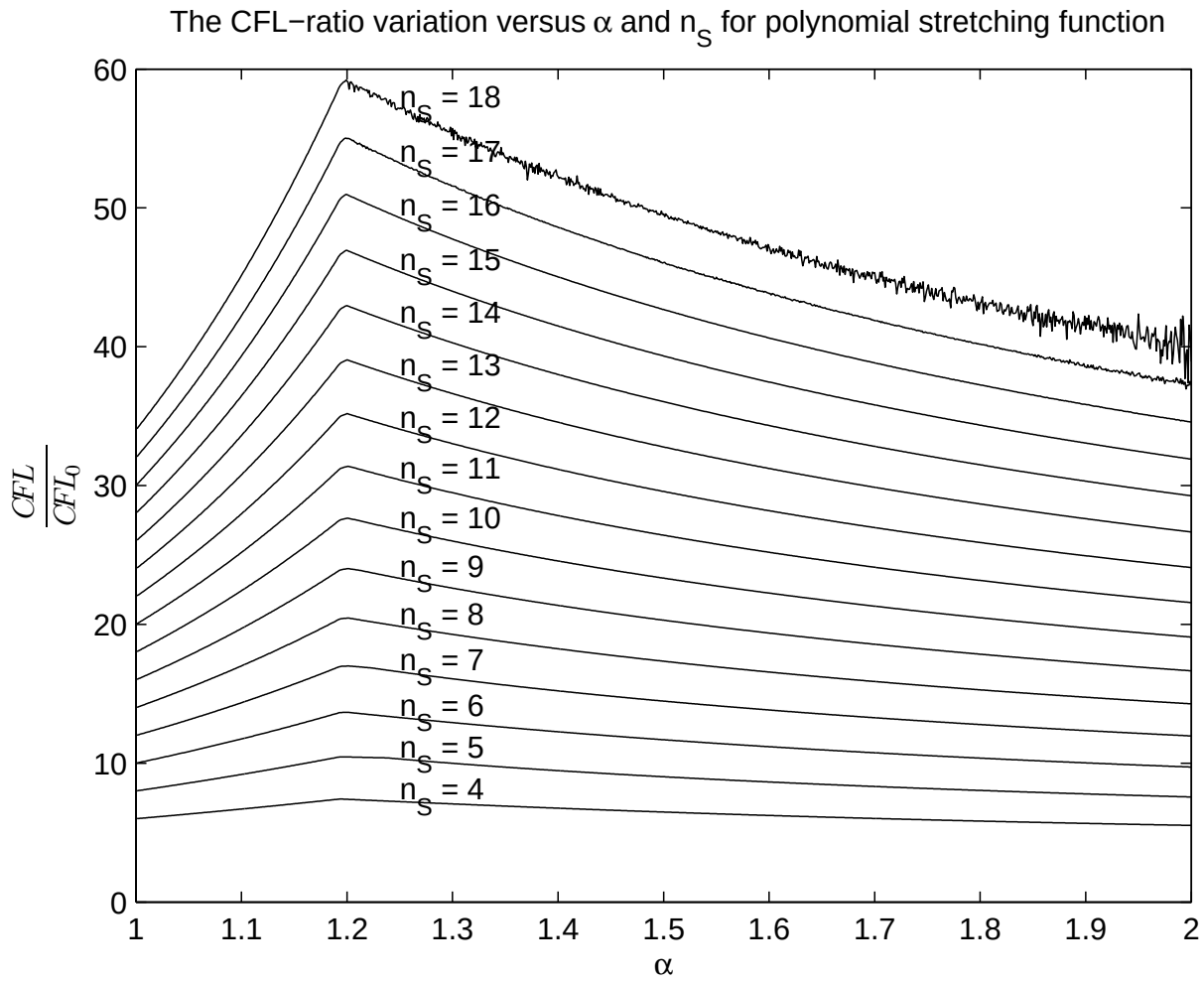


Figure 1.2.2: The ratio of $\frac{1}{\lambda_{max}} = \frac{CFL}{CFL_0}$ for integration operator $\mathbb{S}$ for various sizes of the operator and stretching strength α.

1.3 Numerical Simulations for ODEs

In this section we implement the Discrete Picard Iteration method in the time-marching format to find the numerical solution of the simplest linear ordinary differential equation,

$$\begin{aligned} \frac{du}{dt} &= -cu, \\ u(0) &= 1 \end{aligned} \tag{1.64}$$

with analytical solution

$$u(t) = \exp(-c\,t) \tag{1.65}$$

The explicit Euler time-stepping solution of (1.64) is stable for (Quarteroni *et al.* (2000))

$$\Delta t_{\max} = \frac{2}{c}, \tag{1.66}$$

Therefore we expect that Euler stepping should be stable unless $c\ \Delta t_{\max} = CFL_{\text{ODE}}$ is greater than two.

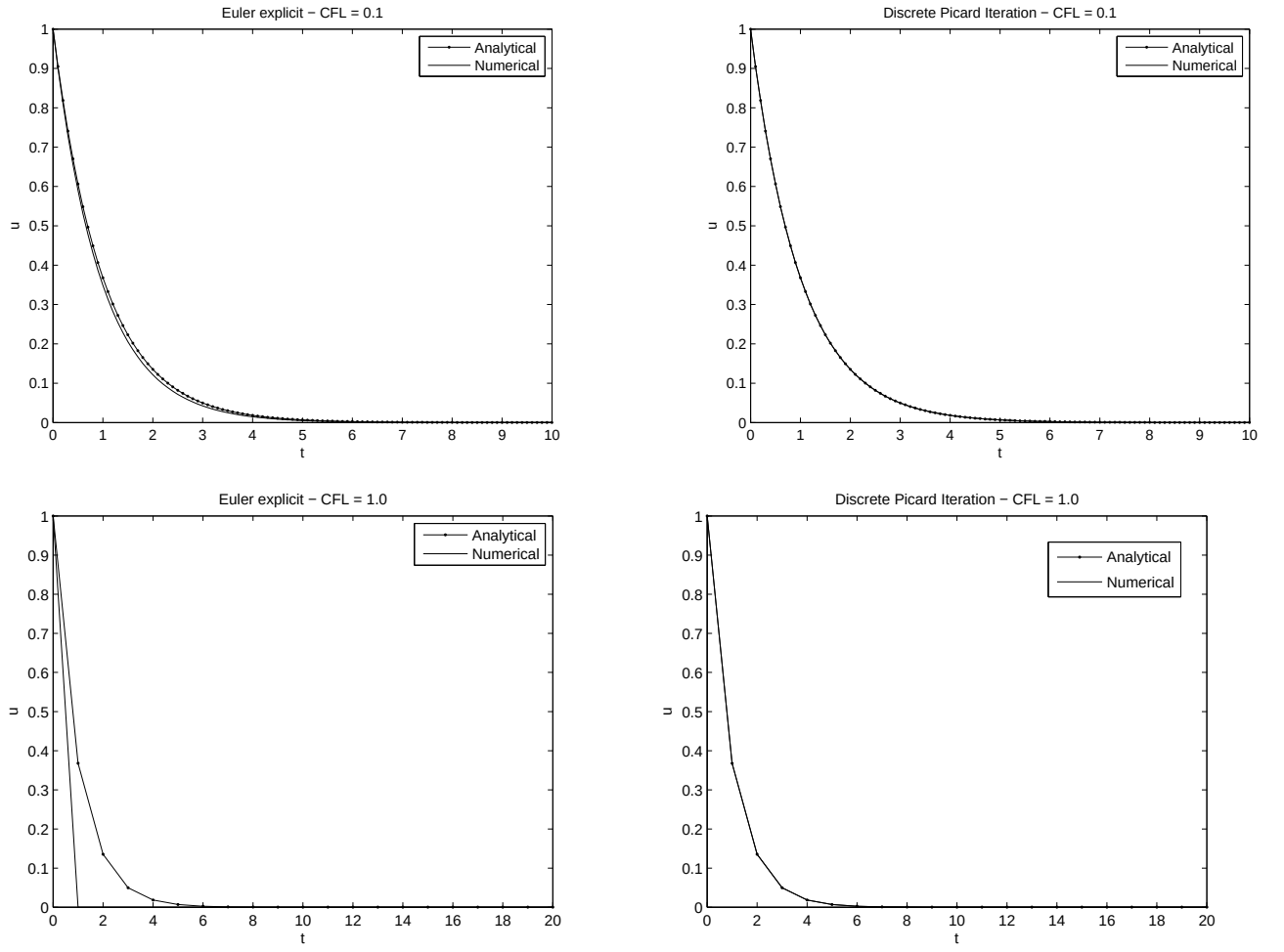


Figure 1.3.1: The comparison between Discrete Picard Iteration method and Euler stepping for solving linear ODE (1.64).

This behavior, is observed in numerical experiment shown in fig.(1.3.1-Left column) where for boundary value $CFL = 2.$, it exhibits oscillatory solution and for $CFL > 2.$ the solution diverges. In addition, note that the Euler explicit method is highly dissipative, even for small CFL numbers. Now we solve the same problem using a time-marching variant of discrete Picard iteration (1.11). The ODE marching algorithm is described as below.

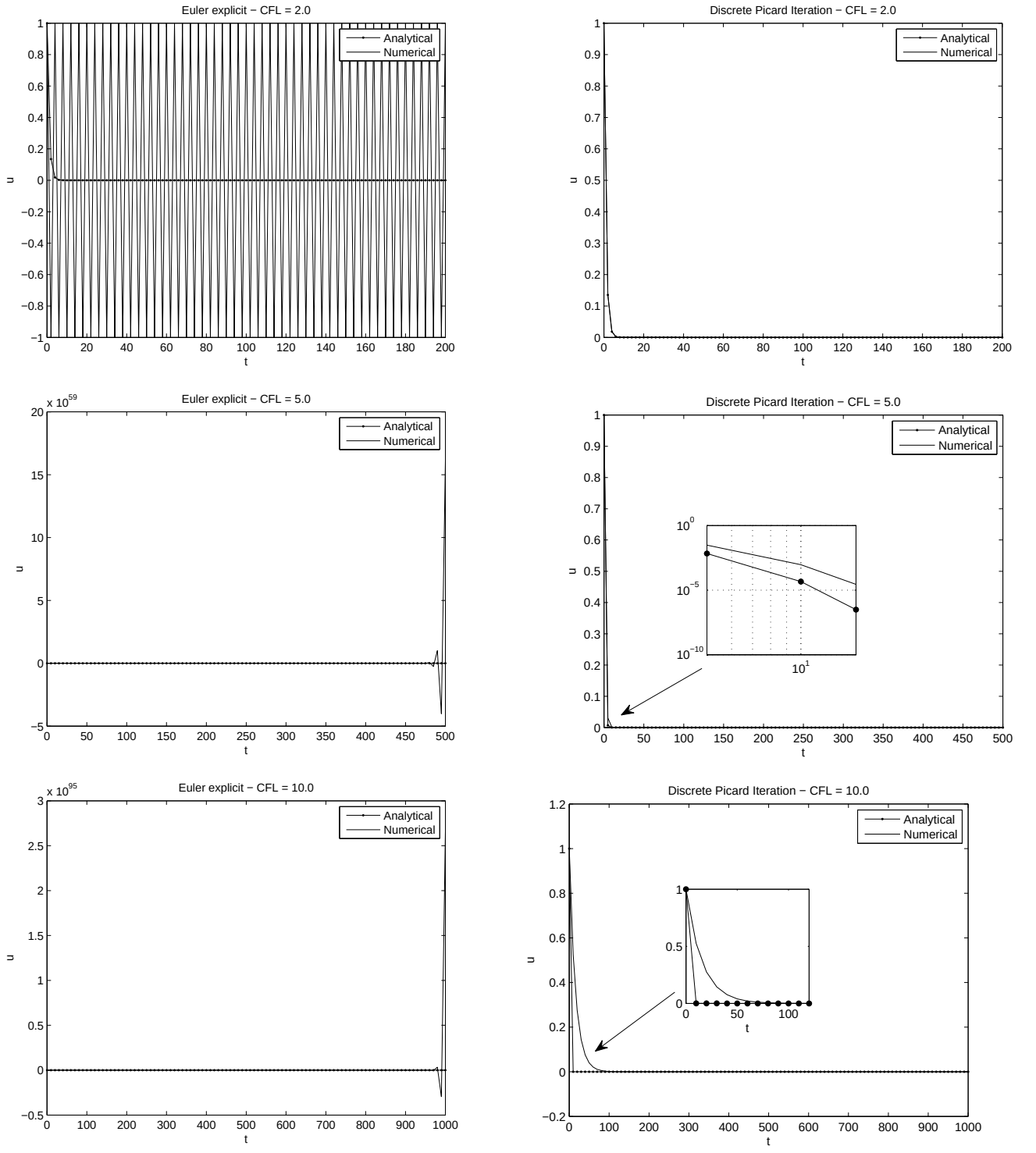


Figure 1.3.2: Continued- The comparison between Discrete Picard Iteration method and Euler stepping for solving linear ODE (1.64).

1. **begin program**

2. given stretching strength α and operator size n_s, calculate $\mathbb{S}$.

3. set u_i to initial condition $u_i = u(t_0)$.

   (a) **main loop**

   (b) initialize u_0 with initial value u_i.

   (c) Select the maximum allowable Δt according to (1.63).

      i. **while** $|u_{n+1} - u_n| \geq$ threshold

      ii. solve (1.11)

      iii. **end while**

   (d) store the last solution of $u(t)$ at u_{n_S}.

   (e) replace the initial value u_i with , i.e., $u_i = u_{n_S}$.

   (f) **end main loop**

4. **end program**

The results of executing the above algorithm are covered in (1.3.1-right). As shown, while DPI is non-dissipative for $CFL \leq 2$, it remains accurate and stable for $CFL \geq 2$ where the dissipation error dominates in $CFL = 10$.

Chapter 2

The Discrete Picard Iteration Method for Partial Differential Equations

2.1 The Rearrangement Operator

Before extending the discrete Picard iteration to multidimensional partial differential equations, we need to define a new operator to rearrange data in the spatial and temporal orders whenever needed. Suppose that the *spatial* space is a z-dimensional continuous space which is discretized using N_{nodes} number of nodes. Each nodes contains the solution at time t_i. Therefore the following space-time vector contains solution for *all* nodes at discrete times t_1 through t_{nS}.

$$[\mathbf{u}] = \begin{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{Nnodes} \end{bmatrix}_{t_1} \\ \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{Nnodes} \end{bmatrix}_{t_2} \\ \vdots \\ \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{Nnodes} \end{bmatrix}_{t_{n_S}} \end{bmatrix} \tag{2.1}$$

We used the notation $[\mathbf{u}]$ since vector $\mathbf{u}$ is nested to form a tensor. Now suppose that we want to find the spatial derivative $\mathbf{u}$ at all time steps. For simplicity, assume

that a one dimensional space $\mathbb{R}_1$ with nodes 1 to N_{nodes} arranged in equal spacing is given. Using a first order upwind difference operator we will have,

$$\mathbf{D}_x u = \frac{1}{\Delta x} \underbrace{\begin{bmatrix} 0 & & & & \\ -1 & 1 & & & \\ & -1 & 1 & & \\ & & \ddots & \ddots & \\ & & & -1 & 1 \end{bmatrix}}_{\mathbf{D}} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{Nnodes} \end{bmatrix} \tag{2.2}$$

to find the spatial derivative of $[\mathbf{u}]$ in all time steps, we simply put block $\mathbf{D}_x$ on the main diagonal of a new tensor, say

$$[\mathbb{D}]_x [\mathbf{u}] = \tag{2.3}$$

$$\frac{1}{\Delta x} \begin{bmatrix} \begin{bmatrix} 0 & & & & \\ -1 & 1 & & & \\ & -1 & 1 & & \\ & & \ddots & \ddots & \\ & & & -1 & 1 \end{bmatrix} & & & \\ & \ddots & & \\ & & \begin{bmatrix} 0 & & & & \\ -1 & 1 & & & \\ & -1 & 1 & & \\ & & \ddots & \ddots & \\ & & & -1 & 1 \end{bmatrix} & \end{bmatrix} \begin{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{Nnodes} \end{bmatrix}_{t_1} \\ \vdots \\ \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{Nnodes} \end{bmatrix}_{t_{n_S}} \end{bmatrix}$$

But what happens if we want to find $[\mathbf{D}]_t [\mathbf{u}]$? Off course we should find a way to first rearrange them in the order of temporal increasement instead of node increase-

ment. The *rearrangement operator* does this thing.

$$\begin{bmatrix} \begin{bmatrix} u_1(t_1) \\ u_1(t_2) \\ \vdots \\ u_1(t_{n_S}) \end{bmatrix} \\ \begin{bmatrix} u_2(t_1) \\ u_2(t_2) \\ \vdots \\ u_2(t_{n_S}) \end{bmatrix} \\ \vdots \\ \begin{bmatrix} u_{N_{nodes}}(t_1) \\ u_{N_{nodes}}(t_2) \\ \vdots \\ u_{N_{nodes}}(t_{n_S}) \end{bmatrix} \end{bmatrix} = \underbrace{\begin{bmatrix} \begin{bmatrix} 1 & & \\ & & \end{bmatrix} & & & \\ & \begin{bmatrix} 1 & \\ & \end{bmatrix} & & \\ & & \ddots & \\ & & & \begin{bmatrix} 1 & \\ & \end{bmatrix} \end{bmatrix} \\ \begin{bmatrix} \begin{bmatrix} & 1 \\ & \end{bmatrix} & & & \\ & \begin{bmatrix} & 1 \\ & \end{bmatrix} & & \\ & & \ddots & \\ & & & \begin{bmatrix} & 1 \\ & \end{bmatrix} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} \begin{bmatrix} & \\ 1 & \end{bmatrix} & & & \\ & \begin{bmatrix} & \\ 1 & \end{bmatrix} & & \\ & & \ddots & \\ & & & \begin{bmatrix} & \\ 1 & \end{bmatrix} \end{bmatrix} \end{bmatrix}}_{[\mathbf{\Omega_{21}}]} \begin{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{N_{nodes}} \end{bmatrix}_{t_1} \\ \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{N_{nodes}} \end{bmatrix}_{t_2} \\ \vdots \\ \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{N_{nodes}} \end{bmatrix}_{t_{n_S}} \end{bmatrix} \tag{2.4}$$

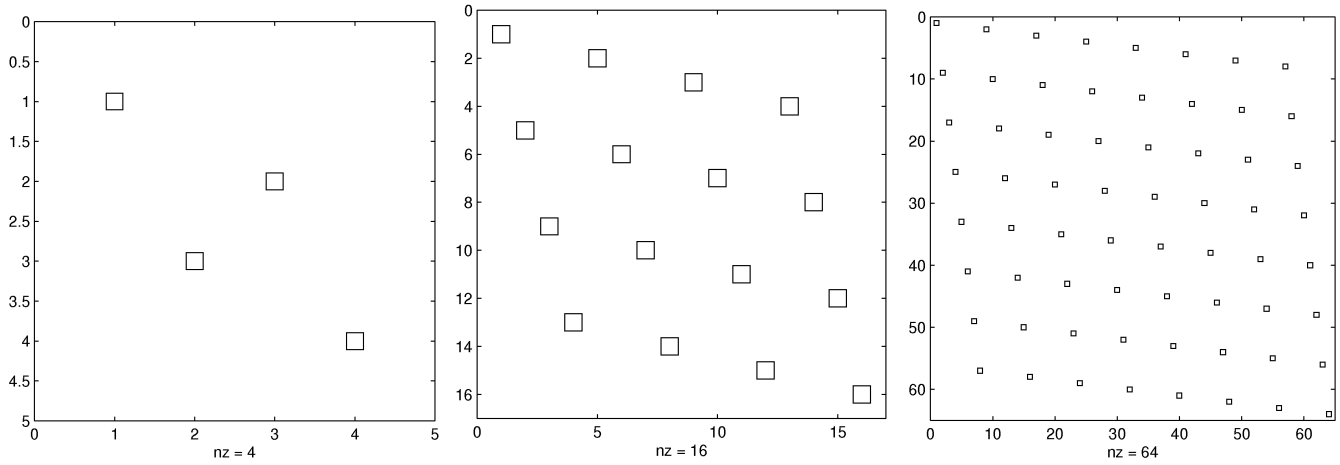


Figure 2.1.1: The rearrangement operator for $N_{nodes} = n_S$ and $nz = 2^{n_S}$.

Or in compact notation,

$$[\mathbf{u}]_2 = [\mathbf{\Omega}]_{21}[\mathbf{u}]_1 \tag{2.5}$$

Which rearranges the right-hand side nested vector $[\mathbf{u}]_1$ (rhs of (2.4)) which is originally in nodal order (order 1) to a nested vector $[\mathbf{u}]_2$ (lhs of (2.4)) which is in the order of time increasement (order 2). Obviously the rearrangement operators are

orthogonal, i.e.

$$\begin{aligned} [\mathbf{u}]_2 &= [\mathbf{\Omega}]_{21}[\mathbf{u}]_1 \\ \overset{[\mathbf{\Omega}]_{12}\times}{\Rightarrow} [\mathbf{\Omega}]_{12}[\mathbf{u}]_2 &= [\mathbf{\Omega}]_{12}[\mathbf{\Omega}]_{21}[\mathbf{u}]_1 = [\mathbf{u}]_1 \\ &\Rightarrow [\mathbf{\Omega}]_{12}[\mathbf{\Omega}]_{21} = \mathbb{I} \end{aligned} \tag{2.6}$$

Therefore we find the following important lemma which will be used in the proof of theorem II.

Lemma II - A rearrangement operator doesn't affect eigen values of its operand. i.e.,

$$\mathrm{eig}\left([\mathbf{\Omega}]_{12}\mathbb{A}[\mathbf{\Omega}]_{21}\right) = \mathrm{eig}\left(\mathbb{A}\right) \tag{2.7}$$

for arbitrary $\mathbb{A}$.

Proof:
Suppose that eigen-values of $[\mathbf{\Omega}]_{12}\mathbb{A}[\mathbf{\Omega}]_{21}$ are desired. In other words,

$$\left([\mathbf{\Omega}]_{12}\mathbb{A}[\mathbf{\Omega}]_{21}\right)[\mathbf{x}] = \lambda\,[\mathbf{x}] \tag{2.8}$$

Changing the variable $[\bar{\mathbf{x}}] = [\mathbf{\Omega}]_{21}\,[\mathbf{x}]$ leads to

$$[\mathbf{\Omega}]_{12}\mathbb{A}\,[\bar{\mathbf{x}}] = \lambda\,[\mathbf{x}] \tag{2.9}$$

Multiplying both sides of (2.9) with $[\mathbf{\Omega}]_{21}$ we will have

$$\left([\mathbf{\Omega}]_{21}[\mathbf{\Omega}]_{12}\right)\mathbb{A}\,[\bar{\mathbf{x}}] = \lambda\underbrace{[\mathbf{\Omega}]_{21}\,[\mathbf{x}]}_{[\bar{\mathbf{x}}]} \tag{2.10}$$

Using orthogonality property (2.6), eq. (2.10) is written as below

$$\mathbb{A}\,[\bar{\mathbf{x}}] = \lambda\,[\bar{\mathbf{x}}] \tag{2.11}$$

Equation (2.11) proves that any λ that is the eigen-value of $[\mathbf{\Omega}]_{12}\mathbb{A}[\mathbf{\Omega}]_{21}$ according to (2.8) is also the eigen-value of $\mathbb{A}$ and the proof is complete. ▲

Now we are ready to study the convergences of discrete Picard iteration method for general class of time-dependent partial differential equations.

Theorem II (The Super Theorem) - Discrete Picard Convergence Theorem for Time-Dependent Partial Differential Equations
Suppose that the time-dependent function f is expressed in the term of dependent variable u and its partial derivatives in z-dimensional space $\mathbb{R}_z = \prod_{i=1}^{z} \mathbb{R}_i$ such that

$$\begin{aligned}
\frac{\partial u}{\partial t} &= f(\mathbf{x}), \\
\mathbf{x} &= (u, t, \beta_1, \beta_2, \ldots, \beta_H) \\
\beta_1 &= \frac{\partial u}{\partial x_1},\ \beta_2 = \frac{\partial^2 u}{\partial x_1^2}, \ldots \\
\beta_{j_1} &= \frac{\partial u}{\partial x_2},\ \beta_{j_1+1} = \frac{\partial^2 u}{\partial x_2^2}, \ldots \\
&\vdots \\
\beta_j &= \frac{\partial^q (u)}{\partial {x_1}^{\omega_{j1}} \partial {x_2}^{\omega_{j2}} \ldots \partial {x_z}^{\omega_{jz}}} \quad \left(q = \sum_{k=1}^{z} \omega_{jk} \right),
\end{aligned} \tag{2.12}$$

where f is continuous and differentiable over $\mathbf{x} = [\mathbf{x_0}, \mathbf{x}]$ and is Lipschitz continuous over t and assume that the discretization of the integration operator $\mathbb{S}$ (1.6) is consistent, i.e. $p \geq 1$. Also assume that β_j is discretized in $\mathbb{R}_z$ such that,

$$\begin{aligned}
\beta_j &= \frac{\partial^q (u)}{\prod_{k=1}^{z} \partial {x_k}^{\omega_{jk}}} \\
&= \frac{1}{\prod_{k=1}^{z} \Delta {x_k}^{\omega_{jk}}} \sum_{l}^{\text{Stencil}} \gamma_{lj} u_l + \mathcal{O}\left(\Delta {x_1}^{P_{j1}}, \Delta {x_2}^{P_{j2}}, \ldots, \Delta {x_z}^{P_{jz}} \right)
\end{aligned} \tag{2.13}$$

Where $P_{j1\ldots z} \geq 1$ for consistency. Then *for the given operator* $\mathbb{S}$, the discrete Picard Iteration converges to a unique solution *if and only if*,

$$\left\| \lambda_i \left(\frac{\partial f}{\partial u}(\mathbf{x}_0) \Delta t + \mathrm{CFL}_i \right) \right\| \leq 1 \tag{2.14}$$

For all i where λ_i is the i^{th} eigen-value of $\mathbb{S}$ and CFL_i is the *local* generalized CourantFriedrichsLewy number defined as

$$\mathrm{CFL}_i = \Delta t \sum_{j=1}^{H} \frac{\frac{\partial f(\mathbf{x}_0)}{\partial \beta_j}}{\prod_{k=1}^{z} \Delta {x_k}^{\omega_{jk}}} \lambda_{\Gamma ji} \tag{2.15}$$

and $\lambda_{\Gamma ji}$ is the ith eigen-value of spatial discretization operator $\mathbf{\Gamma}_j = \gamma_{lj}$ matrix in (2.13).

Proof:

Since f is continuous and differentiable over $\mathbf{x}$, we can expand f near $\mathbf{x_0}$ using Taylor series,

$$f = f(\mathbf{x_0}) + \frac{\partial f}{\partial u}(\mathbf{x_0})\, u + \frac{\partial f}{\partial t}(\mathbf{x_0})\, t + \sum_{j=1}^{H} \frac{\partial f}{\partial \beta_j}(\mathbf{x_0})\, \beta_j, \tag{2.16}$$

Substituting β_j from (2.13) into (2.16) we will have,

$$f = f(\mathbf{x_0}) + \frac{\partial f}{\partial u}(\mathbf{x_0})\, u + \frac{\partial f}{\partial t}(\mathbf{x_0})\, t + \sum_{j=1}^{H} \frac{\partial f(\mathbf{x_0})}{\partial \beta_j} \frac{\partial^q (u)}{\prod_{k=1}^{z} \partial {x_k}^{\omega_{jk}}} \tag{2.17}$$

Now substituting the discretization of β_j from (2.13) into (2.17),

$$\begin{aligned} \frac{\partial u}{\partial t} = f = f(\mathbf{x_0}) + \frac{\partial f}{\partial u}(\mathbf{x_0})\, u + \frac{\partial f}{\partial t}(\mathbf{x_0})\, t + \sum_{j=1}^{H} \frac{\frac{\partial f(\mathbf{x_0})}{\partial \beta_j}}{\prod_{k=1}^{z} \Delta {x_k}^{\omega_{jk}}} \sum_{l} \gamma_{lj} u_l \\ + \sum_{j=1}^{H} \frac{\partial f(\mathbf{x_0})}{\partial \beta_j} \mathcal{O}\left(\Delta {x_1}^{P_{j1}}, \Delta {x_2}^{P_{j2}}, \ldots, \Delta {x_z}^{P_{jz}}\right) \end{aligned} \tag{2.18}$$

We note that (2.18) is valid for all nodes in z-dimensional space $\mathbb{R}_z$. Therefore if we repeat (2.18) for all nodes and combine the results in vector form we will have,

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial t} = f(\mathbf{x}_0)\, \mathbf{e} + \frac{\partial f}{\partial u}(\mathbf{x}_0)\, \mathbf{u} + \frac{\partial f}{\partial t}(\mathbf{x}_0)\, t\mathbf{e} + \sum_{j=1}^{H} \frac{\frac{\partial f(\mathbf{x_0})}{\partial \beta_j}}{\prod_{k=1}^{z} \Delta {x_k}^{\omega_{jk}}} [\mathbf{\Gamma}]_j \mathbf{u} \\ + \sum_{j=1}^{H} \frac{\partial f(\mathbf{x}_0)}{\partial \beta_j} \mathcal{O}\left(\Delta {x_1}^{P_{j1}}, \Delta {x_2}^{P_{j2}}, \ldots, \Delta {x_z}^{P_{jz}}\right) \mathbf{e} \end{aligned} \tag{2.19}$$

Where $\mathbf{u} = [u_1\ u_2\ \ldots\ u_{N_{nodes}}]^{\mathrm{T}}$ contains the nodal value of solution for all nodes at time t. Equation (2.19) may be rearranged as the following,

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial t} = \underbrace{\left(f(\mathbf{x}_0) + \frac{\partial f}{\partial t}(\mathbf{x}_0)\, t\right)}_{\mathbf{c_0'}} \mathbf{e} + \left(\frac{\partial f}{\partial u}(\mathbf{x}_0)\, [\mathbf{I}] + \sum_{j=1}^{H} \frac{\frac{\partial f(\mathbf{x_0})}{\partial \beta_j}}{\prod_{k=1}^{z} \Delta {x_k}^{\omega_{jk}}} [\mathbf{\Gamma}]_j\right) \mathbf{u} \\ + \sum_{j=1}^{H} \frac{\partial f(\mathbf{x}_0)}{\partial \beta_j} \mathcal{O}\left(\Delta {x_1}^{P_{j1}}, \Delta {x_2}^{P_{j2}}, \ldots, \Delta {x_z}^{P_{jz}}\right) \mathbf{e} \end{aligned} \tag{2.20}$$

or

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{c_0'} + \left(\frac{\partial f}{\partial u}(\mathbf{x}_0)[\mathbf{I}] + \sum_{j=1}^{H} \frac{\frac{\partial f(\mathbf{x}_0)}{\partial \beta_j}}{\prod_{k=1}^{z} \Delta x_k{}^{\omega_{jk}}} [\mathbf{\Gamma}]_j \right) \mathbf{u}$$
$$+ \sum_{j=1}^{H} \frac{\partial f(\mathbf{x}_0)}{\partial \beta_j} \mathcal{O}\left(\Delta x_1{}^{P_{j1}}, \Delta x_2{}^{P_{j2}}, \ldots, \Delta x_z{}^{P_{jz}}\right) \mathbf{e} \tag{2.21}$$

We note that since $[\mathbf{\Gamma}]_j$ is *not* diagonal in general, then (2.21) should be diagonalized first. Before doing eigen-value decomposition we note the following interesting property of matrix summation. Assume the we define

$$\left[\bar{\mathbf{\Gamma}}\right] = \sum_{j=1}^{H} \frac{\frac{\partial f(\mathbf{x}_0)}{\partial \beta_j}}{\prod_{k=1}^{z} \Delta x_k{}^{\omega_{jk}}} [\mathbf{\Gamma}]_j \tag{2.22}$$

To rewrite the RHS of (2.21) as the following[1]

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{c_0'} + \left(\frac{\partial f}{\partial u}(\mathbf{x}_0)[\mathbf{I}] + \left[\bar{\mathbf{\Gamma}}\right] \right) \mathbf{u}$$
$$+ \sum_{j=1}^{H} \frac{\partial f(\mathbf{x}_0)}{\partial \beta_j} \mathcal{O}\left(\Delta x_1{}^{P_{j1}}, \Delta x_2{}^{P_{j2}}, \ldots, \Delta x_z{}^{P_{jz}}\right) \mathbf{e} \tag{2.23}$$

So to diagonalize (2.21) we need to find the eigen-value decomposition of $\left[\bar{\mathbf{\Gamma}}\right]$. Assume that the eigen-matrix of $\left[\bar{\mathbf{\Gamma}}\right]$ is $[\mathbf{\Lambda}_{\bar{\mathbf{\Gamma}}}]$. Then for any left eigen-vector $[\mathbf{L}]$ we have

$$[\mathbf{L}]\left[\bar{\mathbf{\Gamma}}\right] = [\mathbf{\Lambda}_{\bar{\mathbf{\Gamma}}}][\mathbf{L}] \tag{2.24}$$

Substituting for $\left[\bar{\mathbf{\Gamma}}\right]$ from (2.22) we will have,

$$\sum_{j=1}^{H} \frac{\frac{\partial f(\mathbf{x}_0)}{\partial \beta_j}}{\prod_{k=1}^{z} \Delta x_k{}^{\omega_{jk}}} [\mathbf{L}][\mathbf{\Gamma}]_j = [\mathbf{\Lambda}_{\bar{\mathbf{\Gamma}}}][\mathbf{L}] \tag{2.25}$$

or

$$\sum_{j=1}^{H} \frac{\frac{\partial f(\mathbf{x}_0)}{\partial \beta_j}}{\prod_{k=1}^{z} \Delta x_k{}^{\omega_{jk}}} [\mathbf{\Lambda}_{\mathbf{\Gamma}}]_j [\mathbf{L}] = [\mathbf{\Lambda}_{\bar{\mathbf{\Gamma}}}][\mathbf{L}] \tag{2.26}$$

[1]It is helpful to note that $\left[\bar{\mathbf{\Gamma}}\right]$ in (2.22) is actually the discretized form of residuals $f(\mathbf{x})$ in (2.12) for only **linear** case. We use this definition in the following sections when we want to study the stability and convergence of implicit form of DPI.

where $[\mathbf{\Lambda_\Gamma}]_j$ is the eigen-matrix of $[\mathbf{\Gamma}]_j$. From (2.26) it is evident that

$$\sum_{j=1}^{H} \frac{\frac{\partial f(\mathbf{x}_0)}{\partial \beta_j}}{\prod_{k=1}^{z} \Delta {x_k}^{\omega_{jk}}} [\mathbf{\Lambda_\Gamma}]_j = [\mathbf{\Lambda_{\bar{\Gamma}}}] \tag{2.27}$$

Now, by multiplying both sides of (2.23) with the left eigen vector $[\mathbf{L}]_{\bar{\mathbf{\Gamma}}}$ and substituting (2.27) and using new variable $\bar{\mathbf{u}} = [\mathbf{L}]\mathbf{u}$ we obtain,

$$\begin{aligned} \frac{\partial \bar{\mathbf{u}}}{\partial t} = \underbrace{[\mathbf{L}]\mathbf{c_0'}}_{\mathbf{c_0}} + \underbrace{\left(\frac{\partial f}{\partial u}(\mathbf{x}_0)\,[\mathbf{I}] + \sum_{j=1}^{H} \frac{\frac{\partial f(\mathbf{x}_0)}{\partial \beta_j}}{\prod_{k=1}^{z} \Delta {x_k}^{\omega_{jk}}} [\mathbf{\Lambda_\Gamma}]_j \right)}_{[\mathbf{c}]} \bar{\mathbf{u}} \\ + \sum_{j=1}^{H} \frac{\partial f(\mathbf{x}_0)}{\partial \beta_j} \mathcal{O}\left(\Delta {x_1}^{P_{j1}}, \Delta {x_2}^{P_{j2}}, \ldots, \Delta {x_z}^{P_{jz}}\right) \underbrace{[\mathbf{L}]\mathbf{e}}_{\bar{\mathbf{e}}} \end{aligned} \tag{2.28}$$

or

$$\frac{\partial \bar{\mathbf{u}}}{\partial t} = \mathbf{c_0} + [\mathbf{c}]\,\bar{\mathbf{u}} + \sum_{j=1}^{H} \frac{\partial f(\mathbf{x}_0)}{\partial \beta_j} \mathcal{O}\left(\Delta {x_1}^{P_{j1}}, \Delta {x_2}^{P_{j2}}, \ldots, \Delta {x_z}^{P_{jz}}\right) \bar{\mathbf{e}} \tag{2.29}$$

We note that (2.29) is valid for all nodes in the spatial domain at time t. Therefore (2.29) should be repeated for discrete times $t_i = \{t_1, t_2, \ldots, t_{n_S}\}$ and the resulting system of equations should be packed using space-time data structure represented in (2.1). The result is presented as follows

$$\frac{\partial [\bar{\mathbf{u}}]}{\partial t} = [\mathbf{c_0}] + \mathrm{diag}([\mathbf{c}])\,[\bar{\mathbf{u}}] + \sum_{j=1}^{H} \mathrm{diag}\left(\frac{\partial f(\mathbf{x}_0)}{\partial \beta_j}\right) \mathcal{O}\left(\Delta {x_1}^{P_{j1}}, \Delta {x_2}^{P_{j2}}, \ldots, \Delta {x_z}^{P_{jz}}\right) \bar{\mathbf{e}} \tag{2.30}$$

Now we use rearrangement operator combined to integration operator $[\mathbf{\Omega}]_{12}\mathbb{S}[\mathbf{\Omega}]_{21}$ to discretize $\frac{\partial}{\partial t}$ in (2.30) as the following

$$\begin{aligned} & [\bar{\mathbf{u}}]_{n+1} = [\bar{\mathbf{u}}]_0 + \Delta t [\mathbf{\Omega}]_{12} \mathrm{diag}(\mathbb{S})\,[\mathbf{\Omega}]_{21}\,[\mathbf{c_0}] + (\Delta t [\mathbf{\Omega}]_{12} \mathrm{diag}(\mathbb{S})\,[\mathbf{\Omega}]_{21} \mathrm{diag}([\mathbf{c}]))\,[\bar{\mathbf{u}}]_n \\ + & \sum_{j=1}^{H} \Delta t [\mathbf{\Omega}]_{12} \mathrm{diag}(\mathbb{S})\,[\mathbf{\Omega}]_{21} \mathrm{diag}\left(\frac{\partial f(\mathbf{x}_0)}{\partial \beta_j} \bar{\mathbf{e}}\right) \mathcal{O}\left(\Delta {x_1}^{P_{j1}}, \Delta {x_2}^{P_{j2}}, \ldots, \Delta {x_z}^{P_{jz}}\right) \\ + & \mathcal{O}(\Delta t^p)\,\mathbf{e}, \end{aligned} \tag{2.31}$$

Equation (2.31) is analogous to (1.18) except the truncation error $\mathcal{O}\left(\Delta {x_1}^{P_{j1}}, \Delta {x_2}^{P_{j2}}, \ldots, \Delta {x_z}^{P_{jz}}\right)$ and eigen value matrix $\mathrm{diag}([\mathbf{c}])$ due to spatial discretization does not appear in the

ODE iteration (1.18). We also note that using the assumption of Lipschitz continuity we can show (as we did in theorem I) that $[\mathbf{c_0}]$ in (2.31) is bounded for arbitrary iteration n. Also the assumption of consistency of temporal and spatial discretization given in theorem leads to $p = P_{j1} = P_{j2} = \ldots = P_{jz} \geq 1$ which shows that spatial and temporal truncation errors in (2.31) vanish as spatial and temporal spacing become infinitesimal. Therefore by using the same geometric series approach of Theorem I, it is evident that the generic discrete Picard iteration for PDE (2.31) converges *if and only if* the eigen values of the coefficient matrix of $[\bar{\mathbf{u}}]_n$ in (2.31) are less that unity, i.e.,

$$\left\| \Delta t \;\; \mathbf{eig}([\mathbf{\Omega}]_{12} \mathrm{diag}\left(\mathbb{S}\right) [\mathbf{\Omega}]_{21} \mathrm{diag}\left([\mathbf{c}]\right)) \right\| \leq 1 \tag{2.32}$$

To find the eigen-values, we note that any eigen matrix of the coefficient matrix satisfies the following equation,

$$([\mathbf{\Omega}]_{12} \mathrm{diag}\left(\mathbb{S}\right) [\mathbf{\Omega}]_{21} \mathrm{diag}\left([\mathbf{c}]\right)) [\mathbf{x}] = [\mathbf{\Theta}] [\mathbf{x}], \tag{2.33}$$

Where $[\mathbf{\Theta}]$ is the eigen-matrix of $[\mathbf{\Omega}]_{12} \mathrm{diag}\left(\mathbb{S}\right) [\mathbf{\Omega}]_{21} \mathrm{diag}\left([\mathbf{c}]\right)$. Substituting new variable

$$[\bar{\mathbf{x}}] = \mathrm{diag}\left([\mathbf{c}]\right) [\mathbf{x}], \tag{2.34}$$

into (2.33) we will have

$$[\mathbf{\Omega}]_{12} \mathrm{diag}\left(\mathbb{S}\right) [\mathbf{\Omega}]_{21} [\bar{\mathbf{x}}] = [\mathbf{\Theta}] [\mathbf{x}], \tag{2.35}$$

Since $\mathrm{diag}\left([\mathbf{c}]\right)$ is a diagonalized matrix of diagonal matrix $[\mathbf{c}]$ in (2.28) then

$$[\mathbf{x}] = \mathrm{diag}\left(\left[\mathbf{c}^{-1}\right]\right) [\bar{\mathbf{x}}], \tag{2.36}$$

Substituting (2.36) into (2.35),

$$[\mathbf{\Omega}]_{12} \mathrm{diag}\left(\mathbb{S}\right) [\mathbf{\Omega}]_{21} [\bar{\mathbf{x}}] = [\mathbf{\Theta}] \mathrm{diag}\left(\left[\mathbf{c}^{-1}\right]\right) [\bar{\mathbf{x}}], \tag{2.37}$$

Or

$$\mathbf{eig}\Big([\mathbf{\Omega}]_{12} \mathrm{diag}\left(\mathbb{S}\right) [\mathbf{\Omega}]_{21} \Big) = [\mathbf{\Theta}] \mathrm{diag}\left(\left[\mathbf{c}^{-1}\right]\right), \tag{2.38}$$

Equation (2.38) proves that the diagonal matrix $[\mathbf{\Theta}] \mathrm{diag}\left(\left[\mathbf{c}^{-1}\right]\right)$ is the eigen-matrix of $[\mathbf{\Omega}]_{12} \mathrm{diag}\left(\mathbb{S}\right) [\mathbf{\Omega}]_{21}$. Since rearrangement operator $[\mathbf{\Omega}]$ doesn't change the eigen-values of its operand (Lemma II), we can rewrite (2.38) as

$$\mathbf{eig}\Big([\mathbf{\Omega}]_{12} \mathrm{diag}\left(\mathbb{S}\right) [\mathbf{\Omega}]_{21} \Big) = \mathbf{eig}\Big(\mathrm{diag}\left(\mathbb{S}\right) \Big) = [\mathbf{\Theta}] \mathrm{diag}\left(\left[\mathbf{c}^{-1}\right]\right), \tag{2.39}$$

Since the integration operator $\mathbb{S}$ is diagonalized, the eigen values are same as (1.23) but repeated. Hence

$$\mathbf{eig}\Big(\operatorname{diag}(\mathbb{S})\Big) = \operatorname{diag}\Big(\mathbf{eig}(\mathbb{S})\Big) = [\mathbf{\Theta}]\operatorname{diag}\left(\left[\mathbf{c}^{-1}\right]\right), \tag{2.40}$$

Substituting (1.23) into (2.40) we obtain

$$\operatorname{diag}\Big(\mathbf{eig}(\mathbb{S})\Big) = \operatorname{diag}(\mathbf{\Lambda}) = [\mathbf{\Theta}]\operatorname{diag}\left(\left[\mathbf{c}^{-1}\right]\right), \tag{2.41}$$

Which gives us the unknow eigen-matrix $[\mathbf{\Theta}]$,

$$[\mathbf{\Theta}] = \operatorname{diag}(\mathbf{\Lambda})\operatorname{diag}([\mathbf{c}]), \tag{2.42}$$

Substituting (2.42) into convergence condition (2.32) we will have,

$$\Big\|\Delta t \;\; \operatorname{diag}(\mathbf{\Lambda})\operatorname{diag}([\mathbf{c}])\Big\| \leq 1 \tag{2.43}$$

We note that entries on the diagonal matrix $\operatorname{diag}([\mathbf{c}])$ are given in (2.28) as $c_i = \frac{\partial f}{\partial u}(\mathbf{x}_0) + \sum_{j=1}^{H} \frac{\frac{\partial f(\mathbf{x}_0)}{\partial \beta_j}}{\prod_{k=1}^{z} \Delta x_k{}^{\omega_{jk}}} \lambda_{\Gamma i}$. Substituting into (2.43) we will have

$$\Big\|\Delta t \;\; \lambda_i\Big(\frac{\partial f}{\partial u}(\mathbf{x}_0) + \sum_{j=1}^{H} \frac{\frac{\partial f(\mathbf{x}_0)}{\partial \beta_j}}{\prod_{k=1}^{z} \Delta x_k{}^{\omega_{jk}}} \lambda_{\Gamma ji}\Big)\Big\| \leq 1 \tag{2.44}$$

Since $\Delta t \;\sum_{j=1}^{H} \frac{\frac{\partial f(\mathbf{x}_0)}{\partial \beta_j}}{\prod_{k=1}^{z} \Delta x_k{}^{\omega_{jk}}} \lambda_{\Gamma ji}$ is analogous to one dimensional CFL number $\mathrm{CFL} = c\Delta t/\Delta x$, we defined it as the generalized CFL_i number for temporal node "i". Note that this definition of CFL number consistently switches to the traditional definition of CFL number for one, two and three dimensional space. Substituting CFL_i into (2.44) we obtain

$$\Big\|\lambda_i\Big(\frac{\partial f}{\partial u}(\mathbf{x}_0)\Delta t + \mathrm{CFL}_i\Big)\Big\| \leq 1 \tag{2.45}$$

And the proof is complete. ▲

2.1.1 Special cases of Super Theorem II

Theorem II is a super theorem containing all possible forms for a differential equation. Here we study special cases which are literally important. The first special case is the case of ODEs where all derivative groups $\beta_j = 0$ in (2.12). Therefore we have $\frac{\partial f}{\partial u}(\mathbf{x}_0) = 0$ and hence $CFL = 0$ according to (2.15). Therefore the convergence

condition (2.14) reduces to Theorem I. The second special case is the one-dimensional linear propagation (convection)

$$\begin{aligned} \frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} &= 0, \\ u\left(x, t=0\right) &= u_{\text{init}} \\ u\left(x=0, t\right) &= u_{\text{init}}\left(x=0, t\right). \end{aligned} \tag{2.46}$$

Where $x = x_1$ is one-dimensional space. Assume that first-order upwind operator (2.2) is used for *spatial* discretization. Therefore $\lambda_{\Gamma ji} = 1$ in (2.15). According to theorem II, for this case we have $f = -c\beta_1$. Also since the equation is one-dimensional we have $\omega_{jk} = 0$ except for $k = j = 1$ which is $\omega_{11} = 1$ therefore the CFL condition (2.15) reduces to

$$\text{CFL} = \Delta t \sum_{j=1}^{H} \frac{\frac{\partial f(\mathbf{x}_0)}{\partial \beta_j}}{\prod_{k=1}^{z} \Delta x_k{}^{\omega_{jk}}} \lambda_{\Gamma ji} = \Delta t \, \frac{\frac{\partial f(\mathbf{x}_0)}{\partial \beta_1}}{\prod_{k=1}^{1} \Delta x_k{}^{\omega_{1k}}} \lambda_{\Gamma 1i} = \Delta t \, \frac{(-c)}{\Delta x_1} \tag{2.47}$$

Substituting (2.47) into Theorem II- (2.14), we will have

$$\begin{aligned} \left\| \lambda_i \left(\frac{\partial f}{\partial u}(\mathbf{x}_0) \Delta t + \text{CFL} \right) \right\| &\leq 1 \\ \rightarrow \left\| \lambda_i \left(0 \times \Delta t + \Delta t \, \frac{(-c)}{\Delta x_1} \right) \right\| &\leq 1 \\ \rightarrow \left| \Delta t \, \frac{c}{\Delta x_1} \right| &\leq \frac{1}{\lambda_i} \end{aligned} \tag{2.48}$$

Which is the *local* stability condtion for node 'i'. Therefore the global stability condition is obtained as

$$\left| \text{CFL} = \Delta t \, \frac{c}{\Delta x_1} \right| \leq \frac{1}{\lambda_{\max}} \tag{2.49}$$

Now if we use the Riemann operator (1.9) for integration, then $\lambda_{\max} = 1$ in (2.49) and we define CFL_0 as

$$\left| \text{CFL}_0 = \Delta t \, \frac{c}{\Delta x_1} \right| \leq 1 \tag{2.50}$$

The value of CFL_0 is equivalent to CFL number of numerical solution to (2.46) which is obtained by applying first-order upwind differencing for spatial derivatives

and Euler explicit stepping for temporal derivatives. Therefore CFL_0 is chosen as a *refrence point* in this work. We note that the ratio of maximum CFL obtained by Discrete Picard Iteration (2.49) to the maximum refrence CFl number (2.50) is

$$\left(\frac{\mathrm{CFL}}{\mathrm{CFL}_0}\right)_{\max} = \frac{1}{\lambda \max} \tag{2.51}$$

Which was presented before in fig.(1.2.2). Interestingly, this ratio goes up to 60 for the developed integration operators.

2.2 Numerical Tests for Partial Differential Equations

2.2.1 One-dimensional Linear Propagation

We solve equation (2.46) using first-order and second-order (upwind) finite difference scheme for discretization of spatial derivatives and Euler explicit method for time marching.

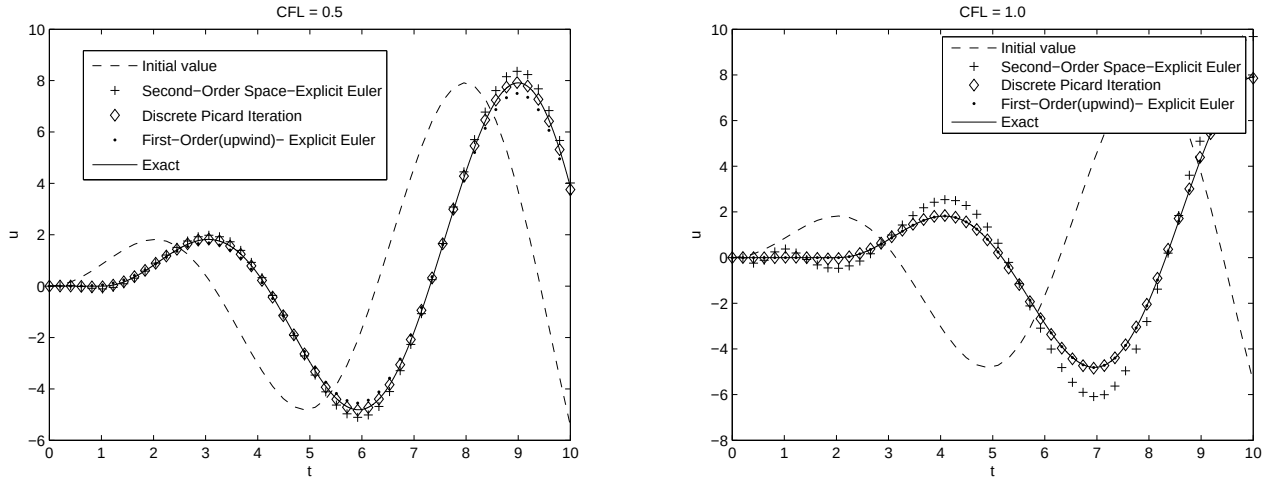
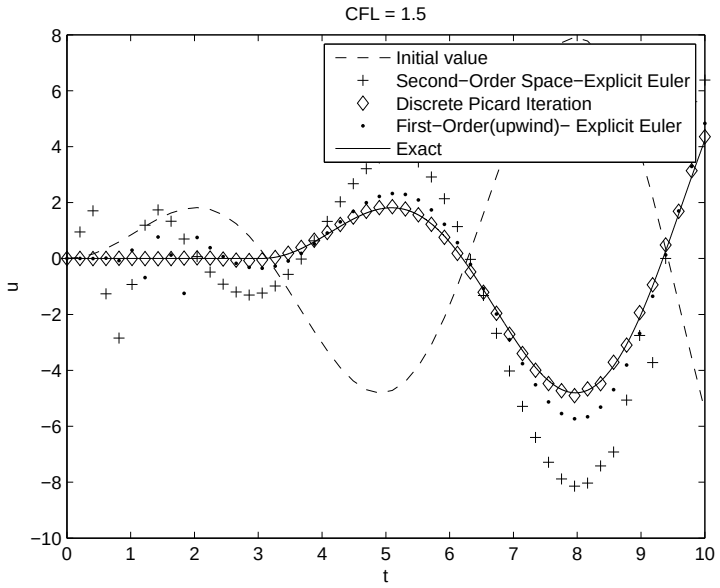
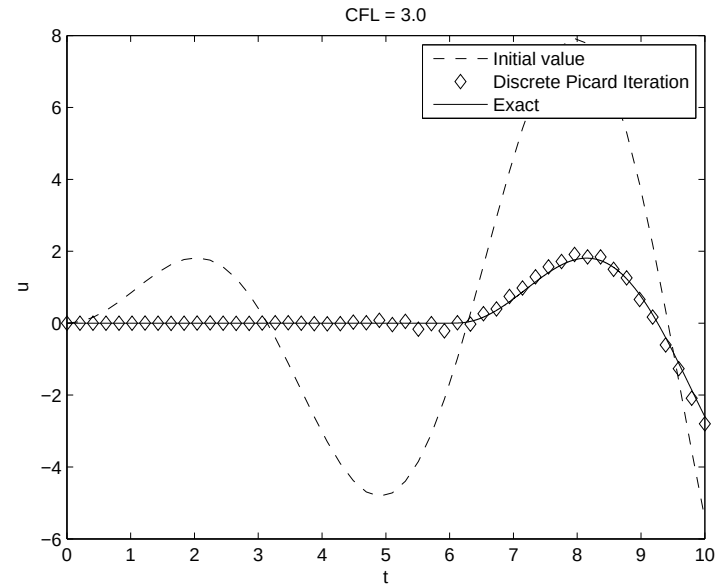


Figure 2.2.1: One-Dimensional advection.

The results are compared with the discrete Picard iteration obtained by using stretching strenght $\alpha = 1.01$ and $n_S = 7$. The initial function $u_{\mathrm{init}} = x\sin(x)$ is selected for eq.(2.46). The wave speed $c = 0.0001$ for fifty nodes in the spatial grid are used in all solutions. The exact solution is obtained using a spectral method (Trefethen (2001)).





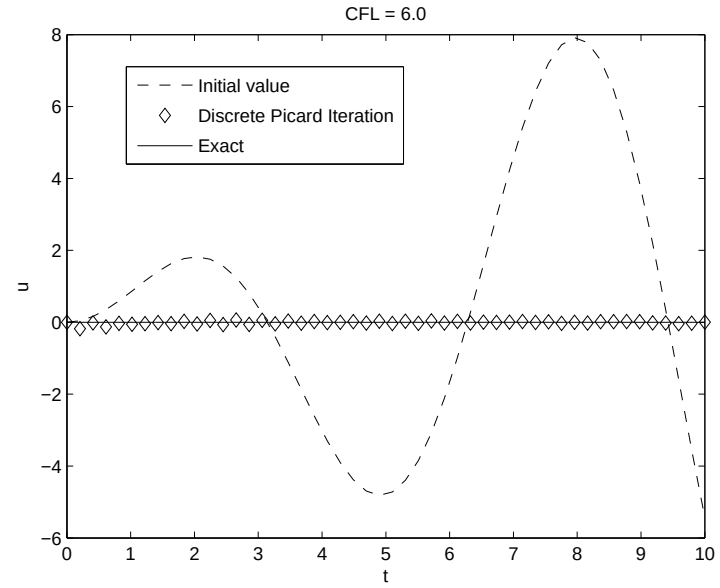


Figure 2.2.2: One-Dimensional advection. (Continued)

As shown in figs.(2.2.1-2.2.2), the fisrt-order and second order methods with Euler explicit lead to unacceptable dissipation error for CFL = 0.5. Also these method are unstable at $CFL > 1$. The discrete Picard iteration is very accurate for $CFL < 1$. The stability and accuracy of DPI method is clrealy shown in fig.(2.2.2-bottom) for $CFL = 6.0$.

Chapter 3

Alternative Discrete Picard Iterations

In an attempt to increase the convergence and/or stability region of DPI, we continue our quest to examine different alternative forms of Picard iterations. Without loosing the generality even for general PDE (2.12), let us rewrite the jth entry of iterative equation (1.27) in a more straight form by neglecting all constants

$$u_n = u_0 + r_j u_{n-1} \tag{3.1}$$

According to theorem (I), we have noticed that additional constants in (1.27) have no effect on the stability of Picard iterations and they are only important to evaluate the truncation error and final solution. Therefore we omitted everything that doesn't have any effect on the stability and hence (1.27) is reduced to (3.1).
Equation (3.1) is the *local* stability equation for the jth node in the solution vector $\mathbf{u}$. To achieve a global stability equation we simply consider that maximum eigen-value,

$$u_n = u_0 + r u_{n-1} \tag{3.2}$$

where $r = c\Delta t \lambda_{\max}$ is the stability number (according to Theorem (I)). Now we replace the previous solution u_{n-1} in the original Picard iteration (1.18) by *various alternatives* to examine different stability characteristics and possibly new interesting properties.
Let us discuss the first form which is more intuitive.

3.1 DPI with averaging of previously stored solutions

If we replace the previous solution u_{n-1} in the rhs of (3.2) with the weighted average of the two last solutions we obtain the sequence,

$$u_n \;=\; u_0 + r\frac{(au_{n-1} + bu_{n-2})}{(a+b)},$$

$$u_0 \;=\; 0,\; u_1 = w_0, \tag{3.3}$$

which is solved with the given initial conditions leading to the following non-dimensional closed form solution,

$$\begin{aligned}
\frac{u_n}{w_0} &= \frac{1}{1-r} + \frac{\left(r(a+2\,b) - \sqrt{r\,(ra^2+4\,ba+4\,b^2)}\right)}{2\,(1-r)\,\sqrt{r\,(ra^2+4\,ba+4\,b^2)}} r_1^n \\
&- \frac{\left(r(a+2\,b) + \sqrt{r\,(ra^2+4\,ba+4\,b^2)}\right)}{2\,(1-r)\,\sqrt{r\,(ra^2+4\,ba+4\,b^2)}} r_2^n \\
r_1 &= \frac{-2rb}{ra + \sqrt{r\,(ra^2+4\,ba+4\,b^2)}} \\
r_2 &= \frac{-2rb}{ra - \sqrt{r\,(ra^2+4\,ba+4\,b^2)}}
\end{aligned} \tag{3.4}$$

We readily note that the stability number of the original Picard iteration (1.18), i.e. $r = c\;\Delta t\;\max\left(\mathrm{eig}\left(\mathbb{S}\right)\right)$ is now changed to two stability numbers r_1 and r_2 in (3.4) which are functions of the original stability number r. For this reason we call them the modified stability number $\bar{r}$. The stability condition for (3.4) states that $|r_{1,2}| \leq 1$. In other words

$$\left| \bar{r} = \frac{-2rb}{ra \pm \sqrt{r\,(ra^2+4\,ba+4\,b^2)}} \right| \leq 1 \tag{3.5}$$

In figure(3.1.1) the modified stability number is plotted for different values of a and b in the region of the convergence (between 0 and 1). Outside of this region, at least one of the modified stability numbers $|\bar{r}_{1,2}|$ is greater than one thus (3.3) is divergent.

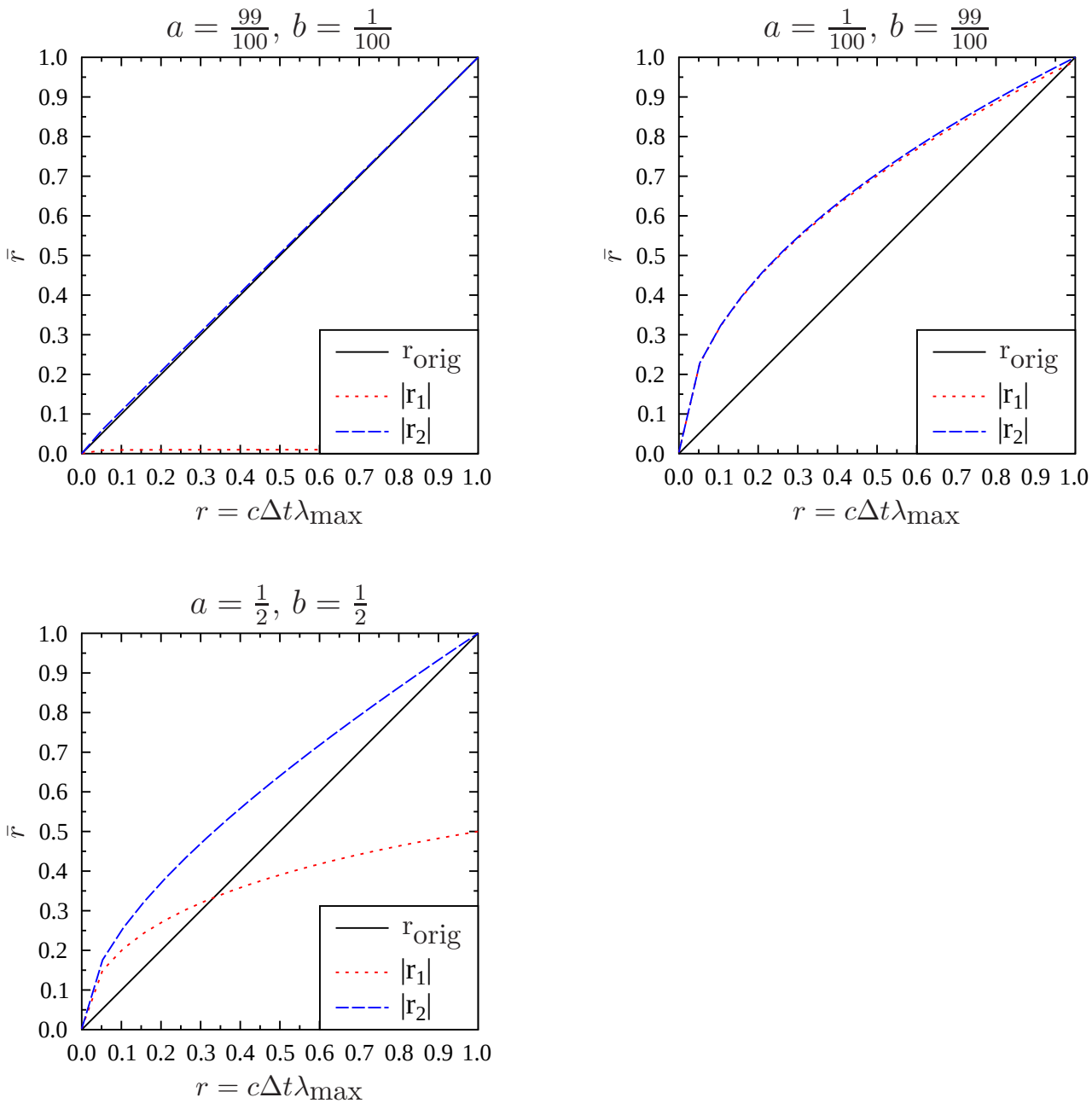


Figure 3.1.1: The modified stability number $\bar{r}$ versus the original number r for DPI with explicit averaging.

According to Lemma (I), the Discrete Picard Iteration converges faster when the stability number r is smaller. Therefore, the favorite modification to the original scheme is the one which leads to smaller modified stability number. As shown in fig.(3.1.1), $|\bar{r}_2|$ is always greater than the original stability number r_{orig} while $|\bar{r}_1|$ is varying between $\bar{r} = |\bar{r}_2|$ and $\bar{r} = 0$. That means for arbitrary values r, a and b, the sequence corresponding to $\bar{r} = |\bar{r}_1|$ (on the rhs of (3.4)) might converge faster than

original Picard iteration because $|r_1|$ might be smaller (as in the plot for $a = 1/2$ and $b = 1/2$ and $r = 0.9$) but meanwhile, the sequence corresponding to $\bar{r} = |\bar{r}_2|$ will converge slower because $|r_2|$ is always greater than $\mathrm{r_{orig}}$ and hences we conclude that the total sequence (3.4) will converge slower than the original Picard iteration. So we leave this section with the conclusion that the averaging of the previously stored solutions (explicit averaging) has no effect on improving the stability and/or accuracy of the method and it degrades the convergence.

3.1.1 Applications: DPI with extrapolating previously stored solutions

Now we consider an application of explicit averaging DPI method eq.(3.3). Assume that the solution u_{n-1} in (3.2) is obtained by linear extrapolation of the previously stored solutions at iterations $n-1$ and $n-2$. The extrapolation between to successive iterations is shown fig.(3.1.2) for the next solution 'u' at iteration 'x'.

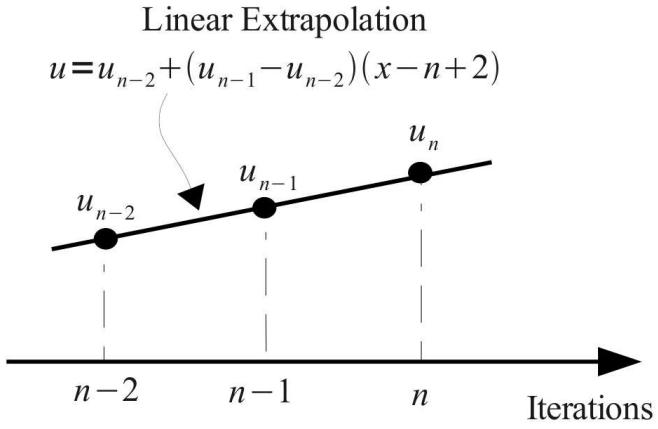


Figure 3.1.2: The linear extrapolation between successive iterations.

For $x = n$, we find the following expression for u_n.

$$u_n = 2u_{n-1} - u_{n-2} \tag{3.6}$$

Substituting into the rhs of (3.2) for u_{n-1} we obtain,

$$u_n = u_0 + r\,(2u_{n-1} - u_{n-2}) \tag{3.7}$$

We note that (3.7) is a special case of (3.3) for a=2 and b = -1. Therefore from Sec. (3.1) we conclude that the extrapolation will not improve stability and/or convergence of the original Picard iterations.

3.2 DPI with implicit averaging

In the previous section, we observed that averaging strategy between previously stored solutions was not successful in improving the convergence of DPI. However in this section we consider the possibility of averaging between **current solution** and previously stored ones. This is the implicit form of averaging because the solution at each iteration depends on **itself** and a history of previously stored solutions. We replace u_{n-1} in rhs (3.2) with $(au_n + bu_{n-1}) / (a + b)$ yielding

$$\begin{aligned} u_n &= u_0 + r\frac{(bu_{n-1} + au_n)}{(a+b)} \\ u_1 &= w_0 \end{aligned} \tag{3.8}$$

Actually, a combination of implicit-explicit solution can be observed in (3.8) since for $a = 1$ and $b = 0$ we have a purely implicit scheme and for $a = 0$ and $b = 1$ we have a purely explicit scheme which is the original Picard iterations. The following is the closed-form solution of sequence (3.8).

$$\frac{u_n}{w_0} = \frac{a(r-1)\left(\frac{rb}{(1-r)a+b}\right)^n - b\left(\left(\frac{rb}{(1-r)a+b}\right)^n - 1\right)}{b\,(1-r)} \tag{3.9}$$

From (3.9) we readily find the modified stability number is

$$\bar{r} = \frac{rb}{(1-r)a + b} \tag{3.10}$$

Figure (3.2.1) contains a detailed discussion of modified stability number (3.10) for a wide range of r. As shown, for $0 \leq r \leq 1$, increasing the value of a decreases $\bar{r}$ thus increases the convergence (by reducing the number of required iterations according to Lemma (I)- eq.(1.47)). Since a is the coefficient of u_n, we conclude that increasing a is actually increasing the implicit nature of (3.8) and thus the more implicit (3.8) is solved, the more convergent the solution is. By increasing the implicit nature, i.e. $a \to 1, b \to 0$, the modified stability number converges to zero $\bar{r} \to 0$ and hence we find a golden case that the solution converges at the first iteration (because (3.8) is no longer iterative). For purely explicit case $a = 0, b = 1$ we retrieve the original Picard iterations which is the 45 degrees straight line. Therefore for all values of a and b over $0 \leq r \leq 1$, the implicit scheme is between the golden iteration and the Picard iteration.

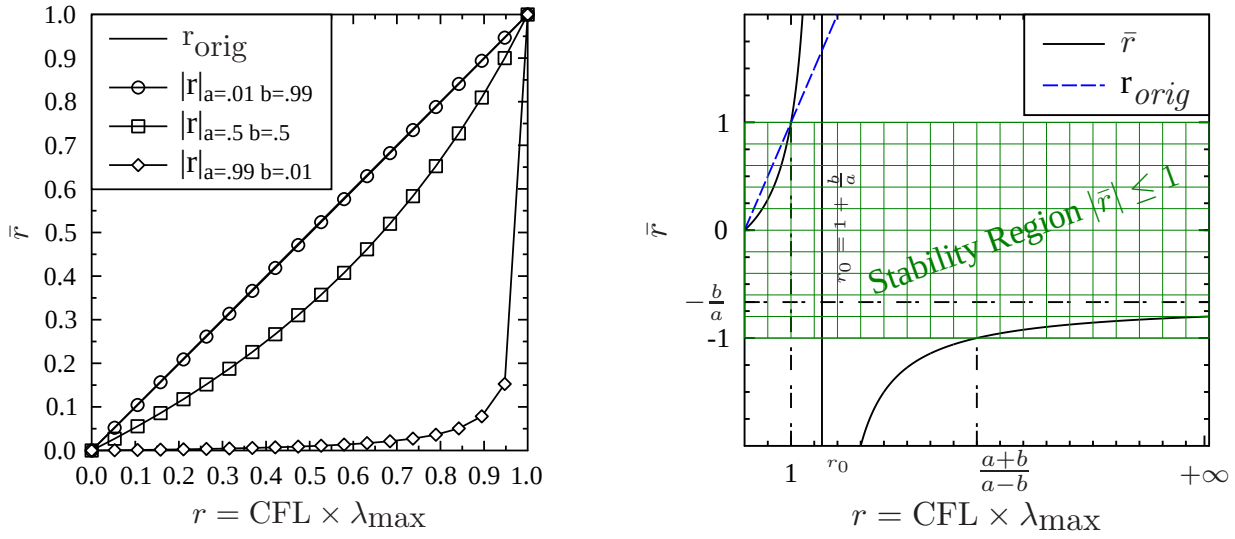


Figure 3.2.1: The modified stability number $\bar{r}$ versus the original number r for DPI with implicit averaging. Left) close up for $0 \leq r \leq 1$. Right) The parametric plot for all stability numbers $0 \leq r \leq +\infty$. The original stability number is $r = c\Delta t \lambda_{\max}$ for ODEs (Theorem I, eq.(TheoremI)) and $r = \text{CFL}\lambda_{\max}$ for PDEs (Theorem II).

To find the regions of stability of implicit DPI, we prepared a parametric plot of (3.10) in fig.(3.2.1)-right. As shown, for arbitrary values of a and b, the implicit DPI (the solid line) escapes from the stability region at $r = 1$. But it miraculously renters to the stability region at $r = (a + b) / (a - b)$ and remains stable unconditionally after that. That means for the case of general time dependent (time evolution) Partial Differential Equation (2.12) where $r = \text{CFL}\lambda_{\max}$, the implicit DPI proposed in (3.8) remains stable for arbitrarily large CFL without needing to modify the integration operator as we did in Sec.(1.2) for the explicit DPI.
The implicit DPI has a vertical asymptote (vertical solid line) at the middle of instability region, i.e., $r_0 = 1 + b/a$, and an horizontal asymptote $\bar{r}_0 = -b/a$ (horizontal dotted dashed line) which is the value of the modified stability number (3.10) for sufficiently large CFL number. The stability number of original Picard Iterations is also brought in fig.(3.2.1)-right (the oblique dashed line). As shown, it is unconditionally unstable for $r > 1$ and also it is always greater than or equal to the implicit DPI. That means in the region of stability, the original explicit DPI is less convergent (in the sense of number of required according to eq.(1.47)) than the implicit version.

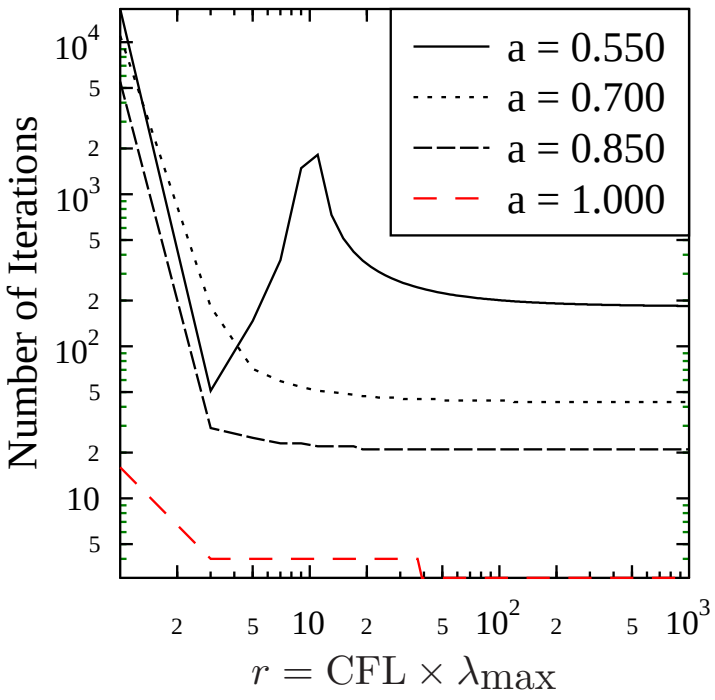


Figure 3.2.2: The number of iterations nedded for convergence (p=52 in Table ()) for DPI with implicit averaging.

A more descriptive account of convergence of implicit DPI is brought in fig.(3.2.2) where eq.(1.47) is evaluated with $\bar{r}$ given in (3.10) instead of r. As $a \to 1$ and $b \to 0$, the number of iteration required for convergences to machine zero (defined by $p = 52$ in Table(1.1.1)) reduces to less than 5. (see the dashed red line)
It should be noted that using implicit DPI is one way to accelerate the convergence. Since we deal with nonlinear series in the DPI method, another potential approach to improve the convergence is to use Series Convergence Acceleration Methods like Aitken's methodSidi (2003) to directly *accelerate* the convergence of series in DPI method instead of reducing modified stability number. In addition, this approach might be combined with the implicit acceleration to result in a super convergent, highly efficient scheme with simultaneous application in ODEs and PDEs. This is left as a future development.

3.2.1 Application of DPI with implicit averaging

In Sec.(3.2) we discussed a family of implicit DPI sequence (eq.(3.8)) which is unconditionally stable for less than unity or sufficiently large CFL number. In this section

we consider an application of this sequence in solving ODEs and/or PDEs. Consider the general ODE/PDE form of *explicit* residual based eq.(1.11),

$$\begin{aligned} \mathbf{u}_{n+1} &= \mathbf{u}_0 + \Delta t[\mathbf{\Omega}]_{12}\,\mathbb{S}\,[\mathbf{\Omega}]_{21}\mathbf{R}_n + \mathcal{O}\left(\Delta t^p, \Delta {x_1}^{p_1}, \Delta {x_2}^{p_2}, \ldots\right) \\ \mathbf{R}_n &= f\left(\mathbf{u}_n, \frac{\partial}{\partial x_1}\left(\mathbf{u}_n\right), \frac{\partial^2}{\partial x_1^2}\left(\mathbf{u}_n\right), \ldots, \frac{\partial}{\partial x_2}\left(\mathbf{u}_n\right), \frac{\partial^2}{\partial x_2^2}\left(\mathbf{u}_n\right), \ldots, t\right), \end{aligned} \quad (3.11)$$

where the nested column vector residual $\mathbf{R}_n$ is initially in the x_1 arrangement[1] and we used the rearrangement operator $[\mathbf{\Omega}]_{21}$ defined in (2.5) to rearrange it in the time direction (or direction 2) when integration in time is taken and then the result of integration are taken back to the original x_1 direction using the inverse rearrangement operator $[\mathbf{\Omega}]_{12} = [\mathbf{\Omega}]_{21}^{-1}$. Since $[\mathbf{\Omega}]_{21} = [\mathbf{\Omega}]_{12} = \mathbb{I}$ for the case of ordinary differential equations, eq.(3.11) consistently changes to (1.11). We note that (3.11) is the nonlinear Picard iterations for general residuals $\mathbf{R}_n$ coming from the rhs of PDEs/ODEs. While formulation (3.11) is valid for both structured and unstructured grids, in the next section (Sec. 4.1), we will devise particular rearrangement operators suitable to handle PDEs in structured grids. However when it comes to algorithm development in general case, we always start from (3.11).
Following the weighted averaging method (eq.(3.8)), the first equation of (3.11) can be written in the following implicit form

$$\mathbf{u}_{n+1} = \mathbf{u}_0 + \Delta t[\mathbf{\Omega}]_{12}\,\mathbb{S}\,[\mathbf{\Omega}]_{21}\frac{a'\mathbf{R}_{n+1} + b'\mathbf{R}_n}{a'+b'} + \mathcal{O}\left(\Delta t^p, \Delta {x_1}^{p_1}, \Delta {x_2}^{p_2}, \ldots\right) \quad (3.12)$$

Note that we still need the residuals at iteration $n+1$. For conservation laws, we use the linearization concept of Briley & McDonald (1977) or Beam & Warming (1978) who adapt it to construct implicit scheme between two time steps t and $t+\Delta t$. We apply the same method to make implicit schemes between two iterations n and $n+1$. The residuals linearization is given as follows[2]

$$\mathbf{R}_{n+1} = \mathbf{R}_n + \left[\frac{\partial \mathbf{R}_n}{\partial u}\right]\left(\mathbf{u}_{n+1} - \mathbf{u}_n\right) \quad (3.13)$$

where $\left[\frac{\partial \mathbf{R}_n}{\partial u}\right]$ is a *diagonal matrix* containing the Jacobian of the residual vector for each point in each iteration n. *One of the important results of this paper is that since the linearization is done in the iteration space (not in temporal space) the total*

[1] node arrangement as shown in eq.(2.1)

[2] We should note that since $\mathbf{R} \equiv \mathbf{R}\left(u,t\right)$ (recall eq.(1.1)), then $\mathbf{R}_{n+1} = \mathbf{R}_n + \left[\frac{\partial \mathbf{R}_n}{\partial u}\right]\left(\mathbf{u}_{n+1} - \mathbf{u}_n\right) + \left[\frac{\partial \mathbf{R}_n}{\partial t}\right]\left(\mathbf{t}_{n+1} - \mathbf{t}_n\right)$. Since we don't have reconfigurable (variable) temporal grid in each iteration (refer to fig.(1.0.1), we fix node location r_i during all iterations), then $\left(\mathbf{t}_{n+1} - \mathbf{t}_n\right) = 0$ is generally zero and residual linearization (3.13) is correct.

order of accuracy of the scheme remains unchanged independent of linearization. Substituting (3.13) into (3.12) leads

$$\mathbf{u}_{n+1} = \mathbf{u}_0 + \Delta t [\mathbf{\Omega}]_{12} \mathbb{S} [\mathbf{\Omega}]_{21} \left(\mathbf{R}_n + \frac{a'}{a'+b'} \left[\frac{\partial \mathbf{R}_n}{\partial u} \right] (\mathbf{u}_{n+1} - \mathbf{u}_n) \right) + \mathcal{O}\left(\Delta t^p, \Delta {x_1}^{p_1}, \Delta {x_2}^{p_2}, \ldots\right) \tag{3.14}$$

or in the more convenient form below

$$\begin{aligned} \left(\mathbb{I} \; - \; \frac{a' \Delta t}{a'+b'} [\mathbf{\Omega}]_{12} \, \mathbb{S} \, [\mathbf{\Omega}]_{21} \left[\frac{\partial \mathbf{R}_n}{\partial u} \right] \right) \mathbf{u}_{n+1} = \qquad\qquad (3.15) \\ \mathbf{u}_0 \; + \; \Delta t [\mathbf{\Omega}]_{12} \, \mathbb{S} \, [\mathbf{\Omega}]_{21} \left(\mathbf{R}_n - \frac{a'}{a'+b'} \left[\frac{\partial \mathbf{R}_n}{\partial u} \right] \mathbf{u}_n \right) + \mathcal{O}\left(\Delta t^p, \Delta {x_1}^{p_1}, \Delta {x_2}^{p_2}, \ldots\right) \end{aligned}$$

which establishes the fundamental scheme of **nonlinear** implicit Discrete Picard Iterations which is arbitrary order accurate in time (depending on the order of accuracy of $\mathbb{S}$ which can be increase arbitrarily). A quick glimpse at eq.(3.15) says that the coefficient of $\mathbf{u}_{n+1}$ is a $n_S \times n_S$ matrix where n_S is the size of the integration operator $\mathbb{S}$. So this time, we should solve a system of linear equations in each iteration.

To study the stability and convergence characteristics of implicit DPI, we need to consider the linear case for residual $\mathbf{R}_n$ as follows,

$$\mathbf{R}_n = \left[\bar{\mathbf{\Gamma}} \right] \mathbf{u}_n \tag{3.16}$$

Where $\left[\bar{\mathbf{\Gamma}} \right]$ is defined in (2.22). We had noticed (3.16) before in Theorem (II) and the footnote therein. The important point is that $\left[\bar{\mathbf{\Gamma}} \right]$ is not diagonal in general because the discretization in space depends on the each nodes and surrounding nodes which are off-diagonals. So before stability and convergence analysis we diagonalized the semi-discrete form (3.11) by following the same approach as we did in Theorem II. That means the residual (3.16) can be diagonalized as below

$$\bar{\mathbf{R}}_n = \mathrm{diag}\left(\lambda_{\bar{\Gamma} i}\right) \bar{\mathbf{u}}_n \tag{3.17}$$

where $\bar{\mathbf{R}} = [\mathbf{L}]\,\mathbf{R}$ and $\bar{\mathbf{u}} = [\mathbf{L}]\,\mathbf{u}$ and $\left[\bar{\mathbf{\Gamma}} \right] = [\mathbf{L}]^{-1} \mathrm{diag}\left(\lambda_{\bar{\Gamma} i}\right) [\mathbf{L}]$ is the eigen-value decomposition of $\left[\bar{\mathbf{\Gamma}} \right]$. In this case the solution vector $\mathbf{u}$ in implicit DPI (3.15) is replaced with $\bar{\mathbf{u}}$. Equation (3.17) gives us the Jacobian immediately

$$\frac{\partial \bar{\mathbf{R}}_n}{\partial \bar{u}} = \mathrm{diag}\left(\lambda_{\bar{\Gamma} i}\right) \tag{3.18}$$

Substituting (3.17) and (3.18) into (3.15) and replacing $\mathbf{u}$ with eigen-vector transferred $\bar{\mathbf{u}}$ and omitting the order of accuracy[3]we will obtain

$$\begin{aligned}\left(\mathbb{I}-\frac{a'\Delta t}{a'+b'}[\mathbf{\Omega}]_{12}\,\mathbb{S}\,[\mathbf{\Omega}]_{21}\mathrm{diag}\,(\lambda_{\bar{\Gamma}i})\right)\bar{\mathbf{u}}_{n+1} &= \bar{\mathbf{u}}_0 \\ +\Delta t[\mathbf{\Omega}]_{12}\,\mathbb{S}\quad[\mathbf{\Omega}]_{21}&\left(\mathrm{diag}\,(\lambda_{\bar{\Gamma}i})\,\bar{\mathbf{u}}_n-\frac{a'}{a'+b'}\mathrm{diag}\,(\lambda_{\bar{\Gamma}i})\,\bar{\mathbf{u}}_n\right) \\ =\bar{\mathbf{u}}_0 \quad + \quad \Delta t[\mathbf{\Omega}]_{12}\,\mathbb{S}\,[\mathbf{\Omega}]_{21}&\left(\frac{b'}{a'+b'}\mathrm{diag}\,(\lambda_{\bar{\Gamma}i})\,\bar{\mathbf{u}}_n\right)\end{aligned} \tag{3.19}$$

or simply,

$$\bar{\mathbf{u}}_{n+1}-\frac{a'\Delta t}{a'+b'}\Big([\mathbf{\Omega}]_{12}\,\mathbb{S}\,[\mathbf{\Omega}]_{21}\mathrm{diag}\,(\lambda_{\bar{\Gamma}i})\Big)\bar{\mathbf{u}}_{n+1}=\bar{\mathbf{u}}_0+\frac{b'\Delta t}{a'+b'}\Big([\mathbf{\Omega}]_{12}\,\mathbb{S}\,[\mathbf{\Omega}]_{21}\mathrm{diag}\,(\lambda_{\bar{\Gamma}i})\Big)\bar{\mathbf{u}}_n \tag{3.20}$$

As discussed in Lemma (II), the rearrangement operator does not change the eigen-value of its operand. Thus exactly similar to derivation of eq.(2.42), the eigen-value of the total operator inside the big parenthesis, i.e.$[\mathbf{\Omega}]_{12}\,\mathbb{S}\,[\mathbf{\Omega}]_{21}\mathrm{diag}\,(\lambda_{\bar{\Gamma}i})$ in (3.20), can be written as

$$\mathrm{eig}\,([\mathbf{\Omega}]_{12}\,\mathbb{S}\,[\mathbf{\Omega}]_{21}\mathrm{diag}\,(\lambda_{\bar{\Gamma}i}))=[\mathbf{\Lambda}]\,\mathrm{diag}\,(\lambda_{\bar{\Gamma}i})=\mathrm{diag}\,(\lambda_i)\,\mathrm{diag}\,(\lambda_{\bar{\Gamma}i})=\mathrm{diag}\,(\lambda_i\lambda_{\bar{\Gamma}i}) \tag{3.21}$$

where λ_i is the ith eigen-value of the integration operator $\mathbb{S}$ as discussed in Theorem (I). Therefore if the left eigen-vector of $[\mathbf{\Omega}]_{12}\,\mathbb{S}\,[\mathbf{\Omega}]_{21}\mathrm{diag}\,(\lambda_{\bar{\Gamma}i})$ is $[\mathbf{W}]$ then the eigen-value decomposition of the total operator is given below (using (3.21))

$$[\mathbf{\Omega}]_{12}\,\mathbb{S}\,[\mathbf{\Omega}]_{21}\mathrm{diag}\,(\lambda_{\bar{\Gamma}i})=[\mathbf{W}]^{-1}\mathrm{diag}\,(\lambda_i\lambda_{\bar{\Gamma}i})\,[\mathbf{W}] \tag{3.22}$$

It is more convenient and meaningful if we rewrite $\lambda_{\bar{\Gamma}i}$ in (3.22) using the definition of generalized CFL number (eq.(2.15)). Thus

$$\lambda_{\bar{\Gamma}i}=\sum_{j=1}^{H}\frac{\frac{\partial f(\mathbf{x}_0)}{\partial\beta_j}}{\prod_{k=1}^{z}\Delta x_k{}^{\omega_{jk}}}\lambda_{\Gamma ji}=\frac{\mathrm{CFL}_i}{\Delta t} \tag{3.23}$$

Substituting (3.23) into (3.22), we obtain

$$[\mathbf{\Omega}]_{12}\,\mathbb{S}\,[\mathbf{\Omega}]_{21}\mathrm{diag}\,(\lambda_{\bar{\Gamma}i})=[\mathbf{W}]^{-1}\mathrm{diag}\left(\frac{\lambda_i\mathrm{CFL}_i}{\Delta t}\right)[\mathbf{W}] \tag{3.24}$$

[3]Since we showed in eq.(3.15) that the order of accuracy remains arbitrary independent of linearization, for the soul purpose of stability analysis we are free to omit it because it doesn't have any effect on stability/convergence as we discussed in Theorem I. Since it a constant value during Picard iterations, it can be counted together with the initial value $\bar{\mathbf{u}}_0$.

Replacing the total operator $[\mathbf{\Omega}]_{12}\,\mathbb{S}\,[\mathbf{\Omega}]_{21}\mathrm{diag}\,(\lambda_{\bar{\Gamma}i})$ in (3.20) with (3.24) we will have

$$\bar{\mathbf{u}}_{n+1}-\frac{a'\Delta t}{a'+b'}[\mathbf{W}]^{-1}\mathrm{diag}\left(\frac{\lambda_i \mathrm{CFL}_i}{\Delta t}\right)[\mathbf{W}]\,\bar{\mathbf{u}}_{n+1}=\bar{\mathbf{u}}_0+\frac{b'\Delta t}{a'+b'}[\mathbf{W}]^{-1}\mathrm{diag}\left(\frac{\lambda_i \mathrm{CFL}_i}{\Delta t}\right)[\mathbf{W}]\,\bar{\mathbf{u}}_n \tag{3.25}$$

Multiplying both sides of (3.25) with $[\mathbf{W}]$ we will have

$$[\mathbf{W}]\,\bar{\mathbf{u}}_{n+1}-\frac{a'\Delta t}{a'+b'}\mathrm{diag}\left(\frac{\lambda_i \mathrm{CFL}_i}{\Delta t}\right)[\mathbf{W}]\,\bar{\mathbf{u}}_{n+1}=[\mathbf{W}]\,\bar{\mathbf{u}}_0+\frac{b'\Delta t}{a'+b'}\mathrm{diag}\left(\frac{\lambda_i \mathrm{CFL}_i}{\Delta t}\right)[\mathbf{W}]\,\bar{\mathbf{u}}_n \tag{3.26}$$

We define the following eigen-vector transferred variable

$$\bar{\bar{\mathbf{u}}}=[\mathbf{W}]\,\bar{\mathbf{u}} \tag{3.27}$$

Substituting (3.27) into (3.26) we obtain the following decoupled (diagonalized) system of equations

$$\bar{\bar{\mathbf{u}}}_{n+1}-\frac{a'\Delta t}{a'+b'}\mathrm{diag}\left(\frac{\lambda_i \mathrm{CFL}_i}{\Delta t}\right)\bar{\bar{\mathbf{u}}}_{n+1}=\bar{\bar{\mathbf{u}}}_0+\frac{b'\Delta t}{a'+b'}\mathrm{diag}\left(\frac{\lambda_i \mathrm{CFL}_i}{\Delta t}\right)\bar{\bar{\mathbf{u}}}_n \tag{3.28}$$

or

$$\left(\mathbb{I}-\mathrm{diag}\left(\frac{a'\lambda_i \mathrm{CFL}_i}{a'+b'}\right)\right)\bar{\bar{\mathbf{u}}}_{n+1}=\bar{\bar{\mathbf{u}}}_0+\mathrm{diag}\left(\frac{b'\lambda_i \mathrm{CFL}_i}{a'+b'}\right)\bar{\bar{\mathbf{u}}}_n \tag{3.29}$$

Rearranging (3.29) leads us to

$$\bar{\bar{\mathbf{u}}}_{n+1}=\bar{\bar{\mathbf{u}}}_0+\mathrm{diag}\left(\frac{a'\lambda_i \mathrm{CFL}_i}{a'+b'}\right)\bar{\bar{\mathbf{u}}}_{n+1}+\mathrm{diag}\left(\frac{b'\lambda_i \mathrm{CFL}_i}{a'+b'}\right)\bar{\bar{\mathbf{u}}}_n \tag{3.30}$$

Evidently, for entry "i" of the diagonal system (3.30) we have

$$\bar{\bar{u}}_{i(n+1)}=\bar{\bar{u}}_{i0}+\lambda_i \mathrm{CFL}_i\frac{a'\bar{\bar{u}}_{i(n+1)}+b'\bar{\bar{u}}_{i(n)}}{a'+b'} \tag{3.31}$$

From Theorem (I) we remember that $r_i=\lambda_i c$ for linear ODE and more comprehensively, from Theorem (II) we recall that $r_i=\lambda_i \mathrm{CFL}_i=\lambda_i c/\Delta x$ is the stability number for linear PDE which consistently switches to ODE when there is no spatial discretization Δx. Therefore we replace $\lambda_i \mathrm{CFL}_i$ in (3.31) with stability number r_i leading to the final result for general implicit DPI scheme,

$$\bar{\bar{u}}_{i(n+1)}=\bar{\bar{u}}_{i0}+r_i\frac{a'\bar{\bar{u}}_{i(n+1)}+b'\bar{\bar{u}}_{i(n)}}{a'+b'} \tag{3.32}$$

Evidently this equation is exactly identical to the implicit DPI sequence (3.8) discussed in Sec.(3.2) for $a' = a$ and $b' = b$. Therefore the analysis presented in that section is valid for the implicit DPI scheme for ODEs and PDEs.
Summarizing the stability of implicit DPI (3.15) it can be said that the scheme is stable for $r = \lambda_{\max}\text{CFL}$ between 0 and 1 and between $(a' + b') / (a' - b')$ and infinity (please refer to fig.(3.2.1)). Therefore implicit scheme is unconditionally convergent in this region. However we should bear in mind that in practical implementation, we should use a condition to prevent the solver to use stability number between unity and $(a' + b') / (a' - b')$ where the implicit DPI is unconditionally divergent. Figure (ref) shows that this gap is very narrow for large value of a' which we use in practical implementation.
Summarizing the convergence of implicit DPI, for $0 \leq r \leq 1$, implicit DPI converges faster for arbitrary constants a' and b' than any explicit DPI. For $b' \to 1$ and $a' \to 0$ implicit DPI convergence is equal to explicit DPI while for $a' \to 1$ and $b' \to 0$ it converges at the very first iterations! For CFL $\to \infty$ or equivalently $r \to \infty$, implicit DPI converges to asymptote $-b'/a'$. So the rate of convergence can be adjusted by modifying this ratio. For $b' \to 0$ the implicit DPI leads to golden scheme which converges at the first iteration for CFL$\to \infty$. We also conclude that:
while implicit DPI (3.15) removes the stability limitation of explicit DPI (1.11), it also improves the convergence of the original DPI. This is while both schemes are arbitrary-order accurate in time and the order of accuracy can be adjusted by replacing the integration operator S with a high-order integrator in eq.(3.15) without reformulation of the scheme and changing the code.

3.3 Comparison of Implicit Picard and Newton Iterations

The Newton Iteration is probably the most popular method for solving a system of nonlinear equations and is easily extended to system of nonlinear PDEs and ODEs as wellBriley & McDonald (2001). In this section we derive the modified stability number of Newton iteration method and we compare the result with explicit and implicit DPI.
Let us consider the residual-based eq.(1.1) again where this time the derivative is discretized instead of the integral form of the equation. Substituting the time derivative with a first order forward differencing we will obtain,

$$\frac{\Delta u}{\Delta t} = \frac{u_{n+1} - u_n}{\Delta t} = R\left(u_n\right), \tag{3.33}$$

Which is the first-order explicit scheme. Assuming implicit relation we have

$$\frac{u_{n+1}-u_n}{\Delta t}=R\left(u_{n+1}\right), \tag{3.34}$$

Rewriting (3.34) in the form of

$$F=\frac{u_{n+1}-u_n}{\Delta t}-R\left(u_{n+1}\right)=0, \tag{3.35}$$

If we repeat (3.35) for all "n" in the time until a terminal value call it "n_S", we can combine the result into the following matrix form,

$$\mathbf{F}\left(u_1,u_2,\ldots,u_n\right)=\mathbf{F}\left(\mathbf{u}\right)=\begin{bmatrix} u_1-u_0 \\ \frac{u_2}{\Delta t}-\frac{u_1}{\Delta t}-R(u_2) \\ \frac{u_3}{\Delta t}-\frac{u_2}{\Delta t}-R(u_3) \\ \vdots \\ \frac{u_{nS}}{\Delta t}-\frac{u_{nS-1}}{\Delta t}-R(u_{nS}) \end{bmatrix}=0 \tag{3.36}$$

Now (3.36) is actually a system of nonlinear equations which can be solved using Newton iterations as follows. For nonlinear vector $\mathbf{F}\left(\mathbf{u}\right)=0$ in (3.36) we can write

$$\mathbf{F}\left(\mathbf{u}\right)=\mathbf{F}\left(\mathbf{u}_0\right)+\frac{\partial\mathbf{F}\left(\mathbf{u}\right)}{\partial\mathbf{u}}\Delta\mathbf{u}=0 \tag{3.37}$$

or in the iterative form

$$\mathbf{F}\left(\mathbf{u}_n\right)+\frac{\partial\mathbf{F}\left(\mathbf{u}\right)}{\partial\mathbf{u}_n}\left(\mathbf{u}_{n+1}-\mathbf{u}_n\right)=0 \tag{3.38}$$

which is solved for the next iteration

$$\mathbf{u}_{n+1}=\mathbf{u}_n-\frac{\partial\mathbf{F}\left(\mathbf{u}\right)}{\partial\mathbf{u}_n}^{-1}\mathbf{F}\left(\mathbf{u}_n\right) \tag{3.39}$$

Here we are only interested in the stability and convergence properties of (3.39). Therefore we consider the case where the residual in (3.33) is linear, say $R(u)=cu$ which comprehends all linear ODEs and PDEs. With this assumption, the target function $\mathbf{F}\left(\mathbf{u}\right)$ in (3.36) is written

$$\mathbf{F}\left(\mathbf{u}\right)=\begin{bmatrix} u_1-u_0 \\ \frac{u_2}{\Delta t}-\frac{u_1}{\Delta t}-cu_2 \\ \frac{u_3}{\Delta t}-\frac{u_2}{\Delta t}-cu_3 \\ \vdots \\ \frac{u_{nS}}{\Delta t}-\frac{u_{nS-1}}{\Delta t}-cu_{nS} \end{bmatrix}=\underbrace{\begin{bmatrix} \frac{-u_0}{\Delta t} & \frac{1}{\Delta t}-c & & & \\ & \frac{-1}{\Delta t} & \frac{1}{\Delta t}-c & & \\ & & & \ddots & \\ & & & \frac{-1}{\Delta t} & \frac{1}{\Delta t}-c \end{bmatrix}}_{\mathbf{\Psi}}\begin{bmatrix} u_2 \\ u_3 \\ \vdots \\ u_{nS} \end{bmatrix}=\mathbf{\Psi}\mathbf{u} \tag{3.40}$$

Substituting (3.40) into (3.39) we will have

$$\mathbf{u}_{n+1} = \mathbf{u}_n - \frac{\partial \mathbf{F}\left(\mathbf{u}\right)}{\partial \mathbf{u}_n}^{-1} \mathbf{\Psi} \mathbf{u}_n \tag{3.41}$$

And for the Jacobian in Newton root-finding equation (3.41) we can write (using (3.40)),

$$\frac{\partial \mathbf{F}\left(\mathbf{u}\right)}{\partial \mathbf{u}} = \mathbf{\Psi} \tag{3.42}$$

Substituting (3.42) into (3.41) we obtain

$$\mathbf{u}_{n+1} = \mathbf{u}_n - \mathbf{\Psi}^{-1}\mathbf{\Psi}\mathbf{u}_n = 0 \tag{3.43}$$

which reveals that the Newton iteration for a general system of **Linear** PDE/ODE converges at the first iteration. This is very interesting convergence property. From Sec. (3.2) we recall that implicit DPI also had the same convergence behavior when $a \to 1$ and $b \to 0$. So we might suspect that there should be a relation between Newton iteration (3.39) and implicit DPI proposed in (3.15).
To find a clue to the relation we start by reconsidering the explicit DPI given in (3.11). If we replace index 'n+1' with 'n' and considering implicit form (both side evaluated at 'n') we will have,

$$\mathbf{u}_n = \mathbf{u}_0 + \Delta t [\mathbf{\Omega}]_{12} \, \mathbb{S} \, [\mathbf{\Omega}]_{21} \mathbf{R}_n + \mathcal{O}\left(\Delta t^p, \Delta {x_1}^{p_1}, \Delta {x_2}^{p_2}, \ldots\right) \tag{3.44}$$

Instead of using Briley-McDonald linearization in the iteration space, this time we use Newton method. Constructing the target function $\mathbf{F}$ we obtain,

$$\mathbf{F}_n = \mathbf{u}_0 - \mathbf{u}_n + \Delta t [\mathbf{\Omega}]_{12} \, \mathbb{S} \, [\mathbf{\Omega}]_{21} \mathbf{R}_n + \mathcal{O}\left(\Delta t^p, \Delta {x_1}^{p_1}, \Delta {x_2}^{p_2}, \ldots\right) \tag{3.45}$$

which should be zero according to the Newton root-finding algorithm. Substituting into (3.39) we will obtain,

$$\mathbf{u}_{n+1} = \mathbf{u}_n - \frac{\mathbf{u}_0 - \mathbf{u}_n + \Delta t [\mathbf{\Omega}]_{12} \, \mathbb{S} \, [\mathbf{\Omega}]_{21} \mathbf{R}_n + \mathcal{O}\left(\Delta t^p, \Delta {x_1}^{p_1}, \Delta {x_2}^{p_2}, \ldots\right)}{-\mathbb{I} + \Delta t [\mathbf{\Omega}]_{12} \, \mathbb{S} \, [\mathbf{\Omega}]_{21} \left[\frac{\partial \mathbf{R}_n}{\partial u}\right]} \tag{3.46}$$

Multiplying both sides of (3.46) with $\left(\mathbb{I} - \Delta t [\mathbf{\Omega}]_{12} \, \mathbb{S} \, [\mathbf{\Omega}]_{21} \left[\frac{\partial \mathbf{R}_n}{\partial u}\right]\right)$ we will have

$$\begin{aligned}\left(\mathbb{I} - \Delta t [\mathbf{\Omega}]_{12} \, \mathbb{S} \, [\mathbf{\Omega}]_{21} \left[\frac{\partial \mathbf{R}_n}{\partial u}\right]\right) \mathbf{u}_{n+1} = \left(\mathbb{I} - \Delta t [\mathbf{\Omega}]_{12} \, \mathbb{S} \, [\mathbf{\Omega}]_{21} \left[\frac{\partial \mathbf{R}_n}{\partial u}\right]\right) \mathbf{u}_n \\ + \mathbf{u}_0 - \mathbf{u}_n + \Delta t [\mathbf{\Omega}]_{12} \, \mathbb{S} \, [\mathbf{\Omega}]_{21} \mathbf{R}_n + \mathcal{O}\left(\Delta t^p, \Delta {x_1}^{p_1}, \Delta {x_2}^{p_2}, \ldots\right)\end{aligned} \tag{3.47}$$

Simplifying the rhs of (3.47) we obtain,

$$\left(\mathbb{I} - \Delta t[\mathbf{\Omega}]_{12}\,\mathbb{S}\,[\mathbf{\Omega}]_{21}\left[\frac{\partial \mathbf{R}_n}{\partial u}\right]\right)\mathbf{u}_{n+1} = \mathbf{u}_0 + \tag{3.48}$$

$$\Delta t[\mathbf{\Omega}]_{12}\,\mathbb{S}\,[\mathbf{\Omega}]_{21}\left(\mathbf{R}_n - \left[\frac{\partial \mathbf{R}_n}{\partial u}\right]\mathbf{u}_n\right) + \mathcal{O}\left(\Delta t^p, \Delta {x_1}^{p_1}, \Delta {x_2}^{p_2}, \ldots\right)$$

which is a special case of (3.15) for $a = 1$ and $b = 0$. Therefore we can use Newton method to derive a special case of implicit DPI which was obtained before using linearization in the iteration space. One thing that we conclude here is that the convergence of implicit DPI is equivalent to Newton iteration for large coefficient a in (3.15). For small a the convergence degrades but we can move the instability gap to wherever we want to prevent instability which occurs in Newton's iteration algorithms.

3.4 Numerical Solution of ODEs using implicit DPI

The numerical characteristics of the implicit DPI scheme are tempting. They are summarized below

Property	**Proof**
arbitrary order accurate in time. [4]	eq.(3.15)
unconditionally stable at large CFL number	eq.(3.10) and fig.(3.2.1)
user-defined rate of convergence even at $CFL \to \infty$	proof in fig.(3.2.2) obtained from eq.(1.47) for the modified stability number given in eq.(3.10)
It is not linearized between two *time* steps so it captures all physical scales for arbitrary large CFL	eq.(3.13)

Table 3.1: The properties of Implicit Discrete Picard Iteration algorithm (3.15) for the linear case of general ODE-PDE.

Now we must perform implementation/evaluation procedures required to investigate the validity of the theory.

3.4.1 Time dependent Linear Scalar ODE

We start by the simplest ordinary differential equation,

$$\begin{aligned} \frac{d}{dt}u\left(t\right) &= cu - 2c + a_1 \sin\left(\frac{t}{a_2}\right), \\ u\left(0\right) &= u_0 \end{aligned} \tag{3.49}$$

For negative c, eq.(3.49) is excellent target for **stability** and **accuracy** analysis of numerical schemes because it has a converging value u_∞[5] so the equation is not physically unstable which is usually confused with the concept of numerical instability in numerical solutions and also it has a amplitude-frequency adjustable sinusoidal exciter which is suitable to increase the frequency spectrum and then asses the accuracy of the method. The linear ODE (3.49) has the following analytical solution,

$$u(t) = \mathrm{e}^{ct}\left(u_0 + \frac{-2\,c^2 {a_2}^2 - 2 + a_2\, a_1}{c^2 {a_2}^2 + 1}\right) - \frac{-2\,c^2 {a_2}^2 - 2 + a_1\, a_2\, \cos\left(\frac{t}{a_2}\right) + a_1\, c {a_2}^2 \sin\left(\frac{t}{a_2}\right)}{c^2 {a_2}^2 + 1} \tag{3.50}$$

Evidently for decaying solution $c < 0$ and no source term $a_1 = 0$, we achieve a steady state solution, i.e. $u(\infty) = 2$. For nonzero values of a_1, the final solution is a *stationary* sinusoidal profile.

Numerical solution of (3.49) presented in this section is done using $a_1 = 1.$, $a_2 = 10000$, $c = -1$ and $u_0 = 3$. For the implicit scheme (3.15), the stabilizer constants $a' = 0.8$ and $b' = 0.2$ are used. The convergence of DPI is controlled by defining a tolerance variable e_{DPI} which is *the final truncation error that we assume instead of the machine zero*. That means, for successive solutions $\mathbf{u}_{n+1}$ and $\mathbf{u}_n$ we define

$$\|\mathbf{u}_{n+1} - \mathbf{u}_n\| \leq e_{DPI}, \tag{3.51}$$

where $\mathbf{u}_{n+1}$ is obtained by solving (3.15) at each iteration. For efficiency considerations, we usually limit e_{DPI} to 10^{-6} instead of machine zero defined in eq.(1.46). Numerical experiments are done using DPI explicit (1.11), DPI implicit (3.15) and MATLAB ode45 (Shampine & Reichelt (1997)[6]) solvers.
In all experiments, the integration operator $\mathbb{S}$ is Newton-Cotes second-order accurate with the size $n_S = 700$. However we use two different values for the first entry of the operator (s_0 in eq.(1.6)). In the first set of experiments, we use $s_0 = 0$ and in the second set we use $s_0 = 0.5$. We will show that although $s_0 = 0$ condition exactly impose initial condition at $u(t_1) = u_0$, it cause nonlinear instabilities when implicit scheme are invoked for large CFL number. Even for the linear case it cause zigzag instabilities and major shift in solution. The choice $s_0 = 0.5$ is nonlinearly stable and remains stable for arbitrarily large CFL number and it doesn't lead to unstable/shifted solution. We will discuss this topic in detail in the following sections.

[5]The solution is convergent therefore the divergent behavior can be easily observed in the early iterations without special runtime requirements. As an example, the Euler explicit method is not stable for $c\Delta t = CFL > 2$ Quarteroni *et al.* (2000).

[6]The 2010 implementation.

The first set of experiments using $s_0 = 0$

We first use $s_0 = 0$ for the integration operator. As shown in (3.4.1), for small CFL numbers all solutions lead to the same result. For CFL = 1000 and higher the explicit DPI does not converge. As CFL increases, the ratio of wave-length per node increases. It is exciting to see that for CFL = 1000000, the implicit DPI has excellent agreement with ode45 while the former uses 700 points (4.34 sec.) and the latter uses 1205301 points (2.92e+02 sec.)! At CFL = 10000000, the difference between ode45 and implicit DPI becomes apparent especially the dissipation error is easily observable. This is mainly because condition $s_0 = 0$ that cause inconsistency. In fig.(3.4.2) we explored this feature in detail for CFL = 100000. As shown, the zigzag instability is very strong near the initial point $t = 0$. For larger t the zigzag wave vanishes but it displaces the solution significantly. This can be seen in the zoomed area on the right where blue circle lines are apparently displaced from the accurate RK mean value solution. It is also important to look at the solution obtained by ode45 showing that this RK scheme has microscopic oscillations everywhere in time.

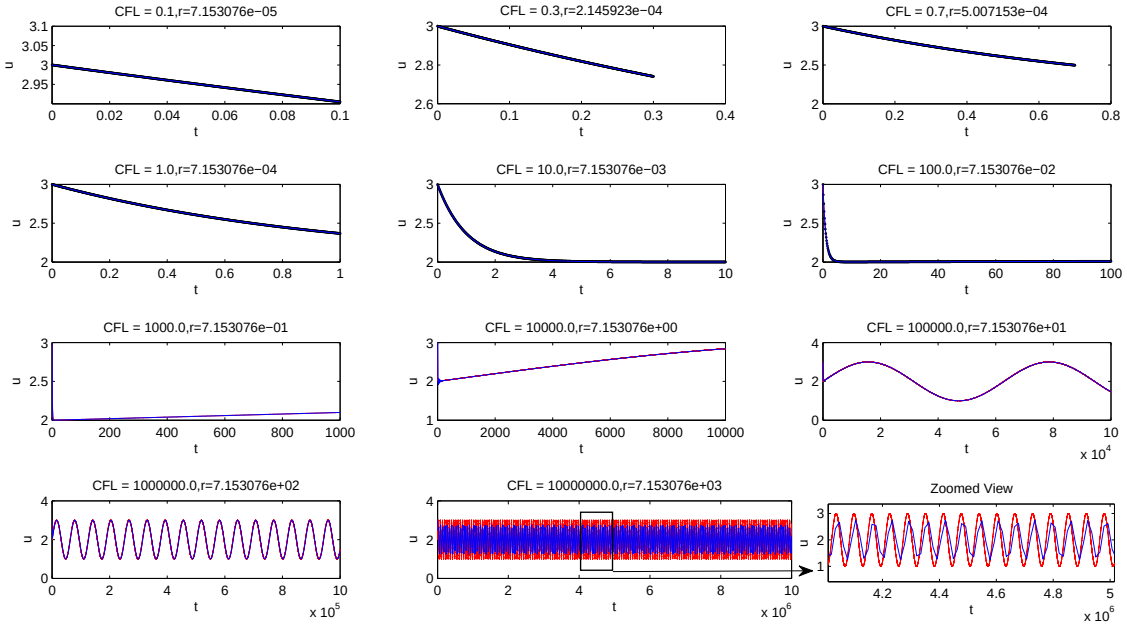


Figure 3.4.1: Case $s_0 = 0$: Numerical solution of linear eq.(3.50) using three refrence methods, Explicit DPI (black points), implicit DPI (blue line) and ode45 from Matlab ODE suite(red line). In this simulation, $a = 0.8$, $b = 0.2$, $e_{\text{DPI}} = 1.e-6$, $u_0 = 3$, $c = -1.0$, $a_1 = 1.0$, $a_2 = 10000$ and $n_S = 700$. The last solution (CFL = 10000000) is compared in fig.(3.4.3) for $s_0 = 0$ and $s_0 = 0.5$.

The zigzag instability problem is completely solved when we used $s_0 = 0.5$. As

shown in fig.(3.4.2), the solution obtained by $s_0 = 0.5$ (black quads) is accurate near $t = 0$ and it is also accurate at larger t (for example in the zoomed region near the extrema).

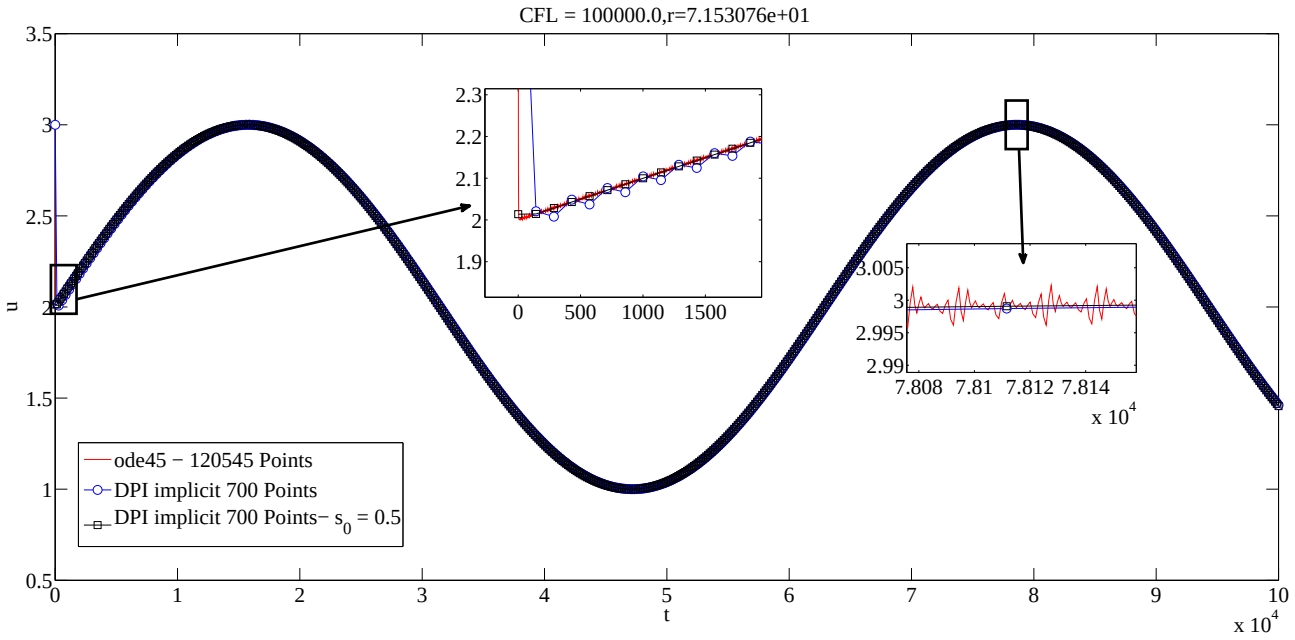


Figure 3.4.2: Zigzag Instabilities due $s_0 = 0$ near initial point t_1. The region near t_1 is twice zoomed to illustrate the instability. In addition, the operator condition $s_0 = 0$ cause a shift in the obtained solution compared to ode45 RK solution. As time increases, the zigzag instability vanishes. We have also zoomed an area of the solution in the middle of the figure. Note that ode45 Runge-Kutta algorithm exhibits a highly oscillatory behavior in this region. (bottom right).

The second experiment using $s_0 = 0.5$

In fig.(3.4.3) we did the last solution (CFL = 10,000,000) with both $s_0 = 0$ and $s_0 = 0.5$. As shown the condition $s_0 = 0.5$ completely resolves the problem and results almost have negligible dissipation/dispersion error (black quad lines). We also performed the numerical solution with a refined operator $n_S = 2000$. As shown, the refined solution (diamond lines) doesn't affect the accuracy.

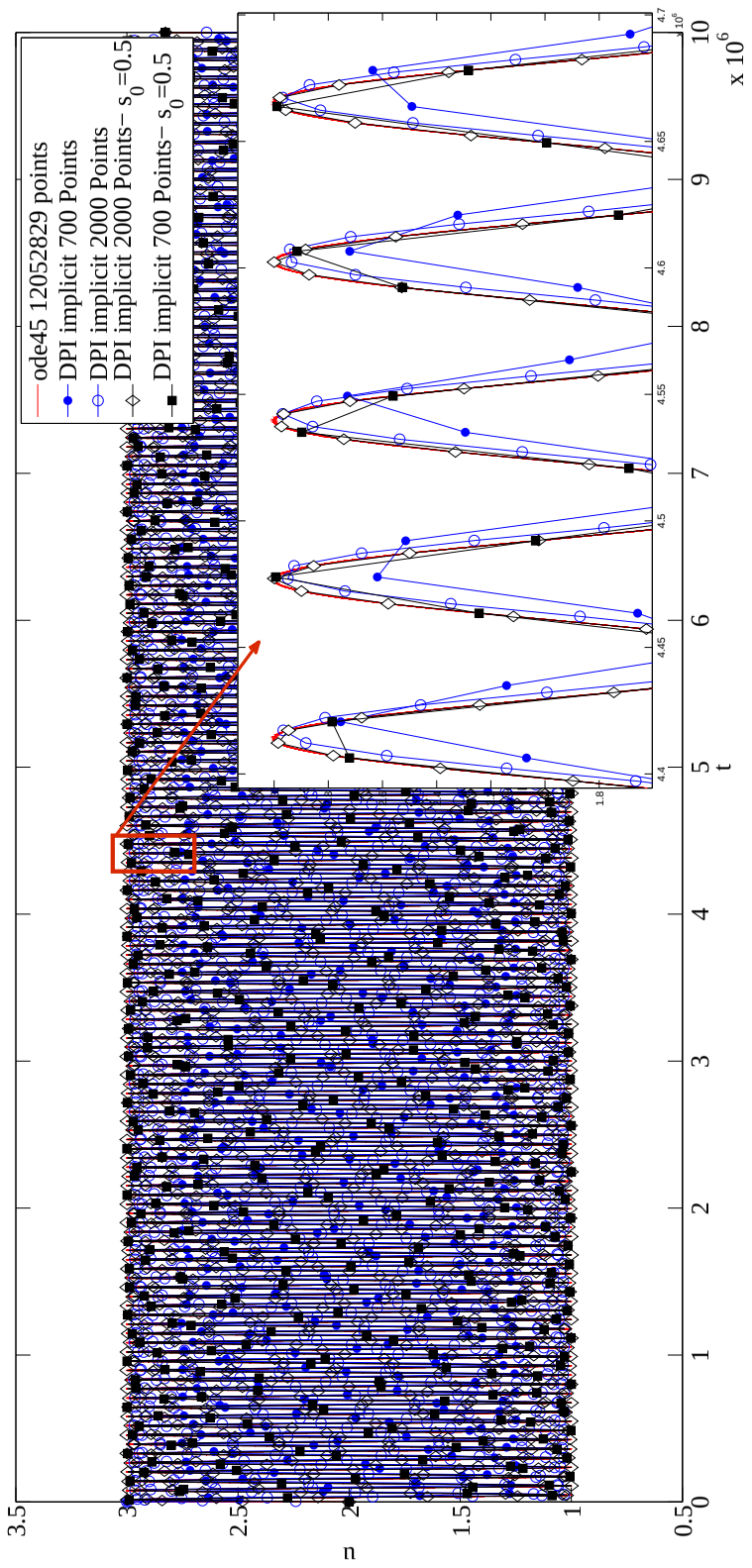


Figure 3.4.3: A comparison between $s_0 = 0$ and $s_0 = 0.5$ conditions for CFL = 10,000,000.

The performance analysis

To do a performance analysis, we derived the computation time using system clock and results are compared in Table(3.2). Please run the code () in appendix () to validate the result on your machine.[7]. The results are also visualized in fig.(3.4.4) for better understanding. As shown, for small CFL numbers, explicit DPI is more efficient than implicit one while ode45 is the most efficient scheme. The reason that explicit DPI is more efficient than implicit version is that there is no matrix inversion in the explicit algorithm. At CFL $\approx$ 13, the explicit and implicit DPI reaches to equality[8]. This is while ode45 is more efficient than both in this range of CFL number. At CFL $\approx$ 11500 we reach to an equality point for ode45 and Implicit DPI. [9] For large CFL numbers $CFL \to \infty$, the time required for implicit method to converges remains constant because the size of integrator operator is kept constant. From the computational performance point of view, at CFL = 10,000,000, the implicit DPI solution is obtained in 4.036114 Sec. while ode45 finish in 1.412809e+04 Sec. or 3.92 hours! This is a excellent result to show that the method is scalable in the sense of speed-accuracy trade-off. Also,for CFL = 10,000,000, implicit DPI is 3500.42 times more efficient than ode45 according to the Performance Ratio.

CFL	Explicit DPI	Implicit DPI	ode45	Performance Ratio[10]
0.1	1.643400e-01	1.360939e+00	3.821070e-01	2.81e-01
0.3	2.236190e-01	1.642657e+00	4.430200e-02	2.70e-02
0.7	2.922300e-01	1.902119e+00	7.128000e-03	3.75e-03
1	3.482880e-01	2.157056e+00	7.039000e-03	3.26e-03
10	1.222615e+00	3.510419e+00	7.373000e-03	2.10e-03
100	9.059893e+00	4.020598e+00	1.940200e-02	4.82e-03
1000	NA	4.037519e+00	1.643580e-01	4.07e-02
10,000	NA	4.024909e+00	1.652751e+00	4.11e-01
100,000	NA	4.055230e+00	1.779778e+01	4.39
1000,000	NA	4.335284e+00	2.922660e+02	67.42
10,000,000	NA	4.036114e+00	1.412809e+04	3500.42

Table 3.2: Timing obtained for different methods by MATLAB tic-toc timer. All values are in the Seconds. Run Program () in Appendix () in your machine to get timing in your machine.

[7]We note that since the timing is machine-dependent, we improvised the non-dimensional number *Performance Ratio* which is the ratio of elapsed time of ode45 to the elapsed time required for DPI implicit to solve the same CFL number. Thus the timing results will be different on different machines but the Performance Ratio (Table(3.2)) is universal meaning that it is almost the same on arbitrary hardware.

[8]This logically means the time required for Picard iterations with *higher* stability number r (more iterations) in explicit DPI is equal to the time required for matrix inversion and Picard iterations with *smaller* stability number (less iterations) in implicit DPI.

[9]The time required for matrix inversion and Picard Iterations in implicit DPI is equal to the computation time required for ode45 to march with very small time-steps due to stability.

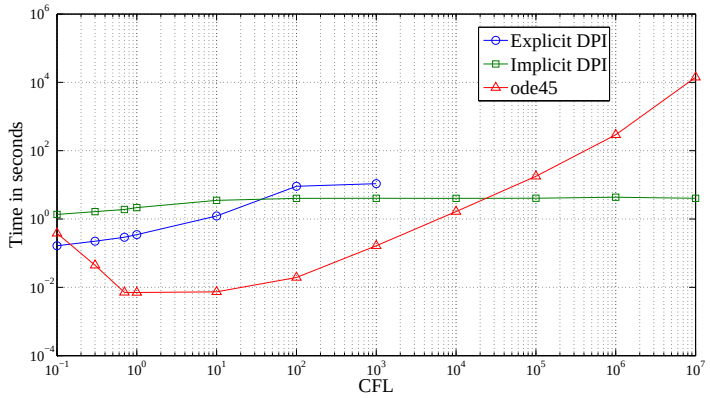


Figure 3.4.4: Visualization of timing presented in Table (3.2).

Conclusions and Observations from the linear case eq.(3.49)

The numerical solution presented in this section validates the second and the fourth statements of implicit DPI properties postulated in table (3.4). It shows that the implicit DPI remains unconditionally stable at large CFL. Also it proves that all physical scales are captured in CFL = 10,000,000 because the dispersion and dissipation errors are almost zero (see black quads in fig.(3.4.3)) through the entire time and there is no increasing attenuating (dissipation) over the time. This spectacular result is a giant motivation for application of the method in high-resolution simulations (like Turbulence and Aeroacoustics) because the numerical method is not dependent on grid scale (CFL number) nor it dependents on physical scales!!! This might be contradictory with classical numerical analysis at the first glance but let us consider the logical reason behind the curtain.

We start by considering the conceptual block diagram of general explicit time marching algorithms in fig.(3.4.5). This include Runge-Kutta schemes, Euler explicit, error-corrector schemes and etc. As shown the solution at the next step is computed merely based on the solution provided in the previous step leading to formation of *numerical amplification coefficient* "A" that changes the amplitude of the solution depending on the range of temporal integration and the order of accuracy of the scheme. This is practically unacceptable in Turbulence/Aeroacoustics as seen my many experiments [][][][][][][] because the high-frequency spectrum arising from a turbulent flow will eventually (as depicted in the figure) loose all spectral information and the final solution is not worthwhile of any physical correlation. For the implicit schemes, we have exactly the same block diagram except the arrows are replaced with double sided ones. In this case, we will have a **amplification (dissipation) error arising from linearization of residual between points**. Therefore, the

same analysis regarding the dissipation error applies to the implicit time-marching schemes as well.

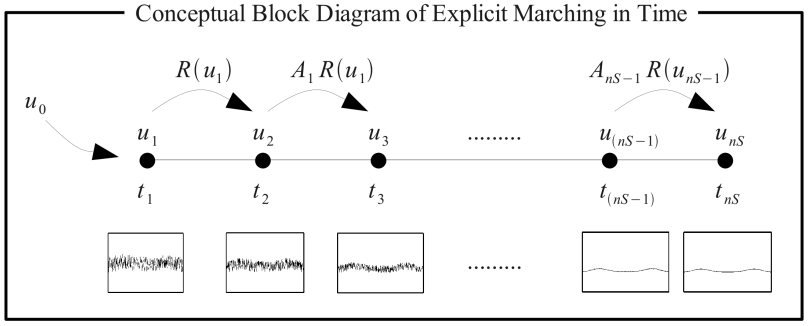


Figure 3.4.5: Conceptual block diagram of general family of explicit time-marching schemes. The corresponding block diagram for implicit time marching schemes is obtained by replacing the arrows with two ended arrows.

Now we pay attention to the block diagram of Picard iteration algorithm in fig(3.4.6). As shown, the arrows direction changes to the direction of the iteration space (vertical) where an initial rough solution is fed into the initial iteration at the bottom and it evolves until it eventually converges to the final solution at the top. As shown, this time another amplification factor "A'" appears because of the fact that solution at each *iteration* depends on the previous *iterations*. But we should note that according to the convergences theorem (I) and since the amplification appears in the iteration space (not time), "A'" converges to unity at the final iteration so it doesn't dissipate and degrade the final solution. This trend is graphically illustrated using small artistic plots of high-frequency signal that evolves from a very crude initial approximation (at the first iteration) to a error free rich spectrum solution at iteration N. Therefore between two time steps (see horizontal directions in the figure), there is *no linearization* and *amplification/dissipation factor*. We summarize these conclusions in the following postulate:

Postulate I: The non-dissipative property of implicit DPI schemes
For a time dependent PDE in the most general case (2.12), assume that

- The Jacobian of residual f is always negative, i.e., for $j = 1 \dots H$, we have $\frac{\partial f(\mathbf{x_0})}{\partial \beta_j} \leq \mathcal{M} < 0$[11] where $\mathcal{M}$ is a negative upper bound for all $\mathbf{x_0}$.[12]

[11] for definition of f please refer to Theorem II on page 31.

[12] This means that for the given ICs/BCs, (2.12) doesn't have limit cycle and all nonlinear effects vanish after a particular time scale determined by upper bound $\mathcal{M}$. Note that $\frac{\partial f(\mathbf{x_0})}{\partial \beta_j}$ with variable sign (positive and negative) creates a self-inductive chaotic source. Please refer to Van der Pol oscillator for more information.

- the residual $f(\mathbf{x})$ is calculated accurately so that the spatial discretization error is negligible.

Then the Implicit Discrete Picard Iteration defined in eq.(3.15) converges without amplification factor (dissipation error). In this case DPI is independent of both grid scale (CFL number) and physical scale (time step).

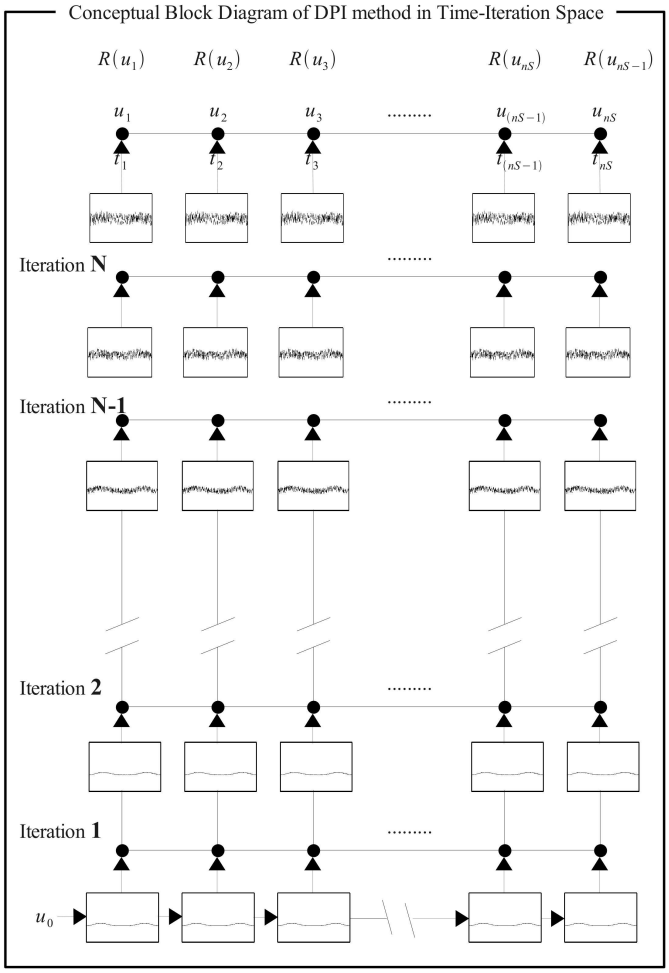


Figure 3.4.6: Conceptual block diagram of general family of DPI schemes. The corresponding block diagram for implicit DPI is obtained by replacing the arrows with two ended arrows.

Let us analyze the conditions given in postulate (I) in more details. It states that $\frac{\partial f(\mathbf{x_0})}{\partial \beta_j} \leq \mathcal{M} < 0$ because otherwise, the operator $\left[\bar{\mathbf{\Gamma}}\right]$ (defined in (2.22)) will have a **time varying** eigen-structure which some of the eigen-values may fall in the instability region $r = (1, (a+b)/(a-b))$ of DPI method. Furthermore, the convergence of DPI for nonlinear PDE (2.12) is proven in Theorem II (page (31)) using linearization of residual f over a small time-step Δt whereas for large time steps the DPI series may or may not converge in general. However if we assume that

f is nonlinear **but** $\frac{\partial f(\mathbf{x_0})}{\partial \beta_j} \leq \mathcal{M} < 0$, and $\mathcal{M} < 0$, then it is easy to show (using basic inequalities) that the partial summation used in DPI always converges for arbitrary time step.[13]

In addition, Postulate (I) requires that $f(\mathbf{x})$ or residuals must be calculated accurately because otherwise, from a conceptual point of view, the *speed* of waves are calculated incorrectly resulting in an initial shift in numerical solution and hence for large time steps, the numerical solution will have a major shift error[14] resulting from summation of all small shifts over the long period of time.

Postulate (I) is shared between the second and the fourth statement of table(3.4). To further validate this crucial postulate, we bring additional examples and numerical test cases in following sections.

3.4.2 Nonlinear ODEs

In this section, we consistently modify eq.(3.49) to make it an appropriate model to study nonlinear equations. If we replace constant c with $-u$ [15] we readily obtain

$$\begin{aligned} \frac{d}{dt}u(t) &= R(u,t) = -u^2 + 2u + a_1 \sin\left(\frac{t}{a_2}\right), \\ u(0) &= u_0 \end{aligned} \tag{3.52}$$

which is a nonlinear ordinary differential equation. The analytical Jacobian of (3.52) is simply given as

$$\frac{\partial R}{\partial u} = -2u + 2 \tag{3.53}$$

which is *negative* according to postulate (I) to make the system physically stable without any nonlinear effect as $t \to \infty$. Equation (3.52) is solved using the same set of constants used in Sec.(3.4.1) for linear eq.(3.49). In this case we use $s_0 = 0.5$ and "Δt" like previous test case is based on linear constant c as $\Delta t = CFL/|c|$. First to validate that Implicit DPI (the partial sum) converges for nonlinear case we choose an operator with size $n_S = 700$ and CFL = 100,000. The numerical solution is done suing ode45 and implicit DPI. As shown in fig.(3.4.7), the implicit DPI is in excellent agreement with ode45. It resolved the minimum point in the middle of the figure accurately. It is important to notice that the oscillatory behavior of ode45

[13] the proof is easy and is left to the reader.

[14] dispersion error

[15] This is done exactly analogous to linear and nonlinear Burgurs equation for PDEs.

exhibited in the previous section is again repeated here for nonlinear case. We have zoomed two regions on the left and right. In all cases the implicit DPI has excellent agreement with the ode45 method.

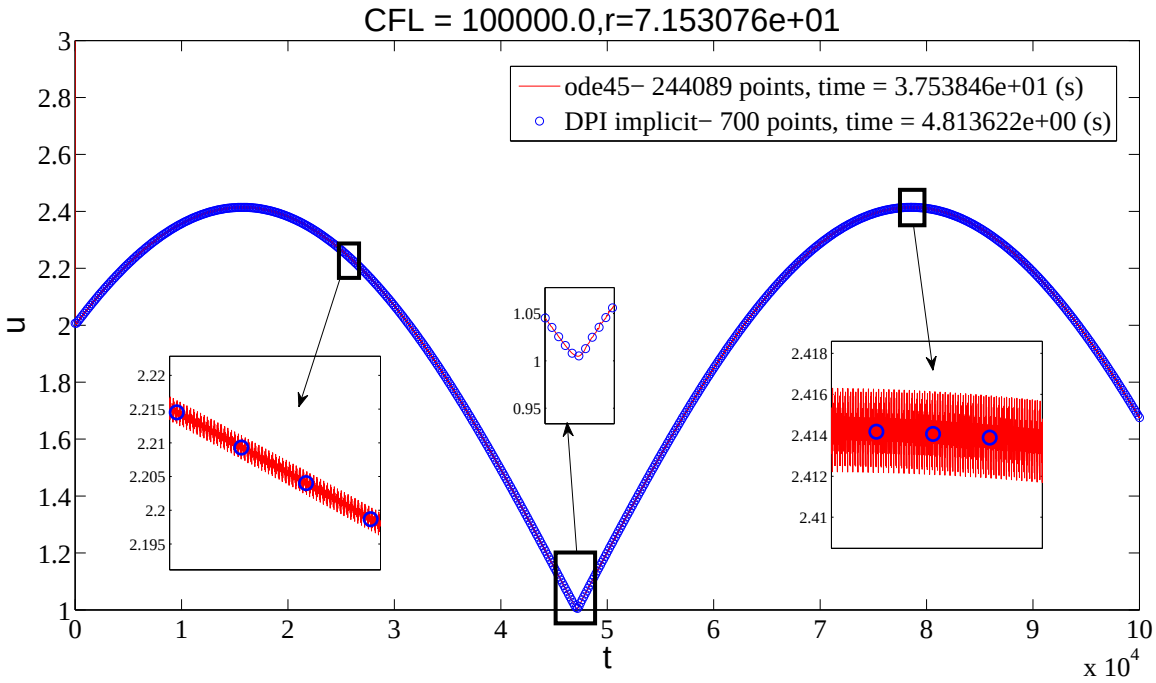


Figure 3.4.7: The comparison between ode45 and implicit DPI for nonlinear ODE (3.52). The integration operator size is $n_S = 700$.

To validate the conclusions and observations regarding the physical-scale independency of implicit DPI (given in Postulate I) for nonlinear case which was previously validated for linear ODE, here we perform a series of numerical solutions by selecting giant time step and reducing the number of points in time to show that the implicit DPI remain accurate regardless of the time step (physical scale).

The results are plotted in fig.(3.4.8). As shown, solution obtained only with four points in time (at the bottom) is exactly on the curve and is equal to the result obtained by 400 points in time (shown at the top). Interestingly the computation time of implicit DPI is 1.26e-2 (s) while the same result is obtained in 3.8e1 (s) using ode45.

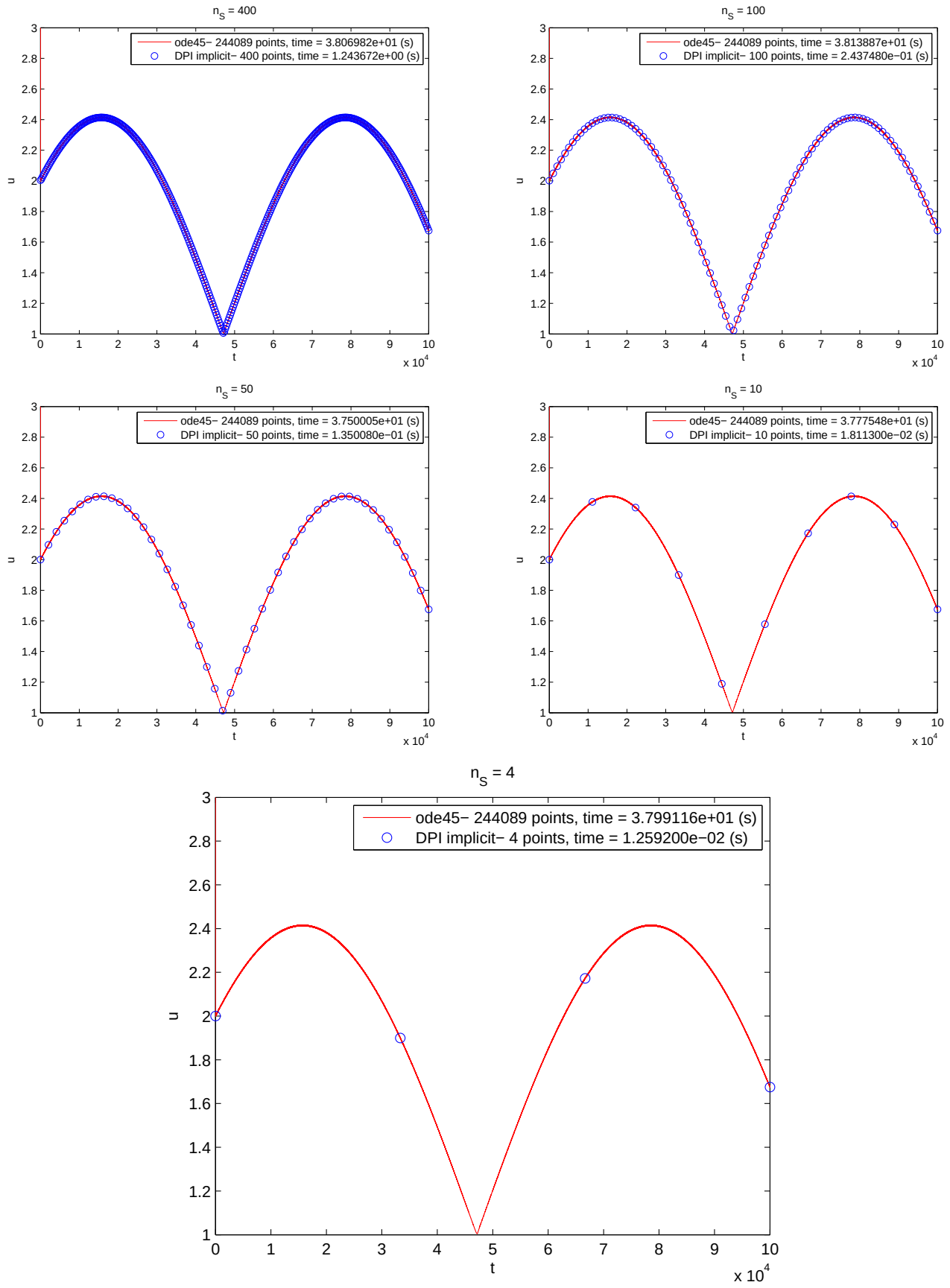


Figure 3.4.8: The accuracy of implicit DPI **for nonlinear case** when giant time steps are taken. The physical-scale independency of the algorithm is clearly visible.

3.4.3 Linear System of Ordinary Differential Equations

Practical applications usually involve solving system of differential equations that are stiff, high-frequency and strongly coupled. We are going to pose a system of ODEs which mimics the same properties. In this section we solve a linear system of ordinary differential equations to investigate the validity of Postulate (I) described in page 62 in the previous section for DPI method. Here we chose the following system

$$\begin{aligned}
\frac{d}{dt}u_1(t) &= -u_1(t) + 2u_2(t) + u_3(t) + a_1 \cos\left(\frac{t}{a_2}\right) \\
\frac{d}{dt}u_2(t) &= -3u_1(t) - 40000u_2(t) - u_3(t) + a_1 \sin\left(\frac{t}{a_2}\right) \\
\frac{d}{dt}u_3(t) &= -0.51u_1(t) - 4u_3(t) + 4a_1 \left|\sin\left(\frac{t}{a_2}\right)\right|
\end{aligned} \tag{3.54}$$

or in the space-state form below

$$\frac{d}{dt}\mathbf{u} = \mathbf{R}\left(t\right) = \mathbb{A}\mathbf{u} + \mathbf{b} \tag{3.55}$$

where

$$\mathbf{u} = \begin{bmatrix} u_1(t) \\ u_2(t) \\ u_3(t) \end{bmatrix},\ \mathbb{A} = \begin{bmatrix} -1 & 2 & 1 \\ -3 & -4000 & -1 \\ -0.51 & 0 & -4 \end{bmatrix},\ \mathbf{b} = a_1 \begin{bmatrix} \cos\left(\frac{t}{a_2}\right) \\ \sin\left(\frac{t}{a_2}\right) \\ 4\left|\sin\left(\frac{t}{a_2}\right)\right| \end{bmatrix} \tag{3.56}$$

and the Jacobian is easily obtained by differentiating residuals in (3.55) with respect to $\mathbf{u}$ as follows

$$\frac{d}{d\mathbf{u}}\mathbf{R}\left(t\right) = \mathbb{A} \tag{3.57}$$

Eigen-value decomposition of the coefficient matrix $\mathbb{A}$ shows that all eigen-values are negative, i.e.

$$\mathbf{\Lambda} = \begin{pmatrix} \text{-3999.99849956057} & 0 & 0 \\ 0 & \text{-1.18241605588794} & 0 \\ 0 & 0 & \text{-3.81908438355077} \end{pmatrix} \tag{3.58}$$

thus the system (3.55) is physically stable which means that after a *time constant* we can easily observe numerical instability (if exists). In addition, we should note that the magnitude of one of the eigen-values is considerably larger than others which means that the system of differential equations (3.54) is *stiff*. Also, the system of

equations (3.55) has an adjustable frequency wave-generator source term $\mathbf{b}$ which is an extension to the form presented in scalar ODE (3.49). Here we apply the implicit DPI of eq.(3.15) exactly similar to the form which was applied to scalar ODE (3.49) except all entries in integration operator $\mathbb{S}$, Jacobian matrix $\frac{\partial \mathbf{R}(t)}{\partial t}$, residual matrix $\mathbf{R}$ and solution vector $\mathbf{u}$ are replaced with block matrices of size 3×1 or 3×3. For example for integration matrix which was originally proposed in (1.6) for scalar case, if all entries are multiplied by sub-block identity matrices with the size equal to the number of equations (here 3×3) we will have

$$\mathbb{S} = \begin{bmatrix} s_0 \begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \end{bmatrix} & & & & \\ s_1 \begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \end{bmatrix} & s_2 \begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \end{bmatrix} & & & \\ s_1 \begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \end{bmatrix} & s_2 \begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \end{bmatrix} & s_3 \begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \end{bmatrix} & & \\ \vdots & \vdots & \vdots & \ddots & \\ s_1 \begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \end{bmatrix} & s_2 \begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \end{bmatrix} & s_3 \begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \end{bmatrix} & \cdots & s_{n_S} \begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \end{bmatrix} \end{bmatrix} \tag{3.59}$$

which is integration operator for system of linear ODEs (3.54) consistent with implicit DPI formulation (3.15). Similarly, for solution vector $\mathbf{u}$ we one can write

$$\mathbf{u} = \begin{bmatrix} u\,|_{t_1} \\ u\,|_{t_2} \\ \vdots \\ u\,|_{t_{nS}} \end{bmatrix} \rightarrow \mathbf{u} = \begin{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}_{t_1} \\ \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}_{t_2} \\ \vdots \\ \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}_{t_{nS}} \end{bmatrix} \tag{3.60}$$

where on the left is a solution vector containing the values of scalar u at various times while on the right, the solution vector contains the values of vector $\mathbf{u} = [u_1, u_2, u_3]^{\mathrm{T}}$

at various times. The scalar-to-vector transformation can be implemented easily in programming using sub-block substitution or using Kronecker matrix product as will be discussed in Chapter (4).

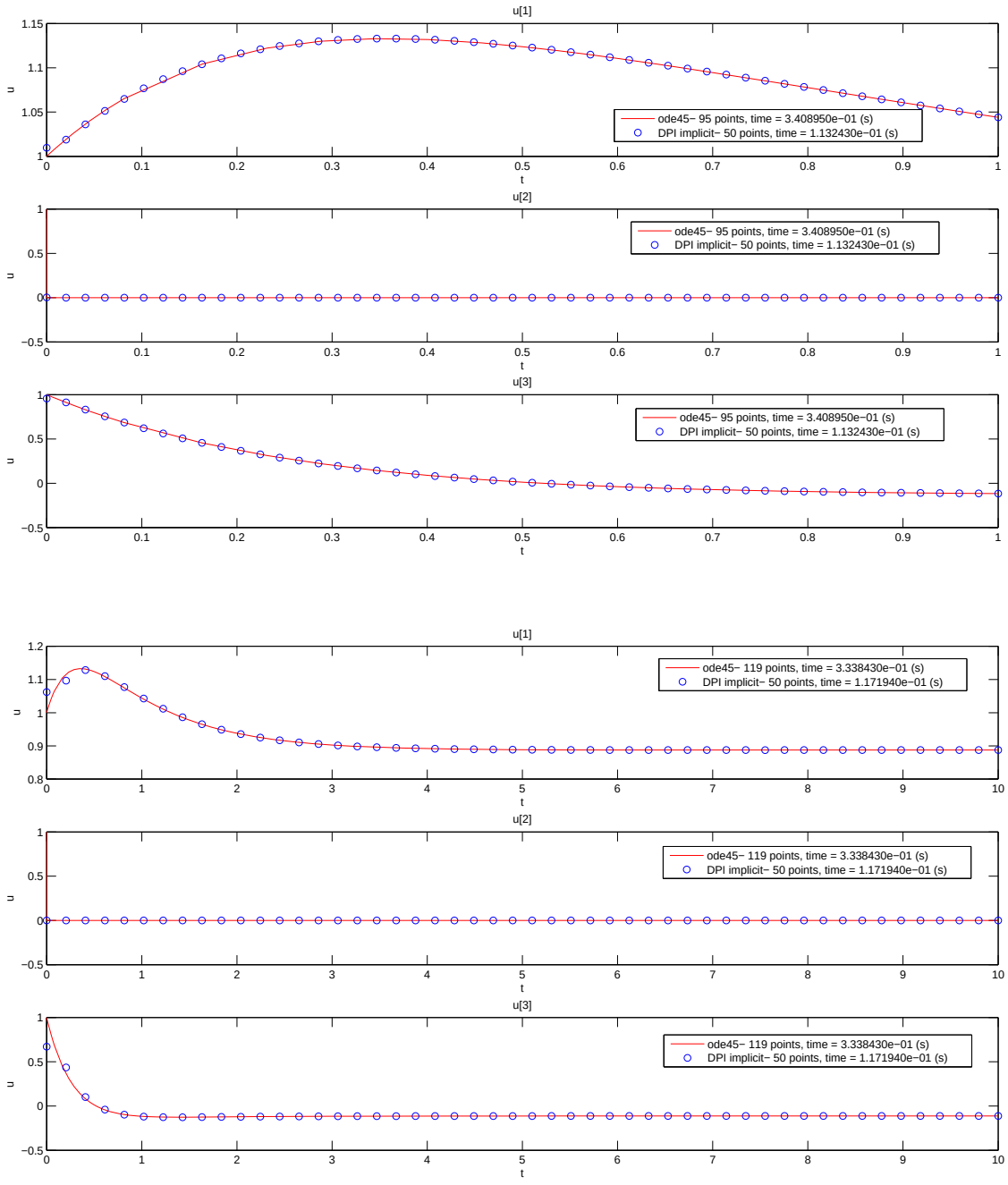


Figure 3.4.9: The accuracy of implicit DPI when giant time steps are taken for system of linear ODEs. The physical-scale independency of the algorithm is clearly visible. For a scalar ODE refer to fig.(3.4.8). (TOP Threes) u_1, u_2, u_3 corresponding to CFL $= 1$. (BOTTOM Threes) u_1, u_2, u_3 corresponding to CFL $= 10$.

Here we use numerical values $a' = 0.8$, $b' = 0.2$, $e_{DPI} = 1.e^{-6}$, $a_1 = 1.0$, $a_2 = 10000$ and $n_S = 50$. Hereafter we use $s_0 = 0.5$ unless it is stated explicitly. The results are shown in figs(3.4.9), (3.4.10) and (3.4.11) for a range of CFL numbers.

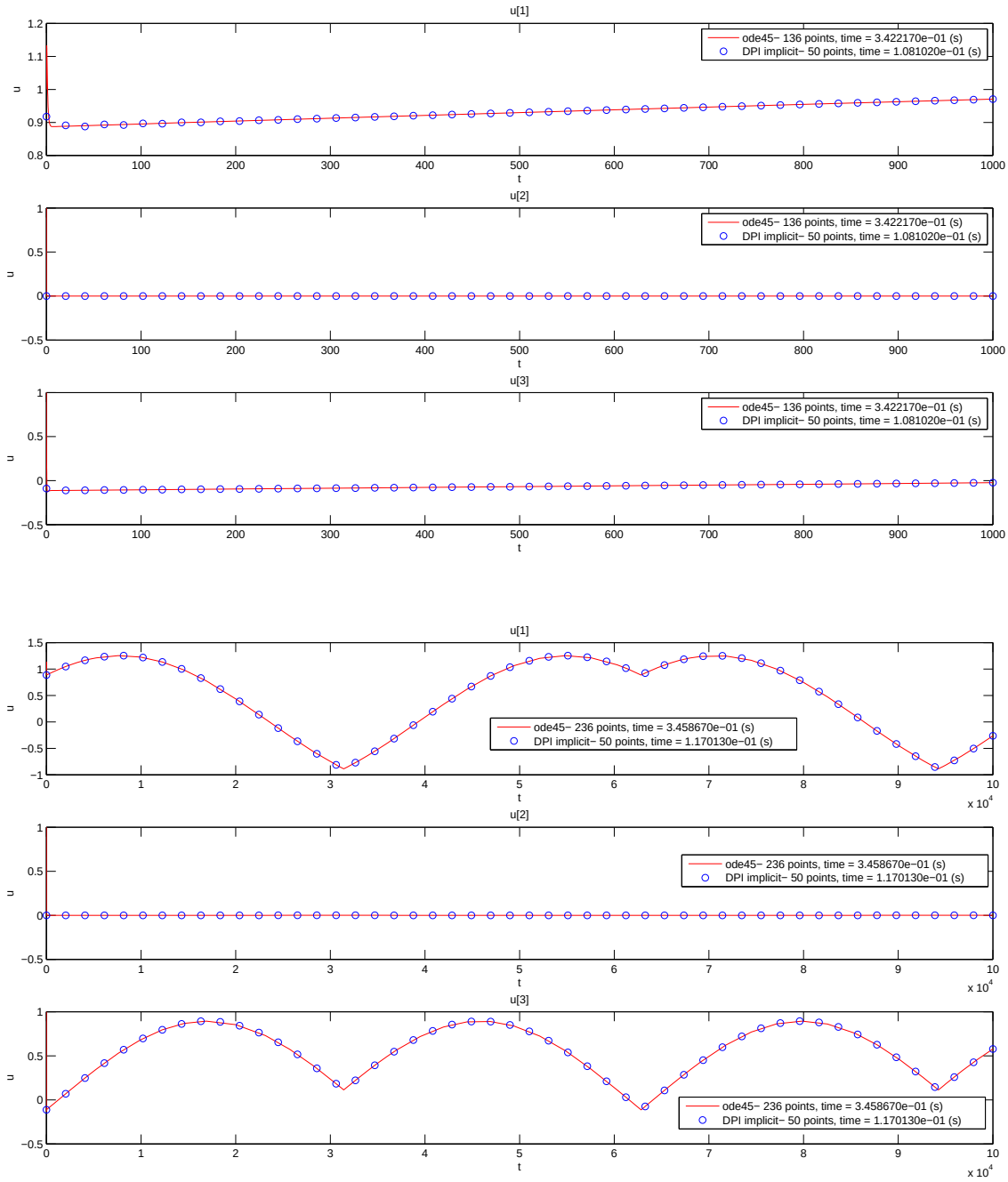


Figure 3.4.10: Continued from fig.(3.4.9). (TOP Threes) u_1, u_2, u_3 corresponding to CFL = 1000. (BOTTOM Threes) u_1, u_2, u_3 corresponding to CFL = 100000.

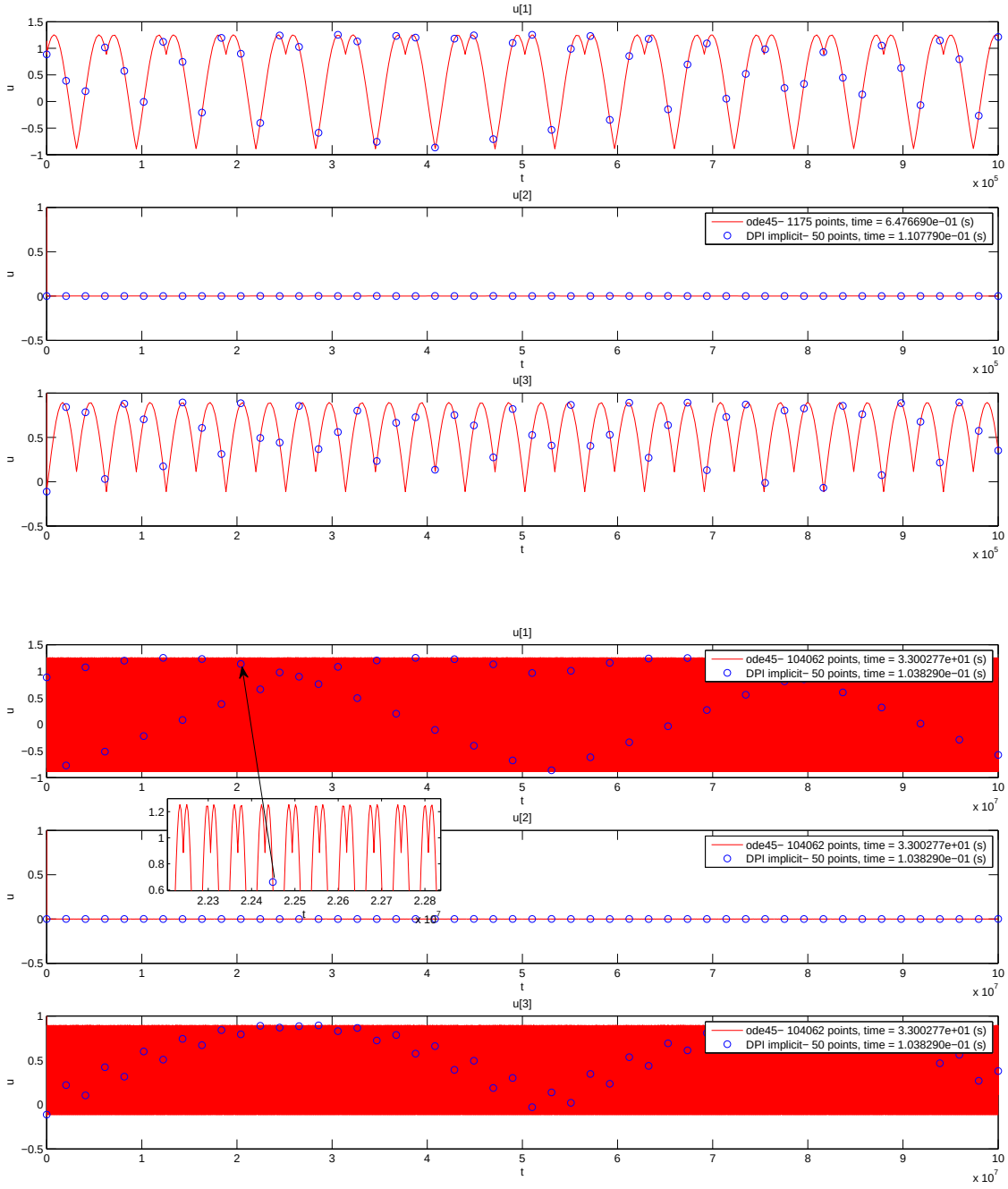


Figure 3.4.11: Continued from fig.(3.4.10). (TOP Threes) u_1, u_2, u_3 corresponding to CFL = 1000000. (BOTTOM Threes) u_1, u_2, u_3 corresponding to CFL = 10000000.

According to fig.(3.4.9), for CFL = 1 and CFL = 10, the implicit DPI is in good agreement with ode45 except near initial value which has a deviation from ode45. This was discussed earlier. For CFL = 1000 and CFL = 100000, (fig.(3.4.10)) we also reach to the same conclusion. As shown, sharp gradients are captured accurately near cusp points. In fig.(3.4.11) we again see interesting results. The solution is so

CFL	DPI implicit	ODE 45	Performance Ratio
10	1.17E-01	3.34E-01	2.85
1000	1.08E-01	3.42E-01	3.17
100000	1.17E-01	3.46E-01	2.96
1000000	1.11E-01	6.48E-01	5.85
10000000	1.04E-01	3.30E+01	317.86

Table 3.3: The performance analysis of implicit DPI when applied to linear system (3.54). Timing is obtained using Matlab tic-toc function which utilizes system clock.

high-frequency in the given time interval that we had to zoom particular regions to show the accuracy of DPI.

As shown, for CFL equal to ten millions, the DPI solution (blue circles) seem to be scattered. However, when zoomed enough, we see that they are exactly on the ode45 curve. This is surprising because according to Table (3.4.3), although ode45 algorithm leads to the same result for CFL = 10,000,000, it is 317.86 times more expensive than Implicit DPI.

Chapter 4

Applying implicit DPI to multi-dimensional PDEs in Structured Formulation

In chapter (3) we developed the implicit form of DPI (originally developed in Chapter (1) in the explicit form) where we applied it to ordinary differential equations. The results validated the superior efficiency and accuracy of this approach. Particularly, it was pointed out in Postulate (I) that (under some conditions that usually exist) the implicit DPI method removes dissipation error from time-dependent solution of differential equations making the numerical method independent of physical scales. In this chapter we proceed to apply the implicit DPI Iteration algorithm (3.15) to the system of PDEs in structured space-time in exactly similar way that we did before.

In Sec.(4.1) of this chapter, we first extend the rearrangement operator (introduced in Sec.(2.1)-eq.(2.4)) to multidimensional **structured discrete space-time**. For unstructured discrete space-time the original operator (2.4) is more appropriate and will be considered in depth in the next chapter. After extending the rearrangement operator, we will be able to consistently discretize any spatial operator in a structured multidimensional space using basic one-dimensional operators. For example a multi-dimensional Divergence operator will be easily constructed using one-dimensional differencing for first derivative and a couple of rearrangements as will be discussed in detail in Sec. (4.2). Then we attack a general conservative PDE in Sec.(4.3) using the spatial discretization obtained in Sec. (4.2) and implicit DPI formulation proposed in eq.(3.15). As a result a *numerical closed form solution* is obtained for general class of linear Partial Differential Equations with arbitrary initial/boundary conditions for the first time. This is actually a particular case of nonlinear implicit DPI solution when the Jacobians are constant. The ultimate generality obtained in this approach is tested for some special cases of multi-dimensional convection and results are compared with theoretical values.

4.1 Rearrangement operators for multidimensional structured discrete space-time

A multidimensional structured space-time is usually defined as a discrete space-time with *constant number of points* in all dimensions. For continuous vector field

$$\vec{u} = u\left(1 \ldots y, x_1, x_2, \ldots, x_{z-1}, t\right) \tag{4.1}$$

defined on a z-dimensional continuous space-time with 'y' number of vector components, we can always represent it in structured form

$$\mathbf{u} = u\left(1 \ldots y, 1 \ldots N_1, 1 \ldots N_2, \ldots, 1 \ldots N_z\right) \tag{4.2}$$

where the set of *constants* N_1,N_2,...,N_z are the number of points in each dimension and z is the number of space-time dimensions. Evidently this is a structured representation of a discrete space because it includes both values and connectivity map together. For example for a three dimensional space-time (two-dimensional space), the value of the third vector component at point (x=1,y=1,t=3) in space-time is $u_3\left(1,1,3\right) = u\left(3,1,1,3\right)$ and also we can extract all surrounding points without any additional information. This is while for unstructured discrete space, we represent spatial nodal information based on the node number which is a global value (as we did in eq.(2.1)) and we still need additional matrix maps for nodes connecting to a particular node in space.

Equation (4.2) is actually a multidimensional tensor which can be programmed easily using multi-dimensional array u[y][N1][N2]...[Nz]. Instead of doing that, we prefer to represent space-time tensor $\mathbf{u}$ in a *nested column format* to make sure that the formulation for multidimensional structured PDEs is perfectly consistent with eq.(3.15) where we used a column vector. Also, using nested column approach makes it easy to develop appropriate rearrangement operators which are vital tools in obtaining generic discrete operators like Divergence and Laplacian.

We represent the multidimensional tensor u[y][N1][N2]...[Nz] (eq.(4.2)) in a column such that inner dimensions are nested by outer dimensions successively. That means we first write the most inner dimension, i.e. u[1...y][1][1]...[1] in a vertical column. This is done while all other dimensions are freezed to 1. Then the same procedure is repeated for u[1...y][2][1]...[1] until the second dimension reaches to N1, i.e. u[1...y][N1][1]...[1]. We collect these columns one after another in vertical form. In this case it is easy to observe that the sub-columns formed by inner indices 1...y are nested by the big outer column formed by changing the outer indices 1...N1. This procedure is repeated until all indices are varied over their ranges. The result is a

giant single column containing all sub columns corresponding to inner dimensions. It should be note this giant column has the dimension of $(y \times N_1 \times N_2 \ldots \times N_z, 1)$.

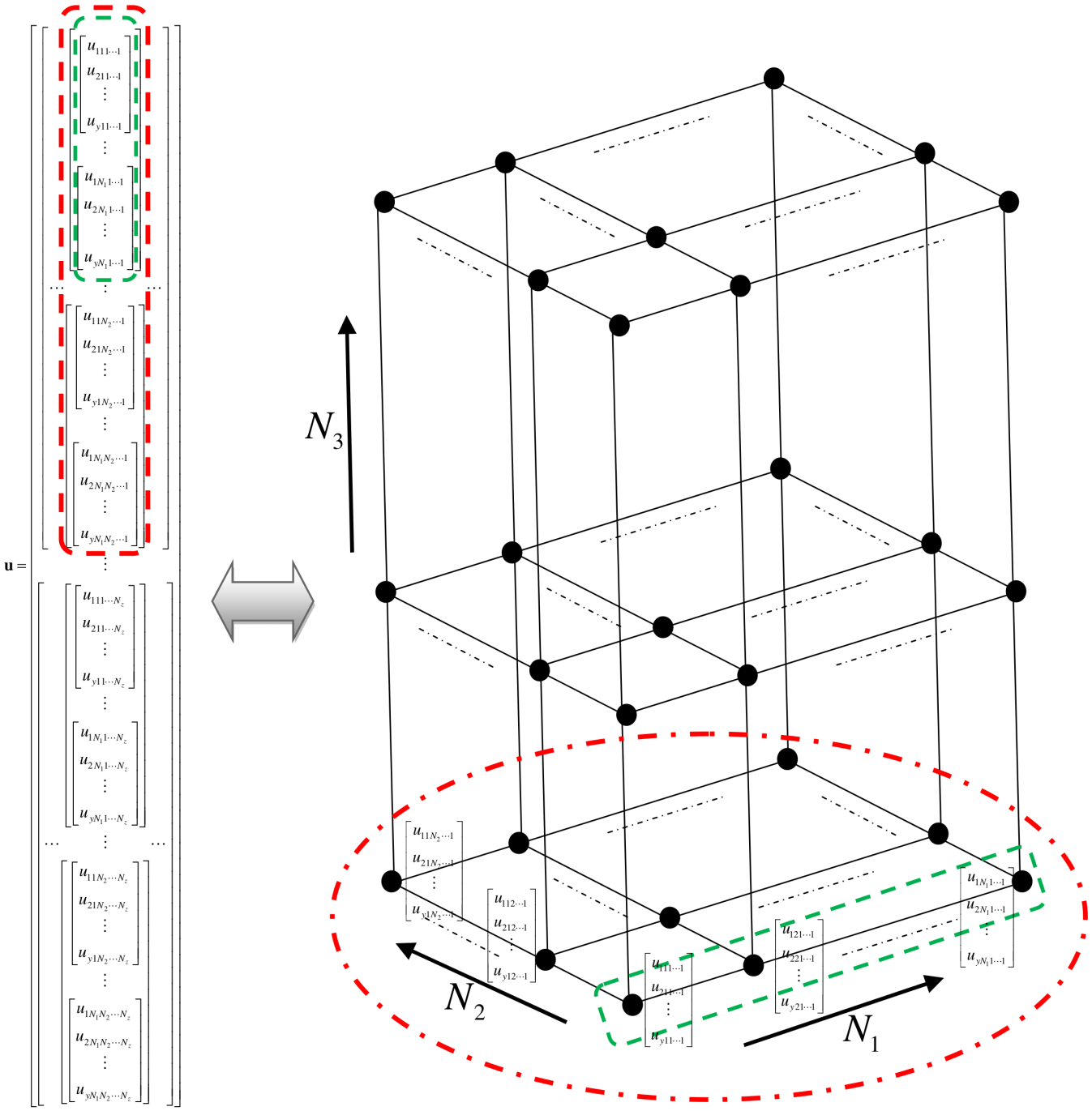


Figure 4.1.1: The graphical representation of the general vector field **u** over z-dimensional structured discrete space-time.

To facilitate the understanding of nested column format which might be confusing at first, we brought it in fig.(4.1.1) where on the left is the **data view** of z-dimensional

vector field **u** in nested column format and on the right is the **grid view** of the same concept. A shown, for the first point in the bottom corner, the value of vector field at that point say $[u_{111\ldots1}, u_{211\ldots1}, \ldots, u_{y11\ldots1}]^{\mathrm{T}}$ appears. This is the first sub-column on the top of data view on the left. Moving in the N1 direction[1], we obtain the value of the vector field at the second point, i.e. $[u_{121\ldots1}, u_{221\ldots1}, \ldots, u_{y21\ldots1}]^{\mathrm{T}}$. Note that the second index which belongs to N1 direction is incremented. We continue moving in N1 direction until we meet the final value $[u_{1N_11\ldots1}, u_{2N_11\ldots1}, \ldots, u_{yN_11\ldots1}]^{\mathrm{T}}$. The set of vectors (points) obtained so far form a one-dimensional discrete space (line) which is highlighted using dotted green lines on the grid view on the right. The corresponding column vector on data view on the left is also highlighted using the same line. For the next point we must take a new point $[u_{112\ldots1}, u_{212\ldots1}, \ldots, u_{y12\ldots1}]^{\mathrm{T}}$ in the N2 direction (as the start point) and repeat the same procedure in the N1 direction until we reach to the final value $[u_{1N_12\ldots1}, u_{2N_12\ldots1}, \ldots, u_{yN_12\ldots1}]^{\mathrm{T}}$. Actually, this is another one-dimensional discrete space (line). Doing so we collect all one-dimensional discrete spaces (lines) in $N_1 \times N_2$ dimension. This is graphically visualized using a dotted red line on the grid view (right) and corresponding column vector is also shown similarly on the data view (left). Thus the column enclosed in red line on the left is a ($N_1 \times N_2$, 1) column vector which represents a two-dimensional discrete space (plane) similar to two-dimensional matrix u[N1][N2] but arranged vertically. Continuing the same procedure we move over planes in vertical direction N3 to form volumes and we move over volumes to form hyper-volumes and so on. The general multidimensional space thus can be uniquely represented using a single column without any ambiguity.

We also note that we always moved in the N1 direction as the primary direction. This is the direction of initial arrangement by default. However, when doing differentiation or any linear operation that depends on direction, we should rearrange data in that direction using rearrangement operators. After operation, the data should be returned back to initial arrangement.

Please note that in Sec.(2.1) where we discussed unstructured discrete space-time as a basis for Theorem II, we only had two direction 1 and 2 where 1 belonged to space (nodes) and 2 belonged to the time direction. Therefore we only needed to find single rearrangement operator $\mathbf{\Omega}_{21}$ to rearrange data between these two directions when integration in time was performed. However, in this section we have direct control over all spatial and temporal directions. In particular, we are interested here to take derivative in all spatial-temporal directions simultaneously and sum-up the result to form numerical Divergence. Therefore we need a set of rearrangement

[1] which is the direction of principal axis x_1.

operators $\mathbf{\Omega}_{21}$, $\mathbf{\Omega}_{31}$, ..., $\mathbf{\Omega}_{z1}$ to rearrange data in all possible directions.

To find the the general rearrangement operator[2] $\mathbf{\Omega}_{x1}$ where $2 \leq x \leq z$ we go back and take a second look at eq.(2.4). If we replace ones with $\mathbb{I}_{y\times y}$ and $N_{nodes} = \prod_{i=1}^{x-1} N_i$ and $N_t = N_x$, we can easily find the rearrangement operator for rearranging direction 1 to direction x analogous to the one given in eq.(2.4). Thus if the direction x is the last dimension in the discrete structured space, i.e. x=z, then $\mathbf{\Omega}_{x1}$ is obtained using the mentioned transformation, otherwise for case $x < z$, we should put lower dimensional rearrangement operator with size $\prod_{i=1}^{x} N_i$ on the main diagonal of a $\prod_{i=x+1}^{z} N_i$ matrix to find the corresponding higher-dimensional operators. We implemented this concept using an efficient recursive algorithm in Listing (4.1). Since there are too many unnecessary zeros in rearrangement operator, we use sparse matrix storage.

Listing 4.1: The general rearrangement operator from initial direction 1 to direction $1 < x \leq z$.

```
%The following generates the [BLOCK] rearrangment operator in general case
function out = OMEGA_x1(N,y,x)
%N = containes an array of dimension of the space [N1 N2 ... Nz]
%so z = length(N);
%y = number of equations (variables) assigened to each node
%x = the desired dimension to be transferred

%Computing the number od dimensions of the space
z = length(N);
if ( x == z)
    prodz_1 = prod(N(1:(z-1)));
    prodz = prod(N(1:(z)));
    Omegax1 = sparse(1:prodz,1:prodz,0); %The final dims of the operator
    j = 0;
    k = 1;
    for i = 1:prodz
        Omegax1(i,j*prodz_1+k) = 1;
        j = j + 1;
        if (rem(i,N(z)) == 0)
            j = 0;
            k = k + 1;
        end
    end
    out = Omegax1; %returning the operator

else
    prodx1_z = prod(N((x+1):z));
    Omegax1 = cell(prodx1_z,prodx1_z);
    prodx = prod(N(1:x));
    Omegax1(:,:) = {sparse(zeros(prodx))};
    diag_Omegax1 = OMEGA_x1(N(1:x),y,x); %recursion
    for i = 1:prodx1_z
        Omegax1(i,i) = {diag_Omegax1}; %putting on main-diag
    end
```

[2]for x=1 we have $\mathbf{\Omega}_{11} = \mathbb{I}$ which means that direction 1 is already rearranged in its direction.

```
35      out = cell2mat(Omegax1); %returning the operator
36  end
```

4.1.1 Sample Application: Multidimensional Differentiation

In page (28) we posed the question 'How can we compute cross derivative $\frac{\partial}{\partial t}u(x,t)$ using matrix multiplication $\mathbb{D}_t\mathbf{u}$?'. It was explained in eq.(2.3) in that section that the multidimensional differentiation matrix in the '$x = x_1$' direction, i.e. $\mathbb{D}_x$ is made of blocks each one corresponding to one-dimensional differentiation matrix. Say

$$\begin{aligned}\mathbb{D}_{x_1} &= \begin{bmatrix} \frac{1}{\Delta x_1}[D]_{N_1\times N_1} & & & \\ & \frac{1}{\Delta x_1}[D]_{N_1\times N_1} & & \\ & & \ddots & \\ & & & \frac{1}{\Delta x_1}[D]_{N_1\times N_1} \end{bmatrix} \\ &= \frac{1}{\Delta x_1}\begin{bmatrix} [D] & & & \\ & [D] & & \\ & & \ddots & \\ & & & [D] \end{bmatrix}\end{aligned} \tag{4.3}$$

When multiplied by discrete space $\mathbf{u}$ which is originally in the x_1 direction, it gives

$$\begin{aligned}\mathbb{D}_{x_1}\mathbf{u} &= \begin{bmatrix} \frac{1}{\Delta x_1}[D]_{N_1\times N_1} & & & \\ & \frac{1}{\Delta x_1}[D]_{N_1\times N_1} & & \\ & & \ddots & \\ & & & \frac{1}{\Delta x_1}[D]_{N_1\times N_1} \end{bmatrix}\begin{bmatrix} [u]_{N_1\times 1} \\ [u]_{N_1\times 1} \\ \vdots \\ [u]_{N_1\times 1} \end{bmatrix} \\ &= \frac{1}{\Delta x_1}\begin{bmatrix} [D]_{N_1\times N_1}[u]_{N_1\times 1} \\ [D]_{N_1\times N_1}[u]_{N_1\times 1} \\ \vdots \\ [D]_{N_1\times N_1}[u]_{N_1\times 1} \end{bmatrix}\end{aligned} \tag{4.4}$$

which is the derivative of entire field in the x_1 direction. However for direction[3] $t = x_2$, we need to first rearrange the discrete space into the x_2 direction, then take the derivative using differentiation operator, say $\mathbb{D}_t\mathbf{u}$ and then transfer the discrete space back into the original arrangement using the inverse of rearrangement operator

[3]We often use the subscript 't' instead of x_z for the last dimension which accounts for time.

$\mathbf{\Omega}_{12} = {\mathbf{\Omega}_{21}}^{-1}$. In other words,

$$\frac{\partial}{\partial t}u(x,t) = \mathbf{\Omega}_{12}\left(\mathbb{D}_t\mathbf{\Omega}_{21}\mathbf{u} + \mathbf{b}_2 + \mathcal{O}\left(\Delta t^p\right)\right), \tag{4.5}$$

where $\mathbb{D}_t$ is formed similar to (4.4) except N_1 is replaced with the number of points in time, i.e. N_2 and grid spacing Δx_1 should also be replaced with Δt. Inside the parenthesis in eq.(4.5), the solution column $\mathbf{u}$ is first rearranged in the time direction '2' using operator $\mathbf{\Omega}_{21}$ and then the numerical differentiation is performed using $\mathbb{D}_t$. In doing numerical differentiation, a vector containing *boundary conditions* appears (Lomax *et al.* (2004)) for the second direction (time direction) which we showed it using $\mathbf{b}_2$. Also a truncation error adds to the terms inside parenthesis to complete numerical replacement of analytical differentiation. The final result is rearranged back to initial arrangement (direction 1) using $\mathbf{\Omega}_{12} = {\mathbf{\Omega}_{21}}^{-1}$. Equation (4.5) is a compact representation of analytical-to-discrete transformation of differentiation which has the advantage that if we want to increase the order of accuracy, we don't need to derive a new formula. All we need is to replace $\mathbb{D}_t$ with a high-order operator without changing a single line of the code. Also if we want to change boundary values we only change *one-dimensional* vector $\mathbf{b}_2$ locally and the rearrangement operator automatically will take care of everything in *multidimensional space*. Note that for periodic boundary conditions, the vector $\mathbf{b}$ is zero and the assumption of periodicity is directly implemented in the differentiation matrix $[\mathbf{D}]$. For example for non-periodic upwind scheme for the jth block $[\mathbf{D}]$ on the main diagonal of multidimensional differentiation matrix $\mathbb{D}_{x_1}$ we have,

$$\frac{1}{\Delta x_1}[\mathbf{D}]_{N_1\times N_1} = \frac{1}{\Delta x_1}\begin{bmatrix} 1 & & & & & & \\ -1 & 1 & & & & & \\ & -1 & 1 & & & & \\ & & \ddots & \ddots & & & \\ & & & -1 & 1 & & \\ & & & & -1 & 1 \end{bmatrix} \tag{4.6}$$

and for jth sub-block boundary condition vector $[\mathbf{b}]_j$ we have,

$$[\mathbf{b}_j] = \frac{1}{\Delta x_1}\begin{bmatrix} -bc_j \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \tag{4.7}$$

which should be imposed for all dimensions except the first dimension, i.e. j$= 2 \ldots N_z$. Assembling the one-dimensional sub-blocks (4.6) and (4.7) in multidimensional form (4.4), we obtain

$$\frac{\partial}{\partial x_1} u\left(x_1, x_2, \ldots, x_z\right) = \mathbb{D}_{x_1}\mathbf{u} + \mathbf{b}_1 + \mathcal{O}\left(\Delta x_1^p 1\right) \tag{4.8}$$

Where $\mathbb{D}_{x_1}$ is multidimensional differentiation matrix in x_1 direction obtained by assembling one-dimensional differentiation matrix (4.6) in the sub-block form presented in (4.4). The multidimensional boundary condition $\mathbf{b}_1$ is also assembled in similar way using one-dimensional form (4.7) as presented below,

$$\mathbf{b}_1 = \begin{bmatrix} [\mathbf{b}_1] \\ [\mathbf{b}_2] \\ \vdots \\ [\mathbf{b}_{N_2 \times \cdots N_z}] \end{bmatrix} \tag{4.9}$$

Instead of non-periodic 1D form in (4.6), for *periodic* upwind we can write,

$$\frac{1}{\Delta x_1}[\mathbf{D}]_{N_1 \times N_1} = \frac{1}{\Delta x_1} \begin{bmatrix} 1 & & & & & -1 \\ -1 & 1 & & & & \\ & -1 & 1 & & & \\ & & \ddots & \ddots & & \\ & & & -1 & 1 & \\ & & & & -1 & 1 \end{bmatrix}, \quad \mathbf{b}_1 = 0, \tag{4.10}$$

similarly for second-order central differentiation we can write,

$$\frac{1}{\Delta x_1}[\mathbf{D}]_{N_1 \times N_1} = \frac{1}{\Delta x_1} \begin{bmatrix} 0 & \frac{1}{2} & & & -\frac{1}{2} \\ -\frac{1}{2} & 0 & \ddots & & \\ & & \ddots & & \\ & & \ddots & 0 & \frac{1}{2} \\ \frac{1}{2} & & & -\frac{1}{2} & 0 \end{bmatrix}, \tag{4.11}$$

or for fourth-order central scheme we have (from Trefethen (2001)),

$$\frac{1}{\Delta x_1}[\mathbf{D}]_{N_1\times N_1} = \frac{1}{\Delta x_1}\begin{bmatrix} & & \ddots & & & \frac{1}{12} & -\frac{2}{3} \\ & & \ddots & -\frac{1}{12} & & & \frac{1}{12} \\ & & \ddots & \frac{2}{3} & \ddots & & \\ & & \ddots & 0 & \ddots & & \\ & & \ddots & -\frac{2}{3} & \ddots & & \\ -\frac{1}{12} & & & \frac{1}{12} & \ddots & & \\ \frac{2}{3} & -\frac{1}{12} & & & \ddots & & \end{bmatrix}, \tag{4.12}$$

For periodic boundaries we are interested in spectral methods because of their astonishing accuracy and rate of convergence. In particular, we enjoy the following spectral collocation,

$$\frac{1}{\Delta x_1}[\mathbf{D}]_{N_1\times N_1} = \begin{bmatrix} & \vdots & \\ \ddots & \frac{1}{2}\cot\frac{3\Delta x_1}{2} & \\ \ddots & -\frac{1}{2}\cot\frac{2\Delta x_1}{2} & \\ \ddots & \frac{1}{2}\cot\frac{1\Delta x_1}{2} & \\ & 0 & \\ & -\frac{1}{2}\cot\frac{1\Delta x_1}{2} & \ddots \\ & \frac{1}{2}\cot\frac{2\Delta x_1}{2} & \ddots \\ & -\frac{1}{2}\cot\frac{3\Delta x_1}{2} & \ddots \\ & \vdots & \end{bmatrix}, \tag{4.13}$$

Note that we couldn't bring the grid spacing out of the differentiation matrix. The advantage of using eq.(4.13) in practice is shown in fig.(4.1.2). We chose a dummy function $u = \exp(\sin(x))$ which has analytical derivative $u = \cos(x)\exp(\sin(x))$. We evaluated the numerical derivative using fourth-order matrix (4.12) and spectral differentiation matrix (4.13). Please note that to reach to the machine zero accuracy, fourth-order algorithm needs as least 10,000 points while the same result is obtained using spectral algorithm with only 20 points! This spectacular result clearly demonstrates superiority of spectral algorithms over conventional high-order algorithms. For more detail please refer to Trefethen (2001).

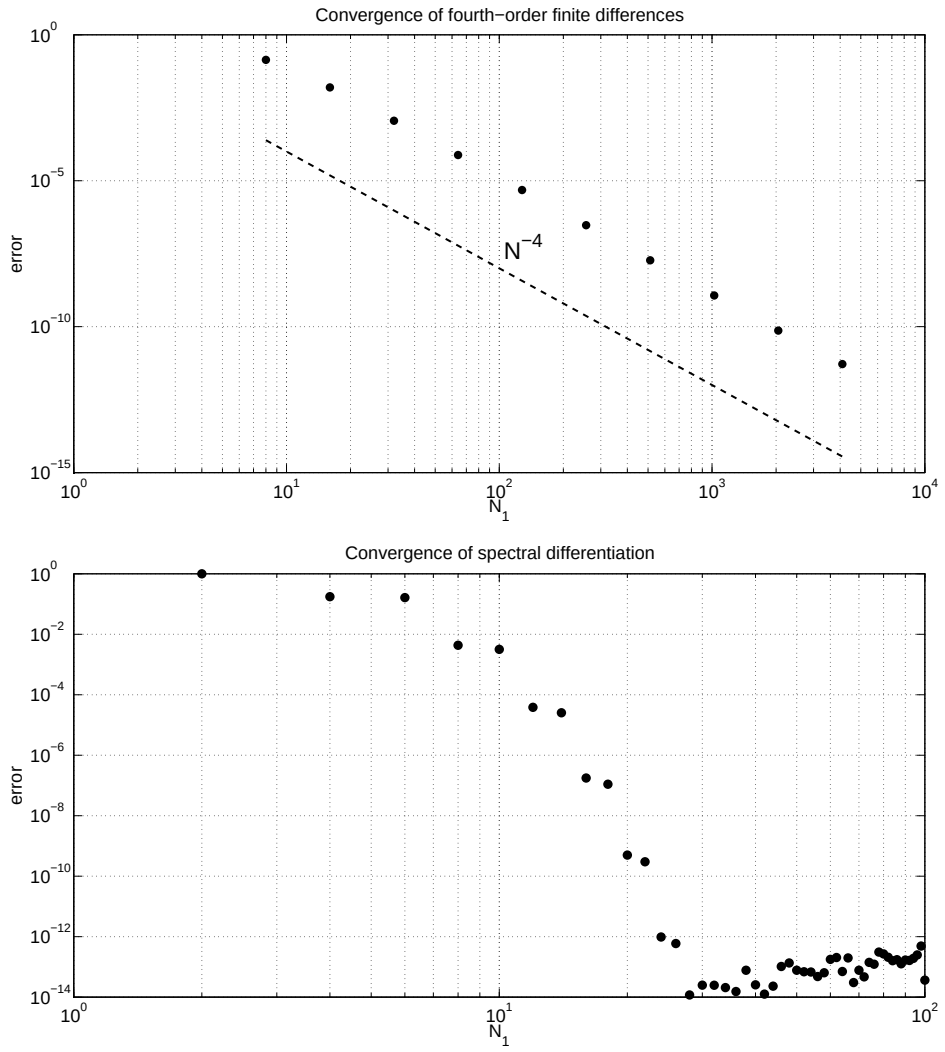


Figure 4.1.2: Left) The convergence of fourth-order 1D differentiation matrix eq.(4.12) versus Right) Spectral collocation eq.(4.13) for different number of points in the first direction N_1.(after Trefethen (2001))

Here we bring code listings that we used in our implementation. Listing (4.2) contains a sub program for calculating upwind differentiation matrix (eq.(4.10)). Listing (4.3) contains sub program for calculating both fourth-order matrix (4.12) and spectral collocation matrix (4.13). The type of the output is specified based on the input switch named "jumper". We will use this sub program in Sec.(4.3.2) for multidimensional periodic convection problem.

Listing 4.2: m-code for first-order upwind scheme

```
1 %This function creates the first−order upwind derivative operator for a
2 %system of fluxes
3 %NOTE c, dx and other constants arising from discretization are used in
4 %the caller routine not here. Here is only the constant body of the
```

```
%operator.
function D = upwind_1st(nD,y)

d = cell(nD,nD); %Allocating the final operator

%substituting all sparse cells with zero matrices so that cell2mat can work
for i = 1:nD
    for j=1:nD
        d{i,j} = zeros(y);
    end
end

d{1,1} = 1*eye(y);
%d{1,2} = 1*eye(y);
for i=2:nD
    d{i,i-1} = -1*eye(y);
    d{i,i} = 1.*eye(y);
end

%The output derivative operator is complete now. Converting cell to matrix
D = cell2mat(d);
```

Listing 4.3: m-code for fourth-order and spectral differentiation matrix

```
%This function computes the one-dimensional derivative operator for a
%system of fluxes using different methods
%NOTE: use examine_D.m to test various differentiation methods
function D = gen_1D_diff(nD,y,h,jumper)
%inputs
%nD the outer-block size of the operator
%y the number of equations in the flux (size of inner block)
%h spacing
%jumper a switch to select appropriate method
% jumper = 1 = fourth-order central periodic
% jumper = 2 = spectral perodic based on toeplitz matrix

%output
%D the block-differentiation matrix in sparse matrix form

%the number of points in one-D space is N to be compatible with Spectral ...
    methods in Matlab Book
N = nD;

%choosing appropriate differentiation method
switch jumper
    case 1 %fourth-order central periodic
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %The following is copied from "p1.m" of Spectral methods in Matlab Book
        % Construct sparse fourth-order differentiation matrix:
        e = ones(N,1);
        D =   sparse(1:N,[2:N 1],2*e/3,N,N)...
            - sparse(1:N,[3:N 1 2],e/12,N,N);
        D = (D-D')/h;
    case 2 %spectral perodic based on toeplitz matrix
```

```
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %The following is copied from "p2.m" of Spectral methods in Matlab Book
        % Construct sparse fourth-order differentiation matrix:

        % Construct spectral differentiation matrix:
        column = [0 .5*(-1).^(1:N-1).*cot((1:N-1)*h/2)];
        D = toeplitz(column,column([1 N:-1:2]));
end

%Converting D to sub-blocks for y dimension
d = cell(nD,nD); %Allocating the final operator
%substituting all sparse cells with zero matrices so that cell2mat can work
for i = 1:nD
    for j=1:nD
        d{i,j} = sparse(D(i,j)*eye(y));
    end
end
%The output derivative operator is complete now. Converting cell to matrix
D = cell2mat(d);
```

In fig.(4.1.2) we demonstrated the fast convergence of spectral schemes to machine zero based on the number of required spatial nodes. However it is also worthwhile to examine the spatial resolution of spectral schemes as well. To do this, we evaluate numerical derivative of the following vector function,

$$\vec{u}\,(x) = \begin{pmatrix} \cos(4x) \\ \sin(4x) + 1 \\ \sin(x) \end{pmatrix}, \tag{4.14}$$

We choose the interval $x = [0, 2\pi]$. The first two entries of vector $\vec{u}$ in (4.14) are selected to be high-frequency such that they possess four wave lengths per differentiation interval. The last entry, i.e. $\sin(x)$ is the low-frequency component which has only one wave length per x. We use the program given in Listing (4.3) to generate fourth-order matrix operator (4.12) using switch 'jumper=1' and spectral matrix (4.13) using switch 'jumper = 2'. The numerical differentiation is performed with ten nodes (points) in space to assess the resolution of schemes. The analytical derivatives are shown in fig.(4.1.3) and fig.(4.1.4) with solid line and numerical values are shown with blue circles. According to fig.(4.1.3), it is evident that fourth-order scheme performs well in resolving low-frequency component (du_3/dx) however it has substantial error when applied to high-frequency waves (du_1/dx and du_2/dx). However, in fig.(4.1.4) with the same number of points in the space, we see that spectral scheme has excellent resolution for high-frequency components. Figures (4.1.3) and (4.1.4) are generated using Listing (4.4).

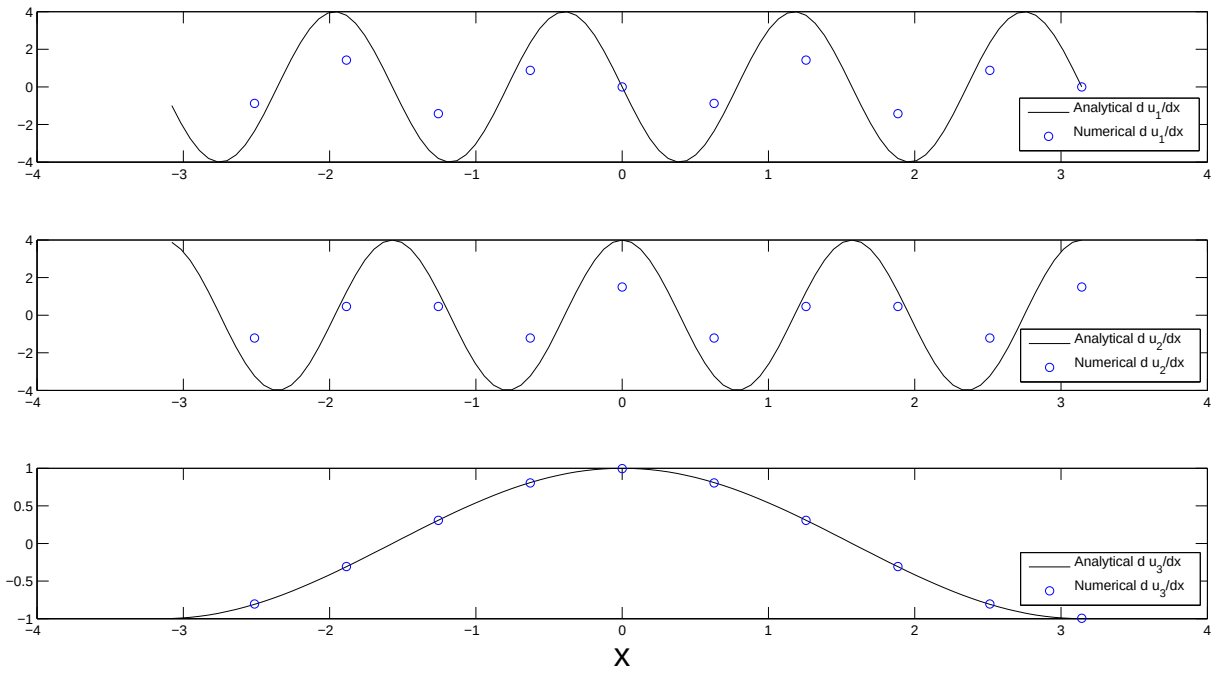


Figure 4.1.3: Numerical differentiation using fourth-order matrix operator (4.12). $N_1 = 10$.

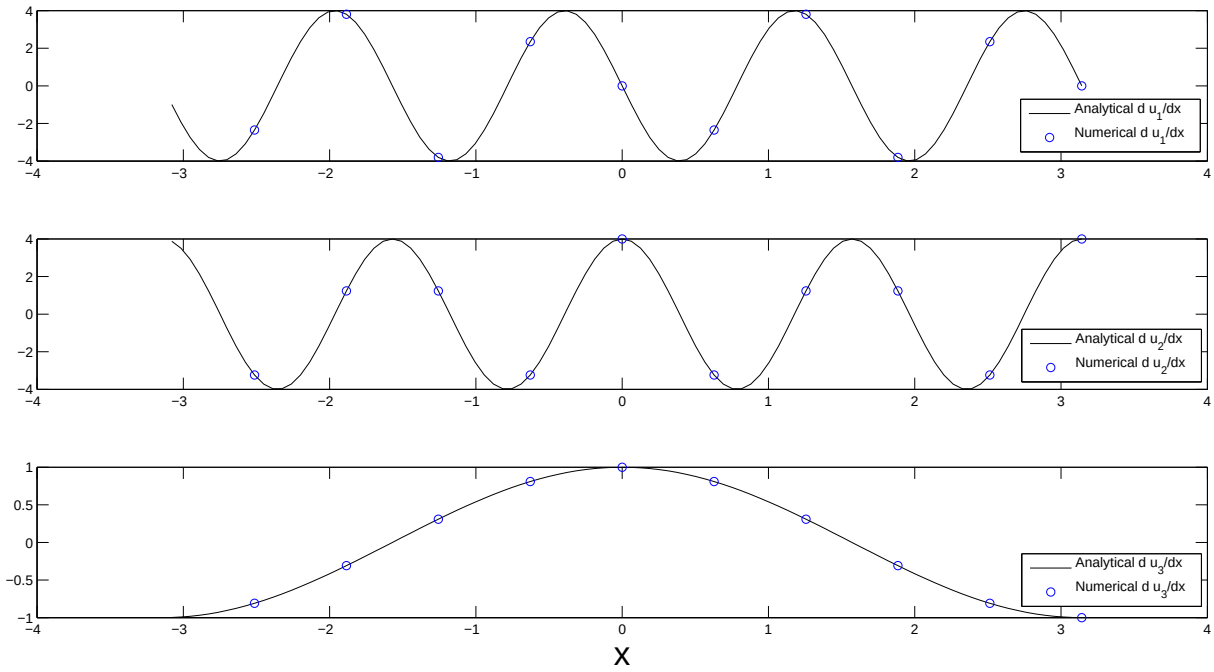


Figure 4.1.4: Numerical differentiation using spectral matrix operator (4.13). $N_1 = 10$.

Listing 4.4: m-code for generating comparison between fourth-order and spectral differentiating given in figs. (4.1.3) and (4.1.4).

```
%examine_D : examines the differentiating operator D before using in
%real problem
clear all;
clc

%constants here
L = 2*pi; %the period
nD = 10; %nomber of spatial nodes
n_anal = 100; %number of spatial nodes for analytical solution
y = 3; %number of equations

%the spatial node distribution
dx = L/nD;
x = -L/2+(1:nD)*dx;
XX = -L/2+(1:n_anal)*L/n_anal;

%the target functions
y1 = cos(4*x);
y2 = sin(4*x)+1;
y3 = sin(x);

%The analytical differentiation of the target functions
dy1 = -4*sin(4*XX);
dy2 = 4*cos(4*XX);
dy3 = cos(XX);

%allocating the solution vector in cells
U = cell(nD,1);

%filling-up the solution vector with target functions
for i = 1:nD
    U{i} = [y1(i);
            y2(i);
            y3(i)];
end

%creating the differentiation operator
D = gen_1D_diff(nD,y,dx,2);

DUdx = D * cell2mat(U);
DUdx = mat2cell(DUdx, y*ones(nD,1),1);

for i = 1:nD
    temp_DU = DUdx{i};
    Dy1(i) = temp_DU(1);
    Dy2(i) = temp_DU(2);
    Dy3(i) = temp_DU(3);
end

figure(1);
subplot(y,1,1);
plot(XX,dy1,'k-',x,Dy1,'o');
legend('Analytical d u_1/dx','Numerical d u_1/dx');
```

```
subplot(y,1,2);
plot(XX,dy2,'k-',x,Dy2,'o');
legend('Analytical d u_2/dx','Numerical d u_2/dx');
subplot(y,1,3);
plot(XX,dy3,'k-',x,Dy3,'o');
legend('Analytical d u_3/dx','Numerical d u_3/dx');
xlabel('x');
```

To test the validity of (4.5), we study the following numerical example. Consider three-dimensional scalar field

$$u\left(x,y,t\right)=\sin\left(xy\right)t \tag{4.15}$$

defined on (x,y,t) = ([0,1],[0,1],[0,1]). Here the initial (default) direction is $x_1 = x$ and the second direction is $x_2 = y$ and the third is $x_3 = t$. We want to find the numerical equivalent for analytical operator $\left(\frac{\partial}{\partial x}+\frac{\partial}{\partial y}\right)(\square)$. For differentiation in x_1 we simply have

$$\frac{\partial}{\partial x}u(x,y,t)=\mathbb{D}_x\mathbf{u}+\mathcal{O}\left(\Delta y^{p_1}\right) \tag{4.16}$$

For differentiation in 'y', we note that, according to eq.(4.5),

$$\frac{\partial}{\partial y}u(x,y,t)=\mathbf{\Omega}_{12}\left(\mathbb{D}_y\mathbf{\Omega}_{21}\mathbf{u}+\mathbf{b}_2+\mathcal{O}\left(\Delta y^{p_2}\right)\right), \tag{4.17}$$

Adding up the lhs and rhs of (4.16) and (4.17), we can easily obtain

$$\frac{\partial}{\partial x}u(x,y,t)+\frac{\partial}{\partial y}u(x,y,t)=\mathbb{D}_x\mathbf{u}+\mathcal{O}\left(\Delta y^{p_1}\right)+\mathbf{\Omega}_{12}\left(\mathbb{D}_y\mathbf{\Omega}_{21}\mathbf{u}+\mathbf{b}_2+\mathcal{O}\left(\Delta y^{p_2}\right)\right), \tag{4.18}$$

According to (4.15), u is zero for x=y=t=0. So we choose these faces to contain boundary points and thus the boundary condition $\mathbf{b}_2$ is always zero. Also by removing the order operator $\mathcal{O}\left(\right)$ from eq.(4.18) we obtain

$$\left(\frac{\partial}{\partial x}+\frac{\partial}{\partial y}\right)u(x,y,t)=\mathbb{D}_x\mathbf{u}+\mathbf{\Omega}_{12}\mathbb{D}_y\mathbf{\Omega}_{21}\mathbf{u}=\left(\mathbb{D}_x+\mathbf{\Omega}_{12}\mathbb{D}_y\mathbf{\Omega}_{21}\right)\mathbf{u}, \tag{4.19}$$

In other words,

$$\frac{\partial}{\partial x}+\frac{\partial}{\partial y}=\mathbb{D}_x+\mathbf{\Omega}_{12}\mathbb{D}_y\mathbf{\Omega}_{21} \tag{4.20}$$

One can readily extend (4.20) to three-dimensional differentiation as,

$$\frac{\partial}{\partial x}+\frac{\partial}{\partial y}+\frac{\partial}{\partial z}=\mathbb{D}_x+\mathbf{\Omega}_{12}\mathbb{D}_y\mathbf{\Omega}_{21}+\mathbf{\Omega}_{13}\mathbb{D}_z\mathbf{\Omega}_{31}, \tag{4.21}$$

or z-dimensional derivative

$$\begin{aligned} \sum_{j=1}^{z} \frac{\partial}{\partial x_j} &= \sum_{j=1}^{z} \mathbf{\Omega}_{1j} \mathbb{D}_j \mathbf{\Omega}_{j1}, \\ \mathbf{\Omega}_{11} &= \mathbb{I} \end{aligned} \tag{4.22}$$

where $\mathbb{D}_1 = \mathbb{D}_x$, $\mathbb{D}_2 = \mathbb{D}_y$, $\mathbb{D}_3 = \mathbb{D}_z$ and so on.

A Matlab m-code is brought in Listing (4.5) which implements (4.19). It calculates operator $\mathbf{\Omega}_{21}$ using Listing (4.1) and also it uses Listing (4.2) to obtain one-dimensional first-order upwind matrices required for x and y directions. Note that this may be done using a spectral differentiation matrix (will be introduced later) only by calling another external function instead of Listing (4.2). At the core of implementation we encounter with the following line,

```
Div = kron(eye(N2*N3),1/dx*D1) +
                (OMEGA_21)^(-1)*kron(eye(N1*N3),1/dy*D2)*OMEGA_21;
```

It actually utilizes Kronecker matrix product to put differentiation block matrices on the main diagonal of the parent matrix to make a multidimensional matrix operator for differentiation. For example D_1 is N_1 by N_1 matrix. The expression "kron(eye(N2*N3),1/dx*D1)" puts $\frac{1}{dx}D_1$ all over the main diagonal of a parent matrix with size $(N_2 \times N_3)$ by $(N_2 \times N_3)$, so that the size of resulting block matrix is $(N_1 \times N_2 \times N_3)$ by $(N_1 \times N_2 \times N_3)$.

Listing 4.5: The numerical implementation of eq.(4.19).

```
%This fuction creates and tests numerical divergence for sample discrete
%space
clear all;
clc
%defining ranges
x_min = 0; x_max = 1; y_min = 0; y_max = 1; t_min = 0; t_max = 1;
%number of points
N1 = 10; N2 = 10; N3 = 4;
%number of equations
y=1;
%spacings
dx = (x_max - x_min)/(N1-1); dy = (y_max - y_min)/(N2-1);

%creating rearrangement operator for 1->2
OMEGA_21 = OMEGA_x1([N1 N2 N3],y,2);
%creating one-dimensional upwind operators for differentiating in x and y
D1 = upwind_1st(N1,y);
D2 = upwind_1st(N2,y);
```

```
% Creating d/dx+d/dy
Div = kron(eye(N2*N3),1/dx*D1) + (OMEGA_21)^(-1)*kron(eye(N1*N3),1/dy*D2)*OMEGA_21;

%creating a dummy field and shifting it to make boundary conditions b=0
x_x=dx+linspace(x_min,x_max,N1);
y_y=dy+linspace(y_min,y_max,N2);
t_t = linspace(t_min,t_max,N3);

%filling the solution vector
for l=1:N3
    for i=1:N1
        for j=1:N2
            u(j,1) = sin(x_x(i)*y_y(j))*t_t(l);
        end
        U(i,1) = {u};
    end
    UU(l,1) = {cell2mat(U)};
end
%calculating numerical divergence
DivU = Div*cell2mat(UU);
%calculating analytical divergence
for l=1:N3
    for i=1:N1
        for j=1:N2
            v(j,1) = (y_y(j)*cos(x_x(i)*y_y(j))+x_x(i)*cos(x_x(i)*y_y(j)))*t_t(l);
        end
        V(i,1) = {v};
    end
    VV(l,1) = {cell2mat(V)};
end

divV = cell2mat(VV);

%exporting to tecplot
divV = mat2cell(divV,N1*N2*ones(N3,1),1);
DivU = mat2cell(DivU,N1*N2*ones(N3,1),1);
for i=1:N3
    divV{i} = mat2cell(divV{i},N1*ones(N2,1),1);
    DivU{i} = mat2cell(DivU{i},N1*ones(N2,1),1);
end

fid = fopen('XY.tec','w');
fprintf(fid,'VARIABLES = "X", "Y", "T","Numerical"\n');
fprintf(fid,'ZONE T="Num", I=%d, J=%d, K=%d F=POINT\n',N1,N2,N3);

for l=1:N3
    tempU = DivU{l};
    for j=1:N2
        %tempV = cell2mat(divV(i));
        tempUU = tempU{j};
        for i=1:N1
            fprintf(fid,'%e %e %e %e\n',x_x(i),y_y(j),t_t(l),tempUU(i));
        end
    end
end

```

```
fprintf(fid,'VARIABLES = "X", "Y", "T","Analytical"\n');
fprintf(fid,'ZONE T="An", I=%d, J=%d, K=%d F=POINT\n',N1,N2,N3);

for l=1:N3
    tempU = divV{l};
    for j=1:N2
        %tempV = cell2mat(divV(i));
        tempUU = tempU{j};
        for i=1:N1
            fprintf(fid,'%e %e %e %e\n',x_x(i),y_y(j),t_t(l),tempUU(i));
        end
    end
end

fclose(fid);
%end of program
```

The results are shown in fig.(4.1.5). Although very coarse grid (N1 = 10, N2 = 10, N3 = 4) is used and the differentiation matrix is first-order, the result are in good agreement with analytical differentiation.

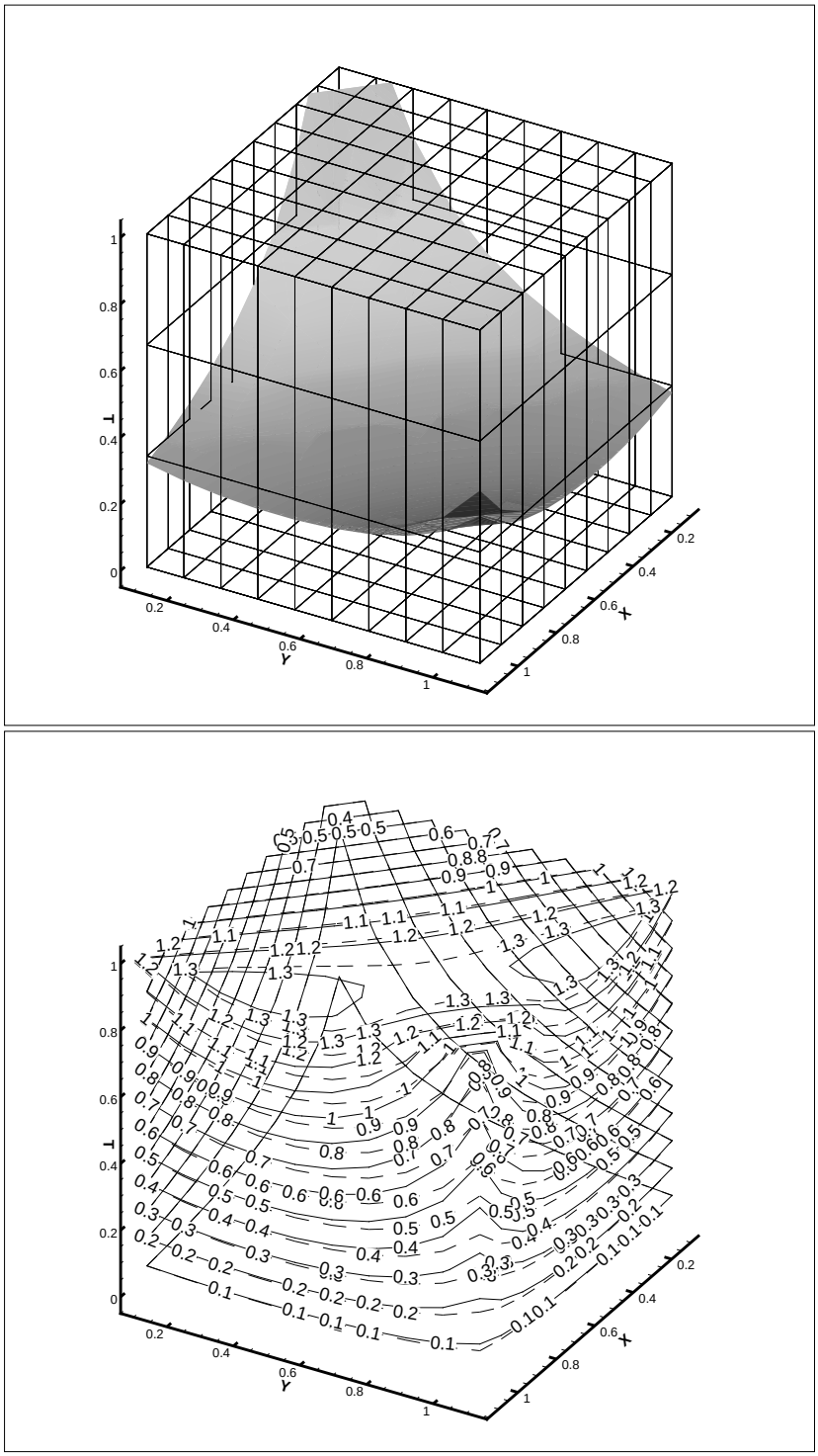


Figure 4.1.5: The numerical derivative $\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y}$ for three dimensional scalar field $u(x, y, t) = \sin(xy)t$ obtained using eq.(4.19). TOP: Two iso-surface, top is analytical derivative while bottom is the numerical. BOTTOM: three-dimensional contours, solid is analytical- dashed is numerical. Please note that only 4 points are used to discretize the field in the time direction.

4.2 Discretization of Spatial Operators

The spatial operator in Partial differential equations in conservative form is usually a combination of Divergence and Laplacian which is called convection-diffusion from the physical point of view. However, there are plenty of higher-order PDEs in computational science and engineering like equations governing plasticity. Therefore to include all cases in a single operator in a compact way, we will introduce a generic linear differentiation operator in this section.

Let us start by considering the analytical operator 'Divergence' say,

$$\text{Div.}\left(\vec{u}\right) = \nabla.\vec{u} = \frac{\partial u_j}{\partial x_j} \tag{4.23}$$

Applying numerical divergence to a discrete space is analogous to applying analytical divergence to continuous vector field $\vec{u}$. One can write using (4.5) in consistent format with (4.22)

$$\begin{aligned}
\left[\nabla\right]\mathbf{u} &= \sum_j \mathbf{\Omega}_{1j}\left(\mathbb{D}_j\mathbf{\Omega}_{j1}\mathbf{u} + \mathbf{b}_j + \mathcal{O}\left(\Delta {x_j}^{p_j}\right)\right) \\
&= \sum_j \left(\mathbf{\Omega}_{1j}\mathbb{D}_j\mathbf{\Omega}_{j1}\right)\mathbf{u} + \sum_j \mathbf{\Omega}_{1j}\mathbf{b}_j + \sum_j \mathbf{\Omega}_{1j}\mathcal{O}\left(\Delta {x_j}^{p_j}\right),
\end{aligned} \tag{4.24}$$

Where $\mathbb{D}_j$ is the discretization matrix of *the first derivative* in the 'jth' direction and the column vector $\mathbf{b}_j$ imposes *boundary conditions* required at the 'j' direction (eq.(4.5)) and the last term is the truncation error. Note that in the inner parenthesis in (4.24-top), we first computed the differentiation in 'j'-arrangement and then the result are taken back to original arrangement (direction 1) by left-multiplying with rearrangement operator $\mathbf{\Omega}_{1j}$ as described completely before. The consistency condition for numerical divergence (4.24) is,

$$\lim_{\Delta x_j \to 0}\left[\nabla\right] = \nabla, \tag{4.25}$$

Therefore, in quickly discretizing equations we usually use $\left[\nabla\right]\mathbf{u} = \sum_j \left(\mathbf{\Omega}_{1j}\mathbb{D}_j\mathbf{\Omega}_{j1}\mathbf{u} + \mathbf{\Omega}_{1j}\mathbf{b}_j\right)$ by neglecting the truncation error unless in situations [4] that convergence or the accuracy of the algorithm is sought. Similar expression can be written for Laplacian operator

$$\text{Div}^2.\left(\vec{u}\right) = \nabla^2.\vec{u} = \frac{\partial^2 u_j}{\partial x_j^2} \tag{4.26}$$

[4] like Sec.(4.3)

just by replacing the first derivative differentiation matrix $\mathbb{D}_j$ in (4.24) with the second derivative $\mathbb{D}_j^2$ as follows[5],

$$\begin{aligned}\left[\nabla^2\right]\mathbf{u} &= \sum_j \mathbf{\Omega}_{1j}\left(\mathbb{D}_j^2\mathbf{\Omega}_{j1}\mathbf{u} + \mathbf{b}_j + \mathcal{O}\left(\Delta {x_j}^{p_j}\right)\right) \\ &= \sum_j \mathbf{\Omega}_{1j}\mathbb{D}_j^2\mathbf{\Omega}_{j1}\mathbf{u} + \sum_j \mathbf{\Omega}_{1j}\mathbf{b}_j + \sum_j \mathbf{\Omega}_{1j}\mathcal{O}\left(\Delta {x_j}^{p_j}\right), \end{aligned} \tag{4.27}$$

Following the trend of discretization method shown in eqs.(4.24,4.27) and as a generalization to Divergence and Laplacian, we consider the following arbitrary-order derivative operator[6],

$$\Upsilon = \sum_i \sum_j \frac{\partial^i}{\partial x_j} \tag{4.28}$$

When applied to vector field $\vec{u}$ it gives the combination of Divergence, Laplacian and higher-order derivatives as follows,

$$\Upsilon.\vec{u} = \sum_i \sum_j \frac{\partial^i}{\partial x_j} u_j \tag{4.29}$$

Or in the discretized form

$$\begin{aligned}\left[\Upsilon\right]\mathbf{u} &= \sum_i \sum_j \mathbf{\Omega}_{1j}\left(\mathbb{D}_j^i\mathbf{\Omega}_{j1}\mathbf{u} + \mathbf{b}_{ij} + \mathcal{O}\left(\Delta {x_j}^{p_{ij}}\right)\right) \\ &= \sum_i \sum_j \mathbf{\Omega}_{1j}\mathbb{D}_j^i\mathbf{\Omega}_{j1}\mathbf{u} + \sum_i \sum_j \mathbf{\Omega}_{1j}\mathbf{b}_{ij} \\ &+ \sum_i \sum_j \mathbf{\Omega}_{1j}\mathcal{O}\left(\Delta {x_j}^{p_{ij}}\right), \end{aligned} \tag{4.30}$$

Where $\mathbb{D}_j^i$ is the discretization matrix for the i-th derivative of $\mathbf{u}$ in the j-th direction and $\mathbf{b}_{ij}$ is the column vector imposing boundary conditions needed for the i-th derivative of $\mathbf{u}$ in the j-th direction. Note that in special cases where we use the assumption of *periodic direction j*, it means that the boundary condition vector $\mathbf{b}_{ij}$ is no longer needed since the assumption of periodicity is implemented in the differentiation operator $\mathbb{D}_j^i$ itself. In addition for high-order spatial discretization, the truncation error $\mathcal{O}\left(\Delta {x_j}^{p_{ij}}\right)$ in (4.30) is practically zero. Therefore the generic

[5]The power '2' used is not $\mathbb{D}_j^2 \neq \mathbb{D}_j \times \mathbb{D}_j$ but it shows that $\mathbb{D}_j^2$ contains the coefficient of finite difference or spectral discretization of the *second derivative*.

[6]Equation (4.28) reduces to Divergence for $i = 1$ and to Laplacian for $i = 2$.

operator $[\Upsilon]$ in this situation is abbreviated to

$$[\Upsilon]\,\mathbf{u} = \sum_i \sum_j \mathbf{\Omega}_{1j} \mathbb{D}_j^i \mathbf{\Omega}_{j1} \mathbf{u} \tag{4.31}$$

Now we are ready to discretize a generalized conservative PDE using Discrete Picard Iteration method.

4.3 General Structured Formulation

Let us consider the following 'z-dimensional' generalized conservative PDE[7],

$$\frac{\partial \mathbf{u}}{\partial t} + \Upsilon \mathbf{F} = \mathbf{G} \tag{4.32}$$

Where $\mathbf{u}$ is the vector of conservative variables and $\mathbf{F} = [\mathbf{F}_1|\mathbf{F}_2|\dots|\mathbf{F}_j|\dots|\mathbf{F}_{z-1}]$ is the flux tensor and $\Upsilon = \sum_i \sum_j^{z-1} \frac{\partial^i}{\partial x_j}$ is the generalized derivative operator (defined in eq.(4.29)) and $\mathbf{G}$ is a source term which only depends on solution vector $\mathbf{u}$ and time. Equation (4.32) includes propagation phenomena (Acoustics, Electromagnetics, ...) for $i = 1$ and it includes generic convection-diffusion phenomena (like Navier-Stokes equation) for $i = 2$. Since we derive DPI scheme using residual-based integral form of ODEs/PDEs, we first need to write (4.32) in the integral form below.

$$\begin{aligned} \mathbf{u} &= \mathbf{u}_0 + \int_0^t \mathbf{R} dt \\ \mathbf{R} &= -\Upsilon \mathbf{F} + \mathbf{G}, \end{aligned} \tag{4.33}$$

where the residual $\mathbf{R}$ appears in the formulation *to preserve the consistency* with previous sections. To find the DPI solution of (4.33) we use two different formulations (1- Directly discretizing Υ. 2- Fully integrated in space-time.) which are presented in the following sections.

4.3.1 Directly discretizing Υ

In this formulation we replace Υ in residual in eq.(4.33-bottom) with the fully discretized eq.(4.30) and then the integral equation (4.33-top) is discretized using implicit DPI formulation (3.15). Substituting (4.30) into (4.33-bottom) we obtain

[7] z-1 spatial dimensions plus one temporal dimension = z dimensions.

$$\mathbf{R} = -\sum_i \sum_j \mathbf{\Omega}_{1j} \mathbb{D}_j^i \mathbf{\Omega}_{j1} \mathbf{F}_{ij} - \sum_i \sum_j \mathbf{\Omega}_{1j} \mathbf{b}_{ij} - \sum_i \sum_j \mathbf{\Omega}_{1j} \mathcal{O}\left(\Delta {x_j}^{p_{ij}}\right) + \mathbf{G}, \quad (4.34)$$

From the implicit DPI formulation (3.15) we note that we still need the Jacobian matrix. To obtain Jacobian, we simply take the derivative of (4.34) with respect to $\mathbf{u}$ as presented below

$$\frac{\partial \mathbf{R}}{\partial \mathbf{u}} = -\sum_i \sum_j \mathbf{\Omega}_{1j} \mathbb{D}_j^i \mathbf{\Omega}_{j1} \frac{\partial \mathbf{F}_{ij}}{\partial \mathbf{u}} + \frac{\partial \mathbf{G}}{\partial \mathbf{u}}, \quad (4.35)$$

Interestingly, we note that there is no truncation error in Jacobian (4.35). Here we repeat eq.(3.15) with these differences that the truncation error only represents temporal error and the rearrangement operator in the time direction, i.e. $\mathbf{\Omega}_{z1}$ is reindexed to $\mathbf{\Omega}_{t1}$;

$$\begin{aligned} &\Big(\mathbb{I} \;-\; \frac{a'\Delta t}{a'+b'} \mathbf{\Omega}_{1t} \,\mathbb{S}\, \mathbf{\Omega}_{t1} \frac{\partial \mathbf{R}}{\partial \mathbf{u}_n}\Big)\mathbf{u}_{n+1} = \quad (4.36) \\ &\mathbf{u}_0 \;+\; \Delta t \mathbf{\Omega}_{1t} \,\mathbb{S}\, \mathbf{\Omega}_{t1} \left(\mathbf{R}_n - \frac{a'}{a'+b'} \frac{\partial \mathbf{R}}{\partial \mathbf{u}_n} \mathbf{u}_n\right) + \Delta t \mathbf{\Omega}_{1t} \,\mathbb{S}_i\, \mathbf{\Omega}_{t1} \mathbf{R}_i + \mathcal{O}\left(\Delta t^p\right) \end{aligned}$$

Thus for the 'z-dimensional' space-time we have 'z' rearrangement operators $\{[\mathbf{\Omega}]_{11} = \mathbb{I}, [\mathbf{\Omega}]_{21}, \ldots, [\mathbf{\Omega}]_{(z-1)1}, [\mathbf{\Omega}]_{t1}\}$ which the first 'z' operators rearrange spatial data and the last operator rearrange data in time. We first substitute $\frac{\partial \mathbf{R}}{\partial \mathbf{u}_n}$ and $\mathbf{R}_n$ from (4.35) and (4.34) into the parenthesis on rhs of (4.36), i.e. $\left(\mathbf{R}_n - \frac{a'}{a'+b'} \frac{\partial \mathbf{R}}{\partial \mathbf{u}_n} \mathbf{u}_n\right)$ leading to

$$\begin{aligned} \mathbf{R}_n - \frac{a'}{a'+b'} \frac{\partial \mathbf{R}_n}{\partial u} \mathbf{u}_n = \; &- \sum_i \sum_j \mathbf{\Omega}_{1j} \mathbb{D}_j^i \mathbf{\Omega}_{j1} \left(\mathbf{F}_{(n)ij} - \frac{a'}{a'+b'} \frac{\partial \mathbf{F}_{ij}}{\partial \mathbf{u}_n} \mathbf{u}_n\right) \\ &- \sum_i \sum_j \mathbf{\Omega}_{1j} \mathbf{b}_{ij} - \sum_i \sum_j \mathbf{\Omega}_{1j} \mathcal{O}\left(\Delta {x_j}^{p_{ij}}\right) \\ &+ \mathbf{G}_n - \frac{a'}{a'+b'} \frac{\partial \mathbf{G}_n}{\partial \mathbf{u}_n} \mathbf{u}_n, \quad (4.37) \end{aligned}$$

Now, by substituting (4.37) and (4.35) into (4.36) we obtain

$$\begin{aligned} &\Big(\mathbb{I} \;-\; \frac{a'\Delta t}{a'+b'} \mathbf{\Omega}_{1t} \,\mathbb{S}\, \mathbf{\Omega}_{t1} \Big(-\sum_i \sum_j \mathbf{\Omega}_{1j} \mathbb{D}_j^i \mathbf{\Omega}_{j1} \frac{\partial \mathbf{F}_{ij}}{\partial \mathbf{u}_n} + \frac{\partial \mathbf{G}}{\partial \mathbf{u}_n}\Big)\Big)\mathbf{u}_{n+1} = \quad (4.38) \\ &\mathbf{u}_0 \;+\; \Delta t \mathbf{\Omega}_{1t} \,\mathbb{S}\, \mathbf{\Omega}_{t1} \left(\begin{array}{l} -\sum_i \sum_j \mathbf{\Omega}_{1j} \mathbb{D}_j^i \mathbf{\Omega}_{j1} (\mathbf{F}_{(n)ij} - \frac{a'}{a'+b'} \frac{\partial \mathbf{F}_{ij}}{\partial \mathbf{u}_n} \mathbf{u}_n) \text{ -} \\ \sum_i \sum_j \mathbf{\Omega}_{1j} \mathbf{b}_{ij} - \sum_i \sum_j \mathbf{\Omega}_{1j} \mathcal{O}(\Delta {x_j}^{p_{ij}}) + \\ \mathbf{G}_n - \frac{a'}{a'+b'} \frac{\partial \mathbf{G}_n}{\partial \mathbf{u}_n} \mathbf{u}_n \end{array} \right) \\ &\quad + \; \Delta t \mathbf{\Omega}_{1t} \,\mathbb{S}_i\, \mathbf{\Omega}_{t1} \mathbf{R}_i + \mathcal{O}\left(\Delta t^p\right) \quad (4.39) \end{aligned}$$

where

$$\mathbf{R}_i = \mathbf{R}\left(\mathbf{u}_0\right) \tag{4.40}$$

Before simplifying (4.38), we define the total operator $\mathbf{T}$ as follows

$$\mathbf{T}_{ij}\left(\square\right) = \Delta t\left(\mathbf{\Omega}_{1t}\,\mathbb{S}\,\mathbf{\Omega}_{t1}\right)\left(\mathbf{\Omega}_{1j}\mathbb{D}_j^i\mathbf{\Omega}_{j1}\right)\left(\square\right) = \Delta t\;\mathbf{\Omega}_{1t}\,\mathbb{S}\,\mathbf{\Omega}_{tj}\mathbb{D}_j^i\mathbf{\Omega}_{j1}\left(\square\right) \tag{4.41}$$

We notice that $\mathbf{T}_{ij}$ is a total operator which means that it includes differentiation in space and integration in time *simultaneously*. Mathematically speaking, according to rhs-(4.41) the operand $(\square)$ is first rearranged in the j direction in space $(1 \leq j \leq (z-1))$ using $\mathbf{\Omega}_{j1}$ and then the i^{th} derivative is taken using $\mathbb{D}_j^i$ operator and then the result is rearranged in the time direction 't' using $\mathbf{\Omega}_{tj} = \mathbf{\Omega}_{t1}\mathbf{\Omega}_{1j}$ and then integration is performed in the time direction (z^{th} direction) by the integration operator $\Delta t\;\mathbb{S}$ and the final result is rearranged back to the original arrangement '1'. The *analytical* analogous of $\mathbf{T}_{ij}$ is ,

$$T_{ij}\left(\square\right) = \int_0^{\Delta t} \frac{\partial^i}{\partial x_j}\left(\square\right), \tag{4.42}$$

Now, using total operator it is possible to abbreviate (4.38) as below

$$\left(\mathbb{I} \;+\; \frac{a'}{a'+b'}\sum_i\sum_j \mathbf{T}_{ij}\frac{\partial \mathbf{F}_{ij}}{\partial \mathbf{u}_n} - \frac{a'\Delta t}{a'+b'}\mathbf{\Omega}_{1t}\,\mathbb{S}\,\mathbf{\Omega}_{t1}\frac{\partial \mathbf{G}}{\partial \mathbf{u}_n}\right)\mathbf{u}_{n+1} = \tag{4.43}$$

$$\begin{aligned}\mathbf{u}_0 \;+\; &\left(\begin{array}{l} -\sum_i\sum_j \mathbf{T}_{ij}\left(\mathbf{F}_{(n)ij} - \frac{a'}{a'+b'}\frac{\partial \mathbf{F}_{ij}}{\partial \mathbf{u}_n}\mathbf{u}_n\right) - \\ \sum_i\sum_j \Delta t\mathbf{\Omega}_{1t}\,\mathbb{S}\,\mathbf{\Omega}_{tj}\mathbf{b}_{ij} - \sum_i\sum_j \Delta t\mathbf{\Omega}_{1t}\,\mathbb{S}\,\mathbf{\Omega}_{tj}\mathcal{O}\left(\Delta {x_j}^{p_{ij}}\right) + \\ \left(\Delta t\mathbf{\Omega}_{1t}\,\mathbb{S}\,\mathbf{\Omega}_{t1}\right)\left(\mathbf{G}_n - \frac{a'}{a'+b'}\frac{\partial \mathbf{G}_n}{\partial \mathbf{u}_n}\mathbf{u}_n\right) \end{array}\right) \\ &+\; \Delta t\mathbf{\Omega}_{1t}\,\mathbb{S}_i\,\mathbf{\Omega}_{t1}\mathbf{R}_i + \mathcal{O}\left(\Delta t^p\right)\end{aligned} \tag{4.44}$$

We notice that the *total truncation error* on rhs on (4.43) is composed of temporal error and spatial truncation error in directions. We consider the total truncation error appearing in eq.(4.43) by summing spatial and temporal truncation errors,

$$\begin{aligned} T.T.E &= -\sum_i\sum_j \Delta t\mathbf{\Omega}_{1t}\,\mathbb{S}\,\mathbf{\Omega}_{tj}\mathcal{O}\left(\Delta {x_j}^{p_{ij}}\right) + \mathcal{O}\left(\Delta t^p\right) \\ &= \mathcal{O}\left(\Delta t^p, \Delta {x_1}^{p_1}, \Delta {x_2}^{p_2}, \ldots, \Delta {x_{z-1}}^{p_{z-1}}\right) \end{aligned} \tag{4.45}$$

We also define the constant column vector $\bar{\mathbf{u}}_0$ as

$$\bar{\mathbf{u}}_0 = \mathbf{u}_0 - \sum_i\sum_j \Delta t\mathbf{\Omega}_{1t}\,\mathbb{S}\,\mathbf{\Omega}_{tj}\mathbf{b}_{ij} + \Delta t\mathbf{\Omega}_{1t}\,\mathbb{S}_i\,\mathbf{\Omega}_{t1}\mathbf{R}_i \tag{4.46}$$

Note that $\bar{\mathbf{u}}_0$ depends only on values which are *constant* during Picard iteration. It depends on initial value $\mathbf{u}_0$, the initial residuals $\mathbf{R}_i = \mathbf{R}(\mathbf{u}_0)$ and the vector of boundary conditions for all high-order derivatives in all spatial directions, i.e $\mathbf{b}_{ij}$ which is merely a source term. Using (4.46) and (4.45), eq.(4.43) can be written as

$$\begin{aligned}
&\Big(\quad \mathbb{I} \quad + \frac{a'}{a'+b'} \sum_i \sum_j \mathbf{T}_{ij} \frac{\partial \mathbf{F}_{ij}}{\partial \mathbf{u}_n} - \frac{a' \Delta t}{a'+b'} \mathbf{\Omega}_{1t} \, \mathbb{S} \, \mathbf{\Omega}_{t1} \frac{\partial \mathbf{G}}{\partial \mathbf{u}_n} \Big) \mathbf{u}_{n+1} = \\
&\bar{\mathbf{u}}_0 \quad - \sum_i \sum_j \mathbf{T}_{ij} \left(\mathbf{F}_{(n)ij} - \frac{a'}{a'+b'} \frac{\partial \mathbf{F}_{ij}}{\partial \mathbf{u}_n} \mathbf{u}_n \right) \\
&+ \quad (\Delta t \mathbf{\Omega}_{1t} \, \mathbb{S} \, \mathbf{\Omega}_{t1}) \left(\mathbf{G}_n - \frac{a'}{a'+b'} \frac{\partial \mathbf{G}_n}{\partial \mathbf{u}_n} \mathbf{u}_n \right) + \mathcal{O}\left(\Delta t^p, \Delta {x_1}^{p_1}, \Delta {x_2}^{p_2}, \ldots, \Delta {x_{z-1}}^{p_{z-1}} \right)
\end{aligned} \tag{4.47}$$

The above huge nonlinear system of iterative equations constitutes arbitrary order accurate CFL-independent [8] DPI method for general nonlinear PDE (4.32) with arbitrary IC/BC. Equation (4.47) is solved for $\mathbf{u}_{n+1}$ iteratively until it converges to the Picard convergence tolerance defined in eq.(3.51) in Sec.(3.4.1).

Numerical Closed-Form solution of General Linear conservative PDES

We notice that for linear flux $\mathbf{F}_{ij}$ and linear source term $\mathbf{G}$, we can find a *Numerical Closed-Form Solution for Arbitrary Conservative PDE* given in (4.33) with *arbitrary IC/BC* represented by column vectors $\mathbf{R}_i = \mathbf{R}(\mathbf{u}_0)$ and $\mathbf{b}_{ij}$ respectively. For linear flux and source term, we can write

$$\begin{aligned}
\mathbf{F}_{ij} &= \mathbb{W}_{ij} \mathbf{u} \\
\mathbf{G} &= \mathbb{Q} \mathbf{u}
\end{aligned} \tag{4.48}$$

and therefore the Jacobians are

$$\begin{aligned}
\frac{\partial \mathbf{F}_{ij}}{\partial \mathbf{u}} &= \mathbb{W}_{ij} \\
\frac{\partial \mathbf{G}}{\partial \mathbf{u}} &= \mathbb{Q}
\end{aligned} \tag{4.49}$$

Substituting (4.48) and (4.49) into (4.47) and choosing $a' = 1$ $b' = 0$ (fully implicit) we will have,

$$\begin{aligned}
&\Big(\quad \mathbb{I} \quad + \sum_i \sum_j \mathbf{T}_{ij} \mathbb{W}_{ij} - \Delta t \mathbf{\Omega}_{1t} \, \mathbb{S} \, \mathbf{\Omega}_{t1} \mathbb{Q} \Big) \mathbf{u} = \\
&\bar{\mathbf{u}}_0 \quad + \mathcal{O}\left(\Delta t^p, \Delta {x_1}^{p_1}, \Delta {x_2}^{p_2}, \ldots, \Delta {x_{z-1}}^{p_{z-1}} \right)
\end{aligned}$$

[8] Off course it is CFL-dependent but we can control parameter a' and b' to make it CFL independent. For a through stability analysis please refer to Sec.(3.2).

or

$$\mathbf{u} = \left(\mathbb{I} + \sum_i \sum_j \mathbf{T}_{ij}\mathbb{W}_{ij} - \Delta t \mathbf{\Omega}_{1t}\,\mathbb{S}\,\mathbf{\Omega}_{t1}\mathbb{Q}\right)^{-1} \bar{\mathbf{u}}_0 + \mathcal{O}\left(\Delta t^p, \Delta {x_1}^{p_1}, \Delta {x_2}^{p_2}, \ldots, \Delta {x_{z-1}}^{p_{z-1}}\right) \tag{4.50}$$

The above result is so general that it deserves to be studied in detail in a separate report. Here are the observations made regarding the numerical closed-form solution to arbitrary time-evolutionary PDE with arbitrary IC/BC given in (4.50).

- For a linear PDE, the numerical closed-form solution $\mathbf{u}$ is a *linear* combination of IC/BC column vector $\bar{\mathbf{u}}_0$ and truncation error. In fact when we look at (4.50), we find that the terms inside the big parenthesis are combination of the total operator $\mathbf{T}_{ij}$, integration operator $\mathbb{S}$ and constant Jacobians $\mathbb{W}_{ij}$ and $\mathbb{Q}$ which are constant over time interval Δt. Therefore, $\mathbf{u}$ is written as a constant matrix multiplication of operator matrices (terms inside parentheses) with $\bar{\mathbf{u}}_0$ plus the truncation error showing that the dependency is always linear, i.e. $\mathbb{A}\bar{\mathbf{u}}_0 + \mathbf{b}$.
- The analytical closed-form solution can be retrieved by taking the limit of expression $(\Delta t, \Delta x_1, \Delta x_2, \ldots, \Delta x_{z-1}) \to 0$. In this case we readily observe that the analytical solution proposed in (4.50) doesn't have linear drift due to the truncation error.

In the next section, we solve special cases of arbitrary nonlinear conservative PDEs (4.32) by running a *single* piece of code containing an implementation of eq.(4.47).

4.3.2 Test case I: Multidimensional Periodic Convection

For $i = 1$, $j = 1 \ldots 2$, $\mathbf{G} = 0$, eq.(4.32) give us three-dimensional convection[9]

$$\frac{\partial \mathbf{u}}{\partial t} + \frac{\partial \mathbf{F}_1}{\partial x} + \frac{\partial \mathbf{F}_2}{\partial y} = 0, \tag{4.51}$$

Choosing a linear flux function

$$\mathbf{F}_1 = \mathbf{F}_2 = c\mathbf{u} \tag{4.52}$$

[9]This is literally called two-dimensional because the equation has two spatial dimensions. However since we solve equations using nested column format (4.2) which comprehends all dimensions simultaneously, we call it three-dimensional.

and assuming that the number of equations is one, i.e. $y = 1$ we will have,

$$\begin{aligned} & \frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} + c\frac{\partial u}{\partial y} = 0, \\ \rightarrow \quad & \frac{\partial u}{\partial t} = R\left(x, y\right) = -c\frac{\partial u}{\partial x} + -c\frac{\partial u}{\partial y} = -c\nabla.\vec{u}, \end{aligned} \tag{4.53}$$

Here we assume that boundary conditions in two space dimensions 'x' and 'y' are periodic. Thus vectors $\mathbf{b}_{11}$ and $\mathbf{b}_{12}$ are zero in (4.46). Also $\mathbf{G}_n$ and its derivative are zero in (4.47). Using (4.46) in this case and neglecting truncation error, the generic implicit DPI (4.47) simplifies to

$$\begin{aligned} & \Big(\ \mathbb{I} \ + \frac{a'}{a'+b'} \sum_{i=1}^{1} \sum_{j=1}^{2} \mathbf{T}_{ij} \frac{\partial \mathbf{F}_{ij}}{\partial \mathbf{u}_n} \Big) \mathbf{u}_{n+1} = \\ & \mathbf{u}_0 + \Delta t \mathbf{\Omega}_{1t} \, \mathbb{S}_i \, \mathbf{\Omega}_{t1} \mathbf{R}_i - \sum_{i=1}^{1} \sum_{j=1}^{2} \mathbf{T}_{ij} \left(\mathbf{F}_{(n)ij} - \frac{a'}{a'+b'} \frac{\partial \mathbf{F}_{ij}}{\partial \mathbf{u}_n} \mathbf{u}_n \right) \end{aligned} \tag{4.54}$$

We remember from Postulate (I) in page (62) that we need to calculate spatial operator accurately to achieve the dissipation-free advantage of implicit DPI. Therefore, we use the fully spectral differentiation matrix presented in eq(4.13). In addition, we always prefer to solve PDEs and ODEs like (4.54) in a unified residual-based (4.36) which is based on the original implicit DPI (3.15). Thus by replacing $\mathbf{T}_{ij}$ from (4.41) into (4.54) and rearranging both sides, we can rewrite (4.54) in the residual-based equation below.

$$\begin{aligned} & \Big(\mathbb{I} \ - \ \frac{a'\Delta t}{a'+b'} \mathbf{\Omega}_{1t} \, \mathbb{S} \, \mathbf{\Omega}_{t1} \frac{\partial \mathbf{R}}{\partial \mathbf{u}_n} \Big) \mathbf{u}_{n+1} = \\ & \mathbf{u}_0 \ + \ \Delta t \mathbf{\Omega}_{1t} \, \mathbb{S} \, \mathbf{\Omega}_{t1} \left(\mathbf{R}_n - \frac{a'}{a'+b'} \frac{\partial \mathbf{R}}{\partial \mathbf{u}_n} \mathbf{u}_n \right) + \Delta t \mathbf{\Omega}_{1t} \, \mathbb{S}_i \, \mathbf{\Omega}_{t1} \mathbf{R}_i \end{aligned} \tag{4.55}$$

where

$$\mathbf{R}_n = -c \left[\nabla\right] \mathbf{u}_n = -c \left(\mathbb{D}_x + \mathbf{\Omega}_{12} \mathbb{D}_y \mathbf{\Omega}_{21}\right) \mathbf{u}_n \tag{4.56}$$

and

$$\frac{\partial \mathbf{R}}{\partial \mathbf{u}_n} = -c \left[\nabla\right] \tag{4.57}$$

are the numerical residual and numerical Jacobian of residuals respectively. Therefore to implement the two-dimensional periodic convection using implicit DPI we do the following steps.

1. **calculate** directional derivative matrices $\mathbb{D}_x = \mathbb{D}_1$ and $\mathbb{D}_y = \mathbb{D}_2$ using Listing (4.3) and appropriate Kronecker product.
2. **calculate** rearrangement operators $\mathbf{\Omega}_{21}$ and $\mathbf{\Omega}_{t1}$ required in eq.(4.56) and eq.(4.55) using Listing (4.1).
3. **calculate** numerical divergence matrix $[\nabla]$ required for both eq.(4.56) and eq.(4.57) using operators obtained in the previous step.
4. **calculate** the initial conditions matrices $\mathbb{S}_i\, \mathbf{\Omega}_{t1}\mathbf{R}_i$ on rhs of (4.55).
5. **initialize** the vector of initial values $\mathbf{u}_0$ using given ICs conditions in problem.
6. **for** temporal steps 'k' **do**
   (a) **while** Picard convergence is not satisfied **do**
      i. **iterate** over 'n' in generic equation (4.55)
   (b) **end**
   (c) **set** the last *spatial* value $\mathbf{u}_n$ at time t as the initial condition $\mathbf{u}_0$ for the next temporal step.
7. **perform** post processing and compare the result with the appropriately shifted analytical solution

We implemented the above algorithm in a very short Matlab m-file given in Listing (4.6). The code is extremely easy and well-commented. A main program which evaluates the DPI solution of the three-dimensional problem (4.53) using Listing (4.6) is brought in Listing (4.10). The main program first calculates appropriate time step Δt, wave speed c and other constants. As shown a coarse $N_1 = 10$, $N_2 = 10$, $N_3 = N_t = 140$ grid is initialized. Then it initializes the filed with a sinusoidal function $u(x, y, t) = \sin(x)$ over one period $[x, y] = [0, 2\pi]$. The number of points in time is 140 points and with chosen $CFL = 10$ this gives the time step exactly equal to the similar test case performed in (Wang (2009)). The implicit DPI solution is marched in time for twenty and fifty complete cycles (periods). The final solution is compared with the initial waveform which is an analytical sinusoidal wave with appropriate shift due to number of selected temporal nodes.

Listing 4.6: This function implements implicit DPI for generic space-time z-dimensional equations in nested column format (4.55). Since it is applicable to both ODEs and PDEs in the same time, we used the name ode-implicit-DPI without any difference. This program uses residual and Jacobian of

the residual obtained using Listings (4.7) and (4.8) respectively. Also for time integration operator $\mathbb{S}$ it uses Listing (4.9).

```
function [out1 out2 out3 out4] = ode_implicit_dpi(ff,jacob_ff,N,y,dx,u0,e_pic,a,b)
%Solving ODE using implicit Picard Iteration schemes
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%% INPUT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%ff = @handle to ode function similar to ode45
%jacob_ff = @handle to jacobian d/du of the ode function ff
%N = [N1 N2 ... Nz] number of points in each dimension
%y = number of equations in the system of PDE/ODE
%dx = [dx1 dx2 ... dxz] the spacing for each dimension
%u0 = initial value
%e_pic = DPI convergence bound
%a,b = the stability-convergence constants in the theory
%%%%%%%%%%%%%%%% OUTPUT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%out1 = solution over the given time interval
%out2 = the time interval
%out3 = residuals history matrix RES_HIST(:,1) = ITR, RES_HIST(:,2)  = res.
%out4 = computation time (in sec) required for DPI to converge to e_pic

%global vars to increase efficiency
global Div
global disc_method

%%%%%%%%%%%%%%%Adapting notation with previous code
%number of temporal points
nS = N(3);

%△_t = the physical (actual) time interval that solution is evaluated
%on.
△_t = (1+1/(N(3))) * dx(end);

%
u0 = mat2cell(u0,N(1)*N(2)*ones(N(3),1),1);
U0 = u0{1};
u0 = cell2mat(u0);

%constructing integration operator of arbitrary-order accurate
disp('Constructing integration operator of arbitrary-order accurate');
[S ri] =S_Newton_Cotes(nS,y); %making the operator
%mapping constructed one-D operator to general multi-dimensional space
S = kron(sparse(1:(N(1)*N(2)),1:(N(1)*N(2)),1),S);
disp('done!');
%Precomputing all required operators
disp('Constructing OMEGA_21');
OMEGA_21 = OMEGA_x1([N(1) N(2) nS],y,2);
disp('Constructing OMEGA_31');
OMEGA_31 = OMEGA_x1([N(1) N(2) nS],y,3);
disp('Constructing D1');
D1 = gen_1D_diff(N(1),y,dx(1),disc_method);
disp('Constructing D2');
D2 = gen_1D_diff(N(2),y,dx(2),disc_method);

disp('Assembling Divergence operator');
Div = kron(sparse(1:(N(2)*nS),1:(N(2)*nS),1),D1) + ...
```

```
    (OMEGA_21)\(kron(sparse(1:(N(1)*nS),1:(N(1)*nS),1),D2)*OMEGA_21);
disp('Constructing OMEGA_S_OMEGA');
Omega_S_Omega = OMEGA_31\(S*OMEGA_31);

%building the time interval
tT = ri*△_t;

%Step 1 allocating and initializing the ResBuffer for each residual whose
%dimension in time is equal to
%nS. Since we only have one residual vector we need only one buffer.
ResBuffer = ff(tT,u0);
%allocating solution buffer. Since we have one scalar unknow, we only need
%one buffer.
uBuffer = u0;

%computing S0
Res0 = cell(nS,1);
for i =1: nS
    Res0{i,1} = U0;
end
Res0 = ff(tT,cell2mat(Res0));
S0 = S(1,1)*OMEGA_31*Res0;

%initializing the iteration index
ITR = 1;

DU = uBuffer; %temp init
DUn = 0.*DU;
tic %begin the timer
while(max(norm(DU-DUn)) >e_pic)

    %computing the jacobian diagonal matrix dR_du
    dR_du = jacob_ff(tT, uBuffer);

    %
    DUn = DU;
    %SUB STEP I: Distributing residuals all over the buffer
    %evaluating the residuals
    ResBuffer = ff(tT,uBuffer);
    %SUB STEP II: updating the solution buffer using implicit method
    %Omega_S_Omega = OMEGA_31\(S*OMEGA_31); (go to top of the main loop)

    A_tot = sparse(1:(N(1)*N(2)*nS*y),1:(N(1)*N(2)*nS*y),1) - ...
        a*△_t/(a+b)*Omega_S_Omega*dR_du;
    b_tot = u0+△_t*Omega_S_Omega*(ResBuffer- a/(a+b)*dR_du*uBuffer) + △_t * ...
        OMEGA_31\S0;

    uBuffer_temp = A_tot\b_tot;
    DU = uBuffer_temp - uBuffer;
    uBuffer = uBuffer_temp;

    %Filling residuals history matrix and updating iteration index
    RES_HIST(ITR,1) = ITR;
    RES_HIST(ITR,2) = max(norm(DU-DUn));
    ITR = ITR + 1;
    max(norm(DU-DUn)) %printing residuals
```

```
107 end
108 Comp_Time = toc;
109
110 out1 = uBuffer; %returning the solution
111 out2 = tT; %returning the time interval
112 out3 = RES_HIST; %returning residuals
113 out4 = Comp_Time; %returning computaton time.
114 disp('ode_impplicit_dpi completed');
115
116 %end
```

Listing 4.7: The residual in eq.(4.53) calculated using numerically discretized Divergence operator. Note that boundary conditions vector is zero since the condition of periodicity is already implemented in differentiation matrix eq.(4.13). Also since $c = -1$ in the main program, this residual is compatible with eq.(4.53) for $c = 1$ (left to right going wave).

```
1 function du = anl_model(t,u)
2 %t = the time vector
3 %u = the nested z-dimensional solution column
4 global c
5 global Div
6
7 bc = 0.*u;
8 du = c*Div*(u) + bc;
```

Listing 4.8: The Jacobian obtained in operator-wise approach.

```
1 %the Jacobian model
2 function out = anl_model_jacob(t,u)
3 %the output Jacobian
4 global c
5 global Div
6
7 out = c*Div;
```

Listing 4.9: This function returns the second-order integration matrix based on Newton-Cotes formula.

```
1 function [out1 out2] =S_Newton_Cotes(nS,y)
2
3 %CONSTANTS ARE SPECIFIED HERE.
4 %The number of points in the quadrature scheme
5 S = cell(nS,nS); %The final operator
6 for i = 1:nS
7     for j=1:nS
8         S{i,j} = sparse(zeros(y));
9     end
10 end
11
12 h = 1./(nS);
13 ri = (1:nS)*h;
```

```

S{1,1} = sparse(.5*eye(y));
%S{2,1} = sparse(1.*eye(y)); S{2,2} = sparse(.5*eye(y));
for i=2:nS
    %S{i,1} = sparse(.5*eye(y));
    for j=1:(i-1)
        S{i,j} = sparse(1.*eye(y));
    end
    S{i,i} = sparse(.5*eye(y));
end

out1 = cell2mat(S)*h;
%out1 = cell2mat(S);
out2 = ri;
```

Listing 4.10: The main program. Computes solution to (4.53) using implicit DPI given in Listing (4.6) and Euler explicit method and exports the results to Tecplot format.

```
%%%%%%%%%%%%%%%%%%%%% MAIN %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Preparing the MATLAB environment.
clear all;
clc

%defininf global variables on need
global c
global disc_method

%CONSTANTS ARE SPECIFIED HERE.
%stability-convergence constants
a = 1.2;
b = .01;
e_pic = 1.e-6;
y=1;
c=-1;
N = [10 10 140]; %The size of the space-time (N1 N2 N3)
disc_method = 2; %spectral

%two-dimensional spatial ranges
x_min = -pi;
x_max = pi;
y_min = -pi;
y_max = pi;

%spatial grid spacing
dx = (x_max - x_min)/N(1);
dy = (y_max - y_min)/N(2);

%creating a dummy field
x_x= x_min + (1:N(1))*dx;
y_y= y_min + (1:N(2))*dy;

for l=1:N(3)
```

```
    for j=1:N(2)
        for i=1:N(1)
             u(i,1) = sin(x_x(i));
        end
        U(j,1) = {sparse(u)};
    end
    UU(l,1) = {cell2mat(U)};
end

u0 = cell2mat(UU);

%CFL condition- change this CFL to the one you desire.
CFL = 10;
%computing the time interval based on CFL condition for convection-only.
△_t = CFL/abs(c)*min(dx,dy);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Solving ODE using implicit Picard Iteration scheme
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
NDPI_MARCH = 50; %for fifty periods
u0_orig = u0; %saving the original initial condition for future use in
%explicit scheme and other verifications
for i=1:NDPI_MARCH
    [u_DPI_implicit time_implicit Res_implicit Comp_Time_Implicit] = ...
        ode_implicit_dpi(@anl_model,@anl_model_jacob,N,...
        y,[dx dy △_t],u0,e_pic,a,b);

    %%%%%%%%%%%%%%%%%%% WINDOWING %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %capturing the last spatial solution as the next initial condition
    u0 = mat2cell(u_DPI_implicit,N(1)*N(2)*ones(N(3),1),1);
    for j = 1:(N(3)-1)
        u0(j) = u0(N(3));
    end
    u0 = cell2mat(u0);
    max(max(abs(u0))) %printing amplification % if increase the marching
    %will diverge.
end

u0 = u0_orig; %turning back u0 to its original state.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Solving ODE using Euler explicit algorithm
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[u_explicit Numb_explicit Comp_Time_explicit] = ...
    pde_euler_explicit(@anl_model,N,y,[dx dy △_t],u0,.0001);

%converting u_DPI_implicit from sparse form to full form
u_DPI_implicit = full(u_DPI_implicit);
%exporting to tecplot
u_DPI_implicit = mat2cell(u_DPI_implicit,N(1)*N(2)*ones(N(3),1),1);

for i=1:N(3)
    u_DPI_implicit{i} = mat2cell(u_DPI_implicit{i},N(1)*ones(N(2),1),1);
end

fid = fopen('u.tec','w');
```

```
fprintf(fid,'VARIABLES = "X", "Y", "T","u"\n');
fprintf(fid,'ZONE T="Num", I=%d, J=%d, K=%d F=POINT\n',N(1),N(2),N(3));

for l=1:N(3)
    tempU = u_DPI_implicit{l};
    for j=1:N(2)
        tempUU = tempU{j};
        for i=1:N(1)
            fprintf(fid,'%e %e %e %e\n',x_x(i),y_y(j),time_implicit(l),tempUU(i));
        end
    end
end

fclose(fid);

%plotting inline in MATLAB
profiles = zeros(N(1),1);
for l=[1 N(3)]
    tempU = u_DPI_implicit{l};
    for j=1:1
        tempUU = tempU{j};
        profiles(:,l) = tempUU;
        hold on
    end
end
x_shifted = x_x - (NDPI_MARCH-1)*pi/N(3);
plot(x_shifted,profiles(:,N(3)),'d');
hold on;
plot(x_shifted,sin(x_shifted),'.-');
plot(x_x,sin(x_x),'-');

%converting u_explicit from sparse form to full form
u_explicit = full(u_explicit);

%exporting to tecplot
u_explicit = mat2cell(u_explicit,N(1)*N(2)*ones(N(3),1),1);

for i=1:N(3)
    u_explicit{i} = mat2cell(u_explicit{i},N(1)*ones(N(2),1),1);
end

fid = fopen('uexpl.tec','w');
fprintf(fid,'VARIABLES = "X", "Y", "T","uexpl"\n');
fprintf(fid,'ZONE T="explicit", I=%d, J=%d, K=%d F=POINT\n',N(1),N(2),N(3));

for l=1:N(3)
    tempU = u_explicit{l};
    tempV = u_DPI_implicit{l};
    for j=1:N(2)
        tempUU = tempU{j};
        for i=1:N(1)
            fprintf(fid,'%e %e %e %e\n',x_x(i),y_y(j),time_implicit(l),tempUU(i));
        end
    end
end
fclose(fid);
```

```
149
150 %end of MAIN
151 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figure (4.3.1) shows the three-dimensional characteristic lines for solution obtained after fifty periods ($t = 50$). As shown the lines have 45 degree slope and the solution retrieves its initial value after 50 periods. The computational grid is also shown at the bottom.

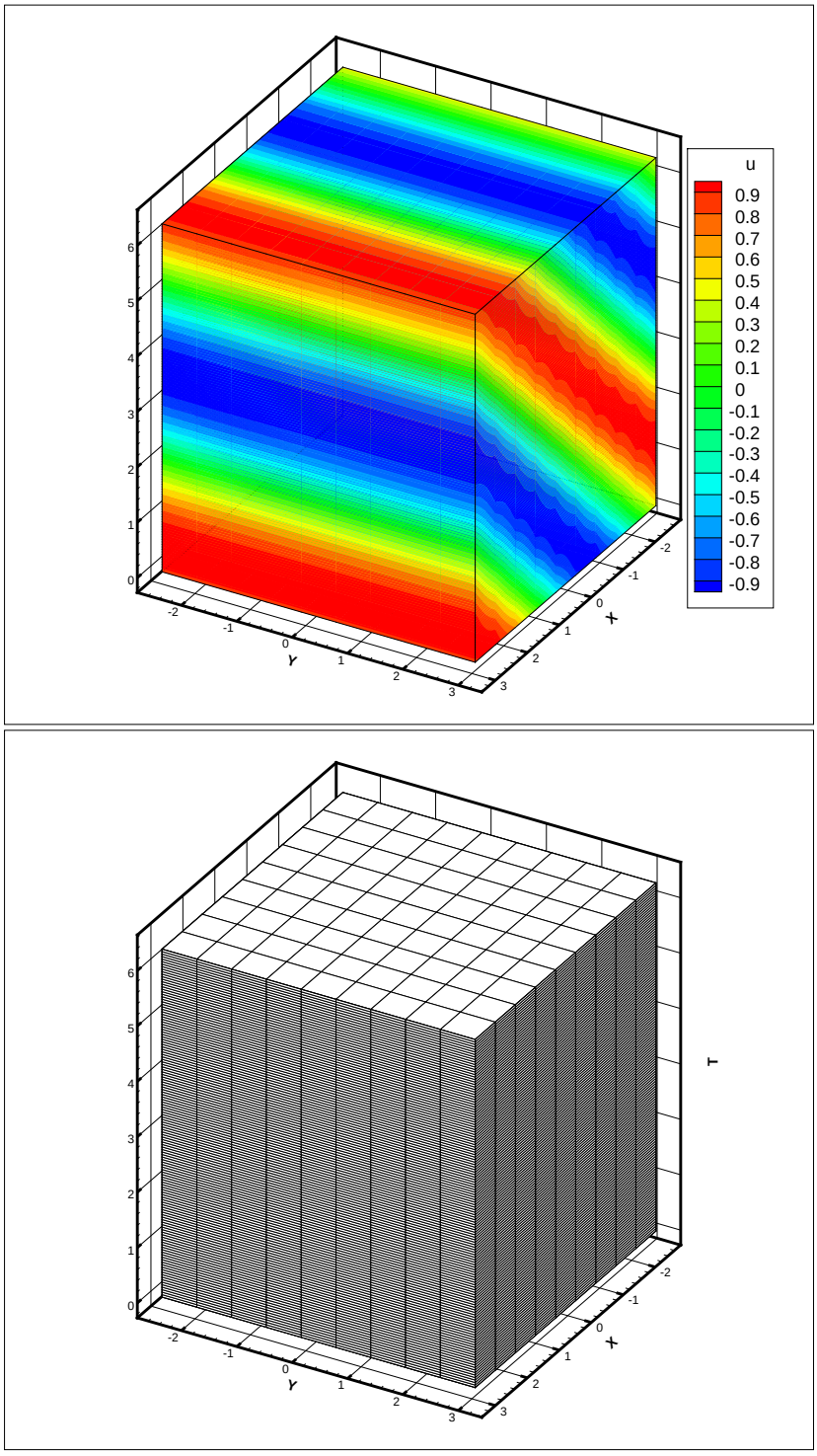
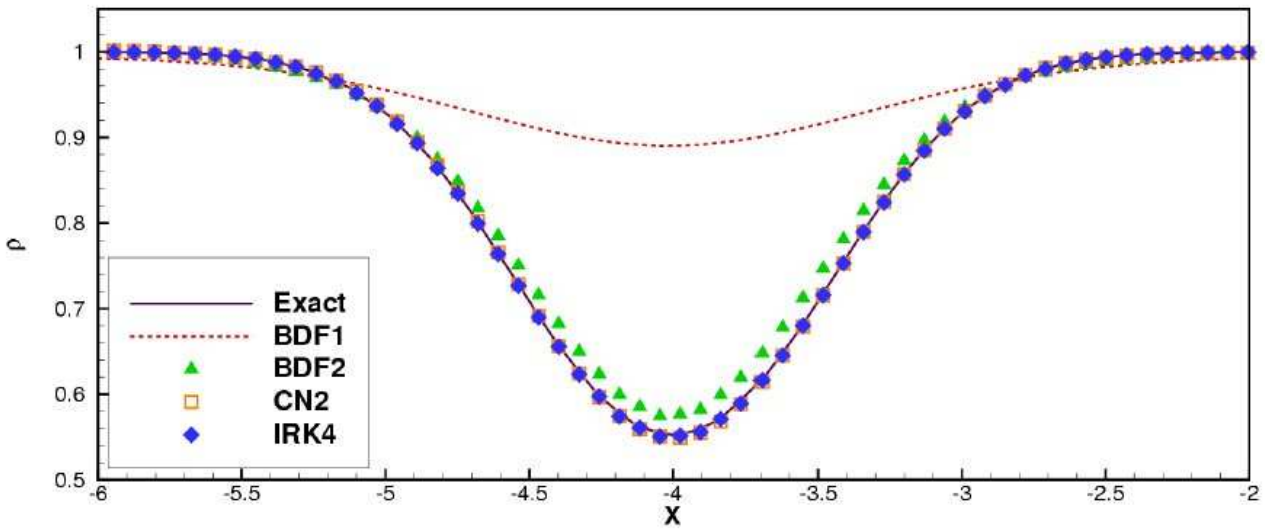


Figure 4.3.1: Top) Solution contours after fifty periods. Bottom) The grid used to obtain the solution.

The solution profile after t=20 and t=50 are shown in fig.(4.3.2) and (4.3.3) respectively. As shown, the second-order implicit method CN2 and BDF2 have acceptable resolution at t=20 while at t=50 we see that they have major dispersion and dissipation. The second-order implicit DPI scheme however have excellent accuracy even

for fifty periods. As we expect from Postulate (I), there is no dissipation error for DPI schemes.



(c) t=20

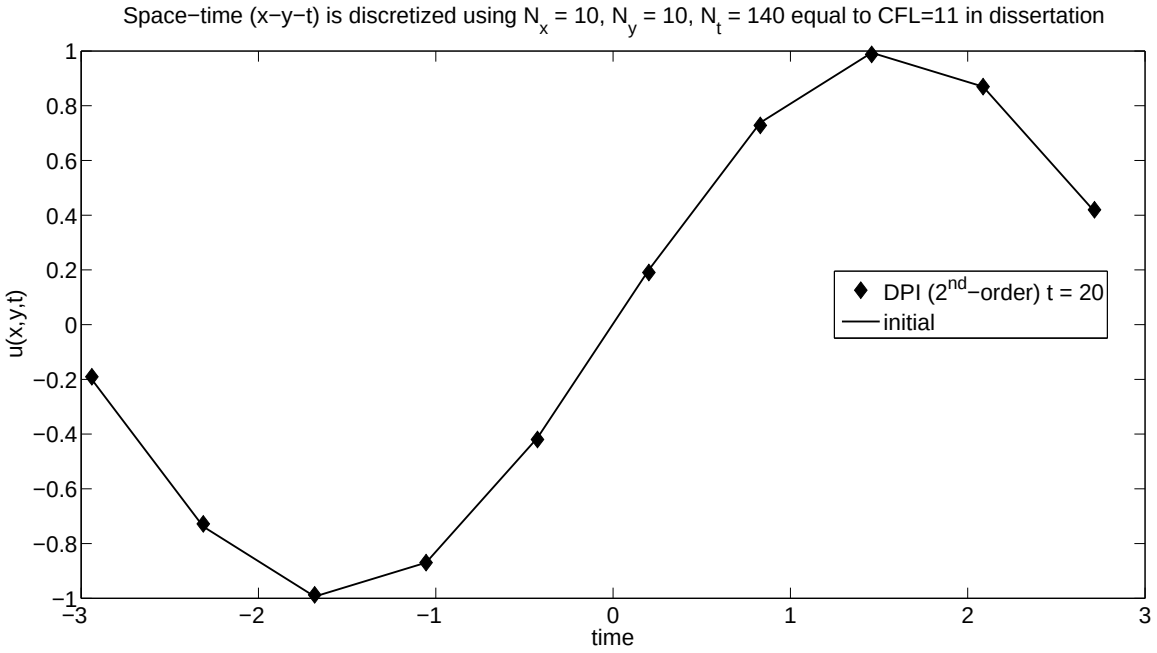


Figure 4.3.2: Solution profiles after twenty periods t=20, Top) problem solved in (Wang (2009)) with initial Gaussian wave and with different methods including second-order BDF2 and CN2 and fourth-order implicit RK4. Bottom) Current solution with sinusoidal initial condition using second order DPI.

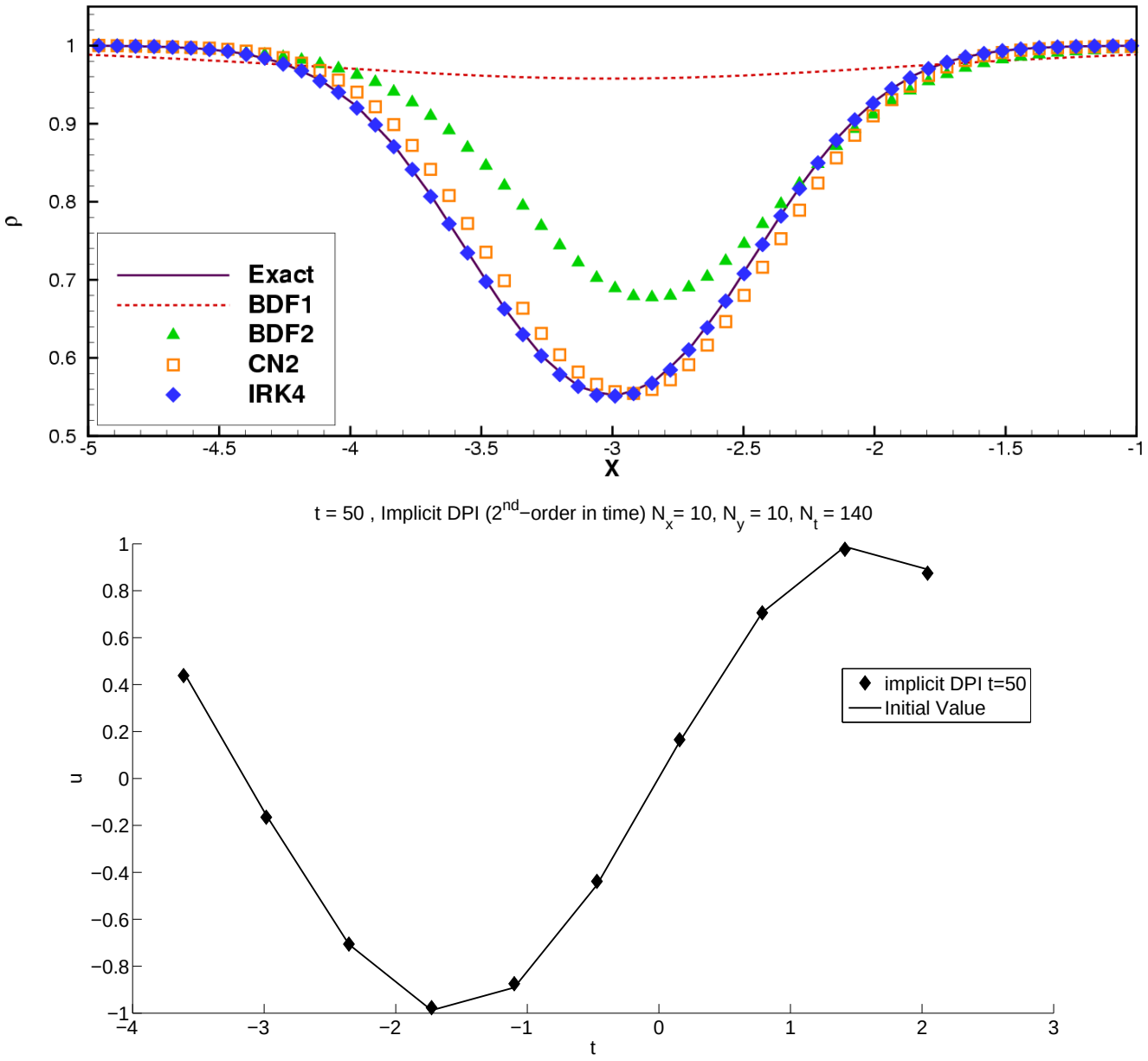


Figure 4.3.3: Solution profiles after fifty periods t=50, Top) problem solved in (Wang (2009)) with initial Gaussian wave and with different methods including second-order BDF2 and CN2 and fourth-order implicit RK4. Bottom) Current solution with sinusoidal initial condition using second order DPI.

4.3.3 Test case II: Nonlinear Wave Propagation

In this section, we replace the linear flux vector of (4.58) with nonlinear Burgers terms

$$\mathbf{F}_1 = \mathbf{F}_2 = c\mathbf{u}^2 \tag{4.58}$$

Also due to formation of shocks, we need to use upwind scheme. Thus the only modification that we do to the code listings presented in Sec.(4.3.2) is to replace the flux function in the residual (Listing (4.7)) and Jacobian of residual (Listing (4.8)) with the nonlinear polynomial (4.58) and then replacing spectral differentiation matrix in Listing (4.3) with upwind scheme presented in Listing (4.2). The solution obtained in this manner is therefore first-order due to the presence of first-order upwind scheme. The results are presented in fig.(4.3.4) for two different values of CFL = 0.4 and CFL = 4,000,000. As shown, for small CFL we see the traveling wave is leaving the domain. It should be noted that only three points in the time direction is selected and the spatial scheme is first-order upwind. Therefore we don't expect that results presented in (4.3.4) demonstrate spectacular accuracy as was shown in fig.(4.3.2) for spectral schemes. Therefore our goal here is to investigate whether the partial sum in implicit DPI converges in nonlinear equations or not. Figure (4.3.4) clearly shows that implicit DPI remains stable for small and large CFLs for multidimensional nonlinear equations.

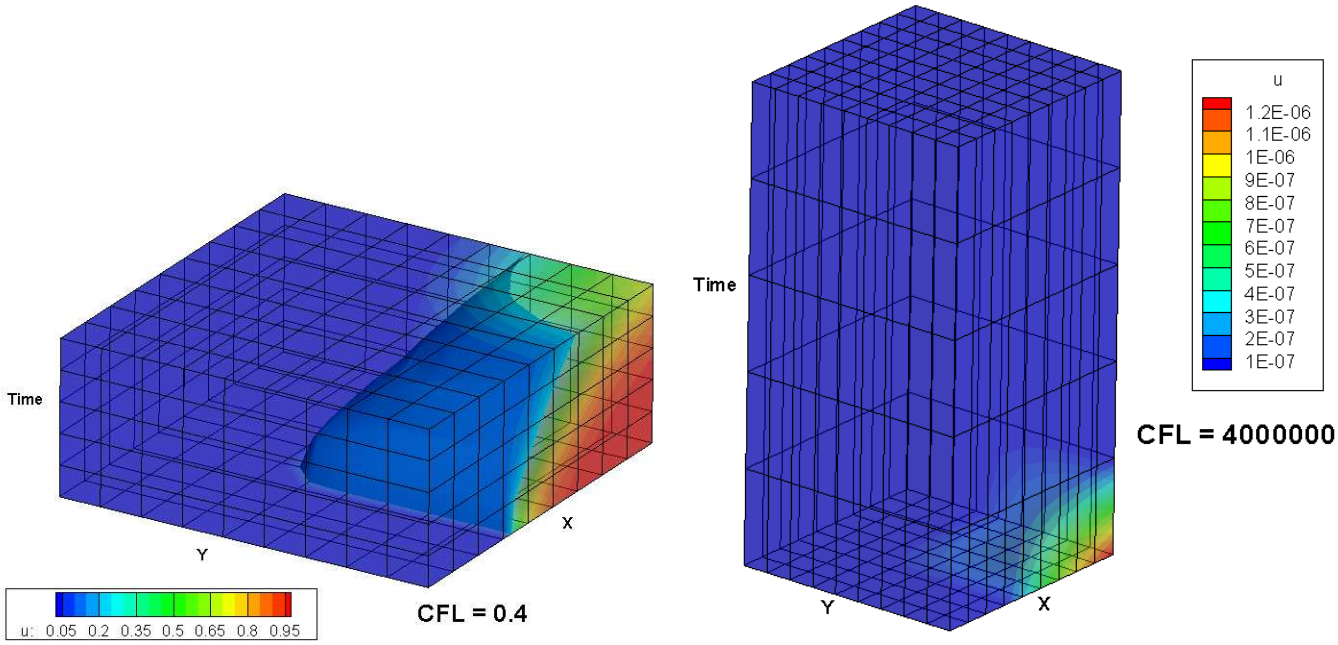


Figure 4.3.4: Solution of three-dimensional Burgers equation for Left) CFL = .4 and Right) CFL = four millions.

Chapter 5

General Unstructured Formulation and Conservation Laws

5.1 Finite Volume Formulation

Let us consider the general time-evolutionary conservative differential equation below

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla.\mathbf{F} = 0, \tag{5.1}$$

where $\mathbf{u} = [u_1, u_2, \ldots, u_y]$ is the vector of conservative variables and $\mathbf{F}$ is the tensor of fluxes. Using finite-volume method we take volume integral of both sides of (5.1) over a finite control volume T in space as follows,

$$\int_T \frac{\partial \mathbf{u}}{\partial t} dV_T + \int_T \nabla.\mathbf{F}\ dV_T = 0, \tag{5.2}$$

We note that using mean-value theorem $\int_T \frac{\partial \mathbf{u}}{\partial t}\, dV_T = \frac{\partial \bar{\mathbf{u}}}{\partial t} V_T$ where $\frac{\partial \bar{\mathbf{u}}}{\partial t}$ is the mean value of time derivative of the solution at centroid of the control volume T. Also applying Gauss-Divergence theorem $\int_T \nabla.\mathbf{F}\ dV_T = \int_{S_T} \mathbf{F}.d\mathbf{S}_T$, eq. (5.2) can be written as

$$\int_T \frac{\partial \mathbf{u}}{\partial t} dV_T + \int_T \nabla.\mathbf{F}\ dV_T = \frac{\partial \bar{\mathbf{u}}}{\partial t} V_T + \int_{S_T} \mathbf{F}.d\mathbf{S}_T = 0, \tag{5.3}$$

Discretizing the surface integral in (5.3) over all faces enclosing the control volume, we obtain,

$$\begin{gathered} V_i \frac{d}{dt}\left(\bar{\mathbf{u}}_i\right) + \sum_j \mathbf{F^*}_{ij}\left(\bar{\mathbf{u}}\right).\Delta\mathbf{S}_{ij} = 0, \\ i = 1 \ldots N_{\text{nodes}}, j = 1 \ldots N_{\text{edge}(i)}, \end{gathered} \tag{5.4}$$

Where $\Delta\mathbf{S}_{ij}$ is the surface normal vector of the j^{th} face wrapping around volume T (umbrella formed over j^{th} edge of node 'i') and $\mathbf{F}^*$ is a tensor which contains the reconstruction of fluxes over the faces enclosing the control volume T. The tensor product $\mathbf{F^*}_{ij}(\bar{\mathbf{u}}).\Delta\mathbf{S}_{ij}$ is actually summed over column fluxes in all spatial directions, i.e. for three-dimensional problems we have

$$\mathbf{F}^*.\Delta\mathbf{S} = \left|\mathbf{f}_x^*, \mathbf{f}_y^*, \mathbf{f}_z^*\right| . \left|\Delta\mathbf{s}_x, \Delta\mathbf{s}_y, \Delta\mathbf{s}_z\right| = \mathbf{f}_x^*\Delta\mathbf{s}_x + \mathbf{f}_y^*\Delta\mathbf{s}_y + \mathbf{f}_z^*\Delta\mathbf{s}_z = \sum_{k=1}^{3} \mathbf{f}_k^*\Delta\mathbf{s}_k \quad (5.5)$$

The index 'i' in eq.(5.4) loops over the entire nodes/cells (depending on node-centered and cell-centered formulation) and the index 'j' loops over the surrounding edges/cells of 'i' entity. The edges connecting to node 'i' are usually divided into two categories of *interior* edges which connect point 'i' to another **real** point 'j' in the grid and *ghost* edges which connect point 'i' to an imaginary point which is usually used to impose boundary condition through an indirect method. It is more convenient to separate boundary (ghost) edges from interior edges in (5.4) resulting

$$\begin{aligned} V_i\frac{d}{dt}(\bar{\mathbf{u}}_i) \;+\; & \sum_{j}^{\text{interior}} \mathbf{F^*}_{ij}(\bar{\mathbf{u}}).\Delta\mathbf{S}_{ij} \\ + \; & \sum_{j}^{\text{ghost}} \mathbf{F^*}_{ij}(\bar{\mathbf{u}}).\Delta\mathbf{S}_{ij} = 0 \end{aligned} \quad (5.6)$$

The reconstructed flux tensor $\mathbf{F}^*$ is usually obtained in the terms of the original flux $\mathbf{F}$ using appropriate Reimann-Solver. For example, using Roe flux vector differencing we have,

$$\mathbf{F}^*(\bar{\mathbf{u}}) = \frac{1}{2}\left(\mathbf{F}(\bar{\mathbf{u}}_i) + \mathbf{F}(\bar{\mathbf{u}}_j) - \left|\tilde{\mathbb{A}}_{ij}\right|(\bar{\mathbf{u}}_j - \bar{\mathbf{u}}_i)\right) \quad (5.7)$$

Substituting (5.7) into (5.6) and using flux notation (5.5) we obtain

$$\begin{aligned} V_i\frac{d}{dt}(\bar{\mathbf{u}}_i) \;+\; & \frac{1}{2}\sum_{k=1}^{3}\sum_{j}^{\text{interior}} \left[\Delta\mathbf{s}_{kij}\left(\mathbf{f}_k(\bar{\mathbf{u}}_i) + \mathbf{f}_k(\bar{\mathbf{u}}_j)\right) - \Delta\mathbf{s}_{kij}\left|\tilde{\mathbb{A}}_{kij}\right|(\bar{\mathbf{u}}_j - \bar{\mathbf{u}}_i)\right] \\ + \; & \frac{1}{2}\sum_{k=1}^{3}\sum_{j}^{\text{ghost}} \left[\Delta\mathbf{s}_{kij}\left(\mathbf{f}_k(\bar{\mathbf{u}}_i) + \mathbf{f}_k(\bar{\mathbf{u}}_j)\right) - \Delta\mathbf{s}_{kij}\left|\tilde{\mathbb{A}}_{kij}\right|(\bar{\mathbf{u}}_j - \bar{\mathbf{u}}_i)\right] = 0 \end{aligned} \quad (5.8)$$

To increase readability, we represent the ghost values with a over hat

$$V_i \frac{d}{dt}\left(\bar{\mathbf{u}}_i\right) \;+\; \frac{1}{2}\sum_{k=1}^{3}\sum_{j}^{\text{interior}} \left[\Delta \mathbf{s}_{kij}\left(\mathbf{f}_k\left(\bar{\mathbf{u}}_i\right)+\mathbf{f}_k\left(\bar{\mathbf{u}}_j\right)\right) - \Delta \mathbf{s}_{kij}\left|\tilde{\mathbb{A}}_{kij}\right|\left(\bar{\mathbf{u}}_j-\bar{\mathbf{u}}_i\right)\right]$$
$$+\; \frac{1}{2}\sum_{k=1}^{3}\sum_{j}^{\text{ghost}} \left[\Delta \hat{\mathbf{s}}_{kij}\left(\mathbf{f}_k\left(\bar{\mathbf{u}}_i\right)+\hat{\mathbf{f}}_k\left(\bar{\mathbf{u}}_j\right)\right) - \Delta \hat{\mathbf{s}}_{kij}\left|\hat{\tilde{\mathbb{A}}}_{kij}\right|\left(\hat{\bar{\mathbf{u}}}_j-\bar{\mathbf{u}}_i\right)\right] = 0 \tag{5.9}$$

The above equation can be effectively written in the following form

$$V_i \frac{d}{dt}\left(\bar{\mathbf{u}}_i\right) + \frac{1}{2}\sum_{k=1}^{3}\Psi_{ki} = 0, \tag{5.10}$$

where Ψ_{ki} is written in the term of diagonal and off-diagonal entries as

$$\Psi_{ki} = \underbrace{\left(\sum_{j}^{\text{int}}\Delta \mathbf{s}_{kij} + \sum_{j}^{\text{bn}}\Delta \hat{\mathbf{s}}_{kij}\right)\mathbf{f}_k\left(\bar{\mathbf{u}}_i\right) + \sum_{j}^{\text{int}}\Delta \mathbf{s}_{kij}\mathbf{f}_k\left(\bar{\mathbf{u}}_j\right)}_{[\mathbf{M}]_{ki}[\mathbf{f}]_{ki}}$$
$$+ \underbrace{\left(\sum_{j}^{\text{int}}\left|\tilde{\mathbb{A}}_{kij}\right|\Delta \mathbf{s}_{kij} + \sum_{j}^{\text{bn}}\left|\hat{\tilde{\mathbb{A}}}_{kij}\right|\Delta \hat{\mathbf{s}}_{kij}\right)\bar{\mathbf{u}}_i - \sum_{j}^{\text{int}}\left|\tilde{\mathbb{A}}_{kij}\right|\Delta \mathbf{s}_{kij}\bar{\mathbf{u}}_j}_{[\mathbf{Q}]_{ki}[\bar{\mathbf{u}}]}$$
$$+ \underbrace{\sum_{j}^{\text{bn}}\Delta \hat{\mathbf{s}}_{kij}\hat{\mathbf{f}}_k\left(\bar{\mathbf{u}}_j\right)}_{[\hat{\mathbf{M}}]_{ki}[\hat{\mathbf{f}}]_{ki}} - \underbrace{\sum_{j}^{\text{bn}}\left|\hat{\tilde{\mathbb{A}}}_{kij}\right|\Delta \hat{\mathbf{s}}_{kij}\hat{\bar{\mathbf{u}}}_j}_{[\hat{\mathbf{Q}}]_{ki}[\hat{\bar{\mathbf{u}}}]} = 0, \tag{5.11}$$

Now, if we write down (5.11) for *all* nodes, i.e. all 'i' indices, we come up with the following equation

$$[\Psi]_k = [\mathbf{M}]_k[\mathbf{f}]_k + [\mathbf{Q}]_k[\bar{\mathbf{u}}] + \left[\hat{\mathbf{M}}\right]_k\left[\hat{\mathbf{f}}\right]_k + \left[\hat{\mathbf{Q}}\right]_k\left[\hat{\bar{\mathbf{u}}}\right] \tag{5.12}$$

where $[\bar{\mathbf{u}}]$ is a nested vector of conservative variables $\bar{\mathbf{u}}$ for all points j=1... $\mathrm{N}_{\text{nodes}}$

in the grid, say

$$[\bar{\mathbf{u}}] = \begin{bmatrix} \begin{bmatrix} \bar{\mathbf{u}}_1 \\ \vdots \\ \bar{\mathbf{u}}_y \end{bmatrix}_1 \\ \begin{bmatrix} \bar{\mathbf{u}}_1 \\ \vdots \\ \bar{\mathbf{u}}_y \end{bmatrix}_2 \\ \vdots \\ \begin{bmatrix} \bar{\mathbf{u}}_1 \\ \vdots \\ \bar{\mathbf{u}}_y \end{bmatrix}_{Nnodes} \end{bmatrix} \tag{5.13}$$

and the coefficient matrix $[\mathbf{M}]_k$ is

$$\mathbf{M}_k = \begin{bmatrix} \left[\sum_j^{\text{int.}} \Delta\mathbf{s}_{k1j} + \sum_j^{\text{bn.}} \hat{\Delta}\mathbf{s}_{k1j}\right] & & \left[\sum_j^{\text{int.}} \Delta\mathbf{s}_{k1j}\right] \\ & \left[\sum_j^{\text{int.}} \Delta\mathbf{s}_{k2j} + \sum_j^{\text{bn.}} \hat{\Delta}\mathbf{s}_{k2j}\right] & \\ & & \ddots \\ & & \left[\sum_j^{\text{int.}} \Delta\mathbf{s}_{k\mathrm{N}j} + \sum_j^{\text{bn.}} \hat{\Delta}\mathbf{s}_k \right. \end{bmatrix}$$

(j^{th} $\downarrow$)

where each sub-block entry $\left[\sum_j^{\text{int.}} \Delta\mathbf{s}_{kij} + \sum_j^{\text{bn.}} \hat{\Delta}\mathbf{s}_{kij}\right]$ in (5.14) is obtained using using multipication of scalar value $\sum_j^{\text{int.}} \Delta\mathbf{s}_{kij} + \sum_j^{\text{bn.}} \hat{\Delta}\mathbf{s}_{kij}$ with *identity* matrix of size $y \times y$ where y is the number of equations in system of conservation laws.

It should be noted that the matrix $[\mathbf{Q}]_k$ in (5.12) is simply obtained by replacing each block on the main dagonal (5.14) with 'y' times 'y' block $\left[\sum_j^{\text{int}} \left|\tilde{\mathbb{A}}_{kij}\right| \Delta\mathbf{s}_{kij} + \sum_j^{\text{bn}} \left|\hat{\tilde{\mathbb{A}}}_{kij}\right|\right.$ and off-diagonal elements with $-\sum_j^{\text{int}} \left|\tilde{\mathbb{A}}_{kij}\right| \Delta\mathbf{s}_{kij}$. We also note that matrices $\left[\hat{\mathbf{M}}\right]_k$ and $\left[\hat{\mathbf{f}}\right]_k$ and $\left[\hat{\mathbf{Q}}\right]_k$ all depend on the conservative variables at boundaries, i.e. $\left[\hat{\bar{\mathbf{u}}}\right]$. Therefore they are independent of $[\bar{\mathbf{u}}]$ and this is important when we want to calculate Jacobian matrix.

We proceed by finding residuals. From eq.(5.10) we have

$$\frac{d}{dt}\left(\bar{\mathbf{u}}_i\right) = \mathbf{R}_i\left(t\right) = -\frac{1}{2V_i}\sum_{k=1}^{3}\Psi_{ki}, \tag{5.15}$$

Substituting (5.12) into (5.15) we will have

$$\left[\mathbf{R}\right]\left(t\right) = -\left[\mathrm{diag}\left(\frac{1}{2V_i}\right)\right]\sum_{k=1}^{3}\left[\mathbf{M}\right]_k[\mathbf{f}]_k + \left[\mathbf{Q}\right]_k\left[\bar{\mathbf{u}}\right] + \left[\hat{\mathbf{M}}\right]_k\left[\hat{\mathbf{f}}\right]_k + \left[\hat{\mathbf{Q}}\right]_k\left[\hat{\bar{\mathbf{u}}}\right] \tag{5.16}$$

Thus the derivative of residual with respect to

$$\frac{\partial\left[\mathbf{R}\right]}{\partial\bar{\mathbf{u}}}\left(t\right) = -\left[\mathrm{diag}\left(\frac{1}{2V_i}\right)\right]\sum_{k=1}^{3}\left[\mathbf{M}\right]_k\left[\frac{\partial\mathbf{f}}{\partial\bar{\mathbf{u}}}\right]_k + \frac{\partial}{\partial\bar{\mathbf{u}}}\left(\left[\mathbf{Q}\right]_k\left[\bar{\mathbf{u}}\right]\right) \tag{5.17}$$

Bibliography

Beam, R.M. & Warming, R.F. 1978 An implicit finite difference algorithm for hyperbolic systems in conservation-law form. *J. Comput. Phys.* **22**, 87–110.

Briley, R. W. & McDonald, H. 1977 Solution of the multi-dimensional compressible navierstokes equations by a generalized implicit method. *J. Comput. Phys.* **21**, 372–397.

Briley, R. W. & McDonald, H. 2001 An overview and generalization of implicit navier-stokes algorithms and approximate factorization. *J. Computers and Fluids* **30**, 807–828.

Coddington, E. A. & Norman, L. 1955 *Theory of Ordinary Differential Equations*. McGraw-Hill.

Ghasemi, A. 2010 Developing nonlinear ode solvers for practical simulation of air vehicle configurations using compact schemes for integration. *Proceedings of AIAA Guidance, Navigation, and Control Conference, AIAA 2010-7808* .

Goldberg, D. 1991 What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys* **23(1)**, 5–48.

Lomax, H., Pulliam, T. H. & Zingg, D. W. 2004 *Fundamentals of Computational Fluid Dynamics*. Springer.

Quarteroni, A., Sacco, R. & Saleri, F. 2000 *Numerical Mathematics*. Springer.

Shampine, L. F. & Reichelt, M. W. 1997 The matlab ode suite. *SIAM Journal on Scientific Computing* **18**, 1–22.

Sidi, A. 2003 Practical extrapolation methods theory and applications. *Cambridge Univ. Press* .

Trefethen, L. N. 2001 *Spectral Methods in MATLAB*. SIAM.

Wang, Li 2009 Techniques for high-order adaptive discontinuous galerkin discretizations in fluid dynamics. PhD thesis, University of Wyoming.