

Appendix: Numerical Solutions of Initial Value Problems

This text book is about finite difference and finite element methods for BVPs of differential equations assuming that the readers have knowledge of numerical analysis which includes numerical methods for IVPs of differential equations. The purpose of the appendix is to provide a necessary supplement for those readers who have not been exposed to numerical methods for IVP.

A.1 System of First-Order ODEs of IVPs

We briefly explain finite difference methods for IVPs, particularly how to use Matlab ODE suite to solve such problems. The purpose of this appendix is to make the book more complete and self-contained. Usually, the material in the appendix with more depth can be found in the later stage of a numerical analysis class. The readers who have not been exposed to the materials will find the appendix useful. The methods described here can be used as time discretization techniques for various applications.

Consider an IVP of the following,

$$\begin{aligned} \frac{dy}{dt} &= \mathbf{f}(t, \mathbf{y}), \\ \mathbf{y}(t_0) &= \mathbf{v}, \end{aligned} \tag{A.1}$$

where $\mathbf{f}(t, \mathbf{y})$ is a given vector (or scalar) function, and \mathbf{v} is a known vector (or scalar). The following is the component form of the problem,

$$\begin{aligned} \frac{dy_1}{dt} &= f_1(t, y_1(t), y_2(t), \dots, y_m(t)), \\ \frac{dy_2}{dt} &= f_2(t, y_1(t), y_2(t), \dots, y_m(t)), \\ &\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\ \frac{dy_m}{dt} &= f_m(t, y_1(t), y_2(t), \dots, y_m(t)), \end{aligned} \tag{A.2}$$

with a known initial condition $y_1(t_0) = v_1, y_2(t_0) = v_2, \dots, y_m(t_0) = v_m$. Such an IVP is a quasilinear system since the highest derivative terms (here is the first order) are linear. Often we have $t_0 = 0$. We also use the notation $\frac{dy}{dt} = \mathbf{y}'(t)$. Below are two such examples. The first one is a scale IVP while the second one is a system of ODEs of an IVP.

Example A.1.

$$y'(t) = \sin(y(t)), \quad y(0) = 1.$$

Example A.2.

$$y_1'(t) = y_2(t), \tag{A.3}$$

$$y_2'(t) = -y_1(t) + y_1(t)y_2(t), \tag{A.4}$$

$$y_1(0) = 1, y_2(0) = 0.$$

Note that a high-order quasilinear ODE IVP can be converted to a system of first-order ODEs of an IVP as shown in the following example.

Example A.3. Convert the IVP,

$$y'' + \cos^2 t y' + \sin y = 0, \quad y(0) = v_1, \quad y'(0) = v_2,$$

to a first-order system of ODEs of an IVP, where y_0 and y_1 are two given constants.

Solution: We set $y_1(t) = y(t)$, and $y_2(t) = y'(t)$. Thus we have $y_1'(t) = y_2(t); y_2'(t) = y''(t) = -\cos^2 t y' - \sin y = -(\cos^2 t) y_2 - \sin y_1$. The original second-order IVP has been transformed to the following first-order system of ODEs of the IVP,

$$\begin{aligned} y_1'(t) &= y_2, & y_1(0) &= v_1 \\ y_2'(t) &= -\sin y_1 - (\cos^2 t) y_2, & y_2(0) &= v_2. \end{aligned}$$

A.2 Well-Posedness of an IVP

Most of the numerical methods are designed for well-posed problems. Ill-posed problems need special algorithms and analysis. Here we discuss only well-posed first-order IVPs (A.1). A well-posed problem means that the solution exists, and is unique, and is not sensitive to perturbations of the data, such as the initial condition. For an IVP (A.1), the Lipschitz continuity can guarantee the well-posedness.

Theorem A.4. Assume that $\mathbf{f}(t, \mathbf{y})$ is Lipschitz continuous in a neighborhood \mathcal{D} of (t_0, \mathbf{y}_0) , that is, given any $\epsilon > 0$, there is a constant L such that

$$\|\mathbf{f}(t, \mathbf{y}_1) - \mathbf{f}(t, \mathbf{y}_2)\| \leq L \|\mathbf{y}_1 - \mathbf{y}_2\| \quad (\text{A.5})$$

for all (t, \mathbf{y}_1) and (t, \mathbf{y}_2) in \mathcal{D} . Then in the neighborhood of \mathcal{D} there is a unique solution $\mathbf{y}(t)$ to (A.1) that is not sensitive to perturbation of the data in the original problem.

For a well-posed problem, the solution should be insensitive to the data such as the initial condition and coefficients involved. For the IVP (A.1), this implies the dynamical stability. If all the eigenvalues of the following matrix

$$\frac{D\mathbf{f}}{D\mathbf{y}}(t, \mathbf{y}(t)) = \begin{bmatrix} \frac{\partial f_1}{\partial y_1} & \frac{\partial f_1}{\partial y_2} & \cdots & \frac{\partial f_1}{\partial y_m} \\ \frac{\partial f_2}{\partial y_1} & \frac{\partial f_2}{\partial y_2} & \cdots & \frac{\partial f_2}{\partial y_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial y_1} & \frac{\partial f_m}{\partial y_2} & \cdots & \frac{\partial f_m}{\partial y_m} \end{bmatrix} \quad (\text{A.6})$$

at $(t_0, \mathbf{y}(t_0))$ are negative, that is, $\lambda_i \left(\frac{D\mathbf{f}}{D\mathbf{y}}(t_0, \mathbf{y}(t_0)) \right) \leq 0, i = 1, 2, \dots, m$, then the IVP is dynamically stable in a neighborhood of $(t_0, \mathbf{y}(t_0))$.

A.3 Some Finite Difference Methods for Solving IVPs

Assume that the IVP (A.1) is well-posed, and we wish to find the solution $\mathbf{y}(t)$ at some final time $T > t_0$. For simplicity of discussion, we simply set $t_0 = 0$ and discuss time marching methods. We start with a uniform time discretization. Given a parameter N , let $\Delta t = \text{int}(T/N)$, $t^0 = 0$, $t^n = n\Delta t$ with Δt be the uniform time step size. We wish to find an approximate solution \mathbf{y}^n to the IVP problem at t^n , $\mathbf{y}^n \approx \mathbf{y}(t^n, \mathbf{y}(t^n))$, $n = 1, 2, \dots, N$.

A.3.1 The Forward Euler’s Scheme

The simplest method is the forward Euler’s method

$$\frac{\mathbf{y}^{n+1} - \mathbf{y}^n}{\Delta t} = \mathbf{f}(t^n, \mathbf{y}^n), \quad n = 0, 1, \dots \quad (\text{A.7})$$

The method is first-order accurate, that is, $\|\mathbf{y}^n - \mathbf{y}(t^n, \mathbf{y}(t^n))\| \leq C\Delta t$. The scheme is conditionally stable meaning that we cannot take very large Δt .

We can give a reasonable guess of $\Delta t < 1$. Theoretically, the method is stable if we choose Δt such that $|\Delta t \lambda_i \left(\frac{Df}{Dy}(t_0, \mathbf{y}(t_0)) \right)| \leq 1$ for all i 's.

A.3.2 The Backward Euler's Scheme

The backward Euler's method is

$$\frac{\mathbf{y}^{n+1} - \mathbf{y}^n}{\Delta t} = \mathbf{f}(t^{n+1}, \mathbf{y}^{n+1}), \quad n = 0, 1, \dots \quad (\text{A.8})$$

The method is also first-order accurate, that is, $\|\mathbf{y}^n - \mathbf{y}(t^n, \mathbf{y}(t_0))\| \leq C\Delta t$. The scheme is unconditionally stable meaning that we can take any Δt . The difficulty of the backward Euler's method is that we need to solve a nonlinear system of equations in order to get the approximate solution at the next time step. The advantage of this method is the strongest stability, sometimes it is called a metastable method.

A.3.3 The Crank–Nicholson Scheme

The Crank–Nicholson scheme is the following

$$\frac{\mathbf{y}^{n+1} - \mathbf{y}^n}{\Delta t} = \frac{1}{2} \left(\mathbf{f}(t^n, \mathbf{y}^n) + \mathbf{f}(t^{n+1}, \mathbf{y}^{n+1}) \right), \quad n = 0, 1, \dots \quad (\text{A.9})$$

The method is second-order accurate, that is, $\|\mathbf{y}^n - \mathbf{y}(t^n, \mathbf{y}(t_0))\| \leq C(\Delta t)^2$. The scheme is unconditionally stable for linear problems. The difficulty of the Crank–Nicholson scheme is that we need to solve a nonlinear system of equations in order to get the approximate solution at the next time step. The stability is better than forward Euler's method but not as good as the backward Euler's method.

The methods that we discussed above are all one step methods. We refer the readers to general textbook on *Numerical Analysis*, for example, the one by Burden and Faires (2010) about other methods including multistep methods for IVPs and analysis.

A.3.4 Runge–Kutta (RK(k)) Methods

In numerical analysis, the Runge–Kutta methods are a family of implicit and explicit finite difference methods, which includes various Euler Methods, used in temporal discretization for the approximate solutions of ODEs. In this subsection, we explain explicit Runge–Kutta methods for scalar IVPs ($m = 1$). Below are some examples.

Example A.5. The Crank–Nicholson scheme is an implicit scheme. We can use a predictor and a corrector scheme to make the scheme explicit,

$$\begin{aligned} \text{predictor: } \quad \tilde{y}^{n+1} &= y^n + \Delta t f(t^n, y^n) \\ \text{corrector: } \quad y^{n+1} &= y^n + \frac{\Delta t}{2} \left(f(t^n, y^n) + f(t^n, \tilde{y}^{n+1}) \right). \end{aligned} \quad (\text{A.10})$$

The scheme is called Heun’s scheme which is second-order accurate and it is conditionally stable. The scheme above can be written as a one-step scheme

$$y^{n+1} = y^n + \frac{\Delta t}{2} \left(f(t^n, y^n) + f(t^n, y^n + \Delta t f(t^n, y^n)) \right). \quad (\text{A.11})$$

A class of Runge–Kutta (2) methods can be written as

$$y^{n+1} = y^n + \Delta t \left(\left(1 - \frac{1}{2\alpha}\right) f(t^n, y^n) + \frac{1}{2\alpha} f\left(t^n + \alpha\Delta t, y^n + \alpha\Delta t f(t^n, y^n)\right) \right),$$

for some $\alpha > 0$. If $\alpha = 1$, we get the above Heun’s scheme. If $\alpha = 1/2$, we get the middle point method. The middle point scheme is also second-order accurate.

The general k -th Runge–Kutta methods (RK(k)) can be written as

$$y^{n+1} = y^n + \Delta t \sum_{i=1}^k b_i f_i, \quad (\text{A.12})$$

where

$$\begin{aligned} f_1 &= f(t^n, y^n), \\ f_2 &= f\left(t^n + c_2\Delta t, y^n + \Delta t(a_{21}f_1)\right), \\ f_3 &= f\left(t^n + c_3\Delta t, y^n + \Delta t(a_{31}f_1 + a_{32}f_2)\right), \\ &\vdots \\ f_k &= f\left(t^n + c_k\Delta t, y^n + \Delta t(a_{k1}f_1 + a_{k2}f_2 + \cdots + a_{k,k-1}f_{k-1})\right). \end{aligned}$$

To specify a particular method, one needs to provide the integer k , and the coefficients a_{ij} , (for $1 \leq j < i \leq k$), b_i (for $i = 1, 2, \dots, k$), and c_i (for $i = 2, 3, \dots, k$). The matrix a_{ij} is called the Runge–Kutta matrix, while the b_i and c_i are known as the weights and the nodes. Those coefficients can be arranged in Table A.1.

The Runge–Kutta (RK(k)) method is consistent if

$$\sum_{j=1}^{i-1} a_{ij} = c_i \quad \text{for } i = 2, \dots, k.$$

Table A.1. The table of the coefficients of a RK(k) method

0					
c_2	a_{21}				
c_3	a_{31}	a_{32}			
\vdots	\vdots	\dots			
c_k	a_{k1}	a_{k2}	\dots	$a_{k,k-1}$	
	b_1	b_2	\dots	b_{k-1}	b_k

A well-known RK(4) has the following form. Given a step size $h > 0$ and define

$$y^{n+1} = y^n + \frac{\Delta t}{6} (f_1 + 2f_2 + 2f_3 + f_4),$$

where

$$\begin{aligned} f_1 &= f(t^n, y^n), \\ f_2 &= f\left(t^n + \Delta t/2, y^n + \Delta t f_1/2\right), \\ f_3 &= f\left(t^n + \Delta t/2, y^n + \Delta t f_2/2\right), \\ f_4 &= f\left(t^n + \Delta t, y^n + \Delta t f_3\right). \end{aligned}$$

The coefficients table is

Table A.2. The table of the coefficients of a RK(4) method

0				
$\frac{1}{2}$	$\frac{1}{2}$			
$\frac{1}{2}$	0	$\frac{1}{2}$		
1	0	0	1	
	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$

A.4 Solving IVPs Using Matlab ODE Suite

It is quite easy to use the Matlab ODE Suite to solve a system of first-order ODEs of an IVP and visualize the results. The Matlab ODE Suite is a collection of five user-friendly finite difference codes for solving IVPs given by first-order systems of ODEs and plotting their numerical solutions. The three codes ode23, ode45, and ode113 are designed to solve nonstiff problems and the two codes ode23s and ode15s are designed to solve both stiff and nonstiff

problems. The mathematical and software developments and analysis are given in Lawrence (1997). The Matlab ODE Suite is based on Runge–Kutta methods and can choose time step size adaptively.

To call one of five methods in the Matlab ODE Suite, for example, `ode23`, we can simply typing in Matlab command window the following

```
[t,y] = ode23('yp',[0,5],[1,0]');
```

assume that we have already defined the ODE system in a Matlab file called `yp.m`. For Example A.2, the Matlab script file `yp.m` can be written as

```
function yp = yp(t,y)
k = length(y); yp = zeros(k,1); % detect the dimension,
                                using column vector.
yp(1) = y(2); % Definition of f(t,y): y(1), y(2),
              ..., y(k).
yp(2) = -y(1) + y(1)*y(2);
```

Running the Matlab command will return outputs t , an array of different time from $t=0$ to $t=5$; the matrix $y \in R^{N;2}$, the approximate solution of $y(t)$ corresponding to the time. The column vectors $y(:,1)$ and $y(:,2)$ are the components of the approximation solution of $y_1(t)$ and $y_2(t)$, respectively. In Figure A.1, we plot the solution of $y_1(t)$ against time (the top plot); $y_2(t)$ against time (the middle plot); and the phase plot $y_2(t)$ against $y_1(t)$ (the bottom plot).

We can put everything into a Matlab script file called `ivp_ex2.m`, below, which we can modify later.

```
[t,y] = ode23('yp',[0,5],[1,0]');
y1=y(:,1); y2=y(:,2);
subplot(3,1,1); plot(t,y1)
xlabel('t'); ylabel('y_1(t)')
subplot(3,1,2); plot(t,y2)
xlabel('t'); ylabel('y_2(t)')
subplot(3,1,3); plot(y1,y2)
xlabel('y_1'); ylabel('y_2')
```

In Matlab command window, we just need to type “`ivp_ex2`” and then we will see the plot.

We can replace `ode23` with other four subroutines in Matlab ODE Suite, `ode45`, and `ode113` that are designed to solve nonstiff problems and the two codes `ode23s` and `ode15s` that are designed to solve both stiff and nonstiff problems. If we are not sure whether a problem is stiff or not, we can always use the codes for stiff problems.

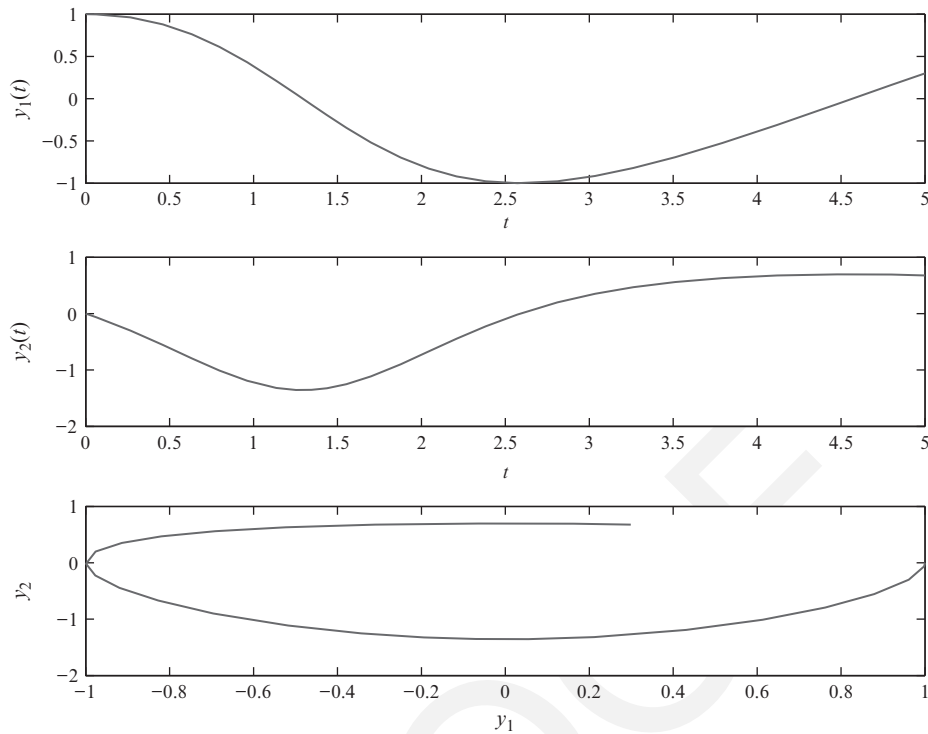


Fig. A.1. Plots of the approximate solution. (a) $y_1(t)$ against time; (b) $y_2(t)$ against time; and (c) the phase plot $y_2(t)$ against $y_1(t)$.

Example A.6. As another demonstration, we solve the nondimensionalized Lotka–Volterra predator–prey model of the following system,

$$\begin{aligned} y_1' &= y_1 - y_1 y_2, \\ y_2' &= -a y_2 + y_1 y_2, \\ y_1(0) &= p_1, \quad y_2(0) = p_2, \end{aligned} \tag{A.13}$$

where p_1 and p_2 are two constants, $y_1(t)$ is the population of a prey, while $y_2(t)$ is the population of a predator. Under certain conditions, predator and prey can coexist.

This problem is potentially stiff as we can see from some of plots of the solutions. We define the system in a Matlab function called *prey_prd.m* whose contents are

```
function yp = prey_prd(t,y)
global a
k = length(y); yp = zeros(k,1);
```

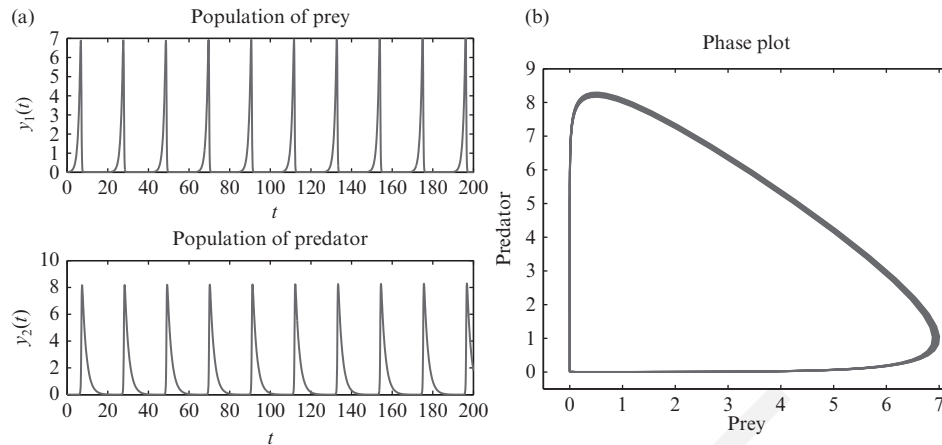



Fig. A.2. Plots of the solution of the prey–predator model from $t=0$ to $t=200$ in which we can see the prey and predator coexist. (a) solution plots against time and (b) the phase plot in which the limit cycle can be seen.

$$\begin{aligned} \text{yp}(1) &= y(1) - y(1)*y(2) ; \\ \text{yp}(2) &= -a*y(2) + y(1)*y(2) ; \end{aligned}$$

To solve the problem, we write a Matlab script file called *prey_prd_drive.m* whose contents are the following.

```
global a
a = 0.5; t0 = 0; y0 = [0.01 0.01]; tfinal=200;
[t y] = ode23s('prey_prd',[t0,tfinal],y0);
y1 = y(:,1); y2=y(:,2); % Extract solution components.
figure(1); subplot(211); plot(t,y1); title('Population of prey')
subplot(212); plot(t,y2); title('Population of predator')
figure(2); plot(y1,y2) % Phase plot
xlabel('prey'); ylabel('predator'); title('phase plot')
```

In Figure A.2, we plot the computed solution for the parameters $a = 0.5$, the initial data is $y_1(0) = y_2(0) = 0.01$. The final time is $T = 200$. The left plot are the solution of each component against time. We can observe that the solution changes rapidly in some regions indicating the stiffness of the problem. The right plot is the phase plot, that is, the plot of one component against the other. The phase plot is more like a closed curve in the long run indicating the existence of the limiting cycle of the model.

Exercises

1. Consider a model for chemical reaction,

$$\begin{aligned}y_1' &= a - (b + 1)y_1 + y_1^2 y_2, \\y_2' &= b y_1 - y_1^2 y(2).\end{aligned}$$

Use Matlab ODE Suite to solve the problem with various parameters, initial data, and final time.

2. Use Matlab ODE Suite to solve the Lorenz equations,

$$\begin{aligned}y_1' &= \sigma (y_2 - y_1) \\y_2' &= y_1 (\rho - y_3) - y_1 y_2, \\y_3' &= y_1 y_2 - \beta y_3,\end{aligned}$$

where σ , ρ , and β are constants. Try to solve the problem with various parameters, initial data, and final time, particularly, try to get the Lorenz attractor.

References

- J. Adams, P. Swarztrauber, and R. Sweet. Fishpack: Efficient Fortran sub-programs for the solution of separable elliptic partial differential equations. www.netlib.org/fishpack/.
- D. Braess. *Finite Elements: Theory, Fast Solvers, and Applications in Solid Mechanics*. Cambridge University Press, Cambridge, UK, 3rd edition, 2007.
- S. C. Brenner and L. R. Scott. *The Mathematical Theory of Finite Element Methods*. Springer, New York, 2002.
- R. L. Burden and J. D. Faires. *Numerical Analysis*. 2010, PWS-Kent Publ. Co., 2006, Brooks/Cole Cengage Learning, Boston, MA, 9th edition, 2010.
- D. Calhoun. A Cartesian grid method for solving the streamfunction-vorticity equation in irregular regions. *J. Comput. Phys.*, 176:231–75, 2002.
- G. F. Carey and J. T. Oden. *Finite Element, I–V*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1983.
- A. J. Chorin. Numerical solution of the Navier–Stokes equations. *Math. Comp.*, 22:745–62, 1968.
- P. G. Ciarlet. *The Finite Element Method for Elliptic Problems*. North Holland, 1978 and *Classics in Applied Mathematics* 40. SIAM, Philadelphia, 2002.
- D. De Zeeuw. Matrix-dependent prolongations and restrictions in a blackbox multigrid solver. *J. Comput. Appl. Math.*, 33:1–27, 1990.
- J. E. Dennis, Jr. and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. SIAM, Philadelphia, PA, 1996.
- L. C. Evans. *Partial Differential Equations*. AMS, Providence, RI, 1998.
- G. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, 2nd edition, 1989.
- H. Huang and Z. Li. Convergence analysis of the immersed interface method. *IMA J. Numer. Anal.*, 19:583–608, 1999.
- A. Iserles. *A First Course in the Numerical Analysis of Differential Equations*. Cambridge University Press, Cambridge, UK, 2008.
- H. Ji, J. Chen, and Z. Li. A symmetric and consistent immersed finite element method for interface problems. *J. Sci. Comput.*, 61(3):533–57, 2014.
- C. Johnson. *Numerical Solution of Partial Differential Equations by the Finite Element Method*. Cambridge University Press, Cambridge, UK, 1987.
- R. J. LeVeque. Clawpack and Amrclaw – Software for high-resolution Godunov methods. *4th International Conference on Wave Propagation*, Golden, CO, 1998.

- R. J. LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations, Steady State and Time Dependent Problems*. SIAM, Philadelphia, PA, 2007.
- Z. Li. *The Immersed Interface Method – A Numerical Approach for Partial Differential Equations with Interfaces*. PhD thesis, University of Washington, Seattle, Washington, 1994.
- Z. Li. The immersed interface method using a finite element formulation. *Appl. Numer. Math.*, 27:253–67, 1998.
- Z. Li and K. Ito. The immersed interface method – Numerical solutions of PDEs involving interfaces and irregular domains. *SIAM Front. Ser. Appl. Math.*, FR33, 2006.
- Z. Li and M.-C. Lai. The immersed interface method for the Navier–Stokes equations with singular forces. *J. Comput. Phys.*, 171:822–42, 2001.
- Z. Li, T. Lin, and X. Wu. New Cartesian grid methods for interface problem using finite element formulation. *Numer. Math.*, 96:61–98, 2003.
- Z. Li and C. Wang. A fast finite difference method for solving Navier–Stokes equations on irregular domains. *J. Commun. Math. Sci.*, 1:180–96, 2003.
- M. Minion. A projection method for locally refined grids. *J. Comput. Phys.*, 127:158–78, 1996.
- K. W. Morton and D. F. Mayers. *Numerical Solution of Partial Differential Equations*. Cambridge University Press, Cambridge, UK, 1995.
- J. W. Ruge and K. Stuben. Algebraic multigrid. In S. F. McCormick, ed., *Multigrid Method*. SIAM, Philadelphia, PA, 1987, 73–130.
- Y. Saad. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7:856–69, 1986.
- L. F. Shampine and M. W. Reichelt. The MATLAB ODE suite. *SIAM J. Sci. Comput.*, 18(1):1–22, 1997.
- Z.-C. Shi. Nonconforming finite element methods. *J. Comput. Appl. Math.*, 149:221–5, 2002.
- G. Strang and G. J. Fix. *An Analysis of the Finite Element Method*. Prentice-Hall, Upper Saddle River, NJ, 1973.
- J. C. Strikwerda. *Finite Difference Scheme and Partial Differential Equations*. Wadsworth & Brooks, Belmont, CA, 1989.
- K. Stüben. Algebraic multigrid (AMG): An introduction with applications. *Gesellschaft für Mathematik und Datenverarbeitung*, Nr. 70, 1999.
- J. W. Thomas. *Numerical Partial Differential Equations: Finite Difference Methods*. Springer, New York, 1995.
- R. E. White. *An Introduction to the FE Method with Applications to Non-linear Problems*. John Wiley & Sons, Charlottesville, VA, 1985.

Index

- 1D IFEM, 221
- 1D IIM, 39
- 1D Sturm–Liouville problem, 27, 169
- 1D interface problem, 39
- 1D interpolation function, 175
- 2D IFEM, 269
- 2D interface problem, 270
- 2D second order self-adjoint elliptic PDE, 230

- abstract FE method, 219
- ADI method, 99
 - consistency, 103
 - stability, 103
- algebraic precision of Gaussian quadrature, 196
- assembling element by element, 203

- backward Euler method
 - 1D parabolic, 84
 - 2D parabolic, 98
 - stability, 96
- Beam–Warming method, 118
- biharmonic equation, 263
- bilinear basis on a rectangle, 265
- bilinear form, 137
 - 1D elliptic, 169
 - 2D elliptic PDE, 231
- boundary value problem (BVP), 3

- Cauchy–Schwartz inequality, 163
- CFL condition, 83, 111
- characteristics, 109
- compatibility condition, 50, 230
- conforming FE methods, 168
- conforming FEM, 143
- Courant–Friedrichs–Lewy, 83
- Crank–Nicolson scheme
 - for 1D advection equation, 119
 - 1D parabolic equation, 87
 - 2D, 99
 - in FEM for 2D parabolic, 274
- cubic basis function
 - in 1D H^1 , 195
 - in 1D H^2 , 210
 - in 2D $H^1 \cap C^0$, 261
 - in 2D $H^2 \cap C^1$, 263
- D’Alembert’s formula, 122
- degree of freedom, 190, 210
- discrete Fourier transform, 92
- discrete inverse Fourier transform, 92
- discrete maximum principle in 2D, 57
- distance in a space, 159
- domain of dependence, 122
- domain of influence, 122
- double node, 212
- dynamical stability, 281

- eigenvalue problem, 26, 225
- energy norm, 170, 216
- essential BC, 209
- essential boundary condition, 181, 209
- explicit method, 81

- fast Fourier transform (FFT), 71
- FD approximation for $u'(x)$, 14
 - backward finite difference, 14
 - central finite difference, 14
 - forward finite difference, 14
- FD in polar coordinates, 69
- FE method for parabolic problems, 272
- FE space in 1D, 144
- finite difference grid, 9
- finite difference (FD) method, 5
 - 1D grid points, 9
 - 1D uniform Cartesian grid, 9
 - consistency, 23
 - convergence, 24
 - discretization, 23
 - finite difference stencil, 10
 - five-point stencil in 2D, 52

292

Index

- ghost point method in 2D, 60
- local truncation error, 10, 22
- master grid point, 23, 52
- stability, 24, 25
- step size, 15
- finite element method (FEM), 5, 135
 - a 1D element, 143
 - a 1D node, 143
 - assembling element by element, 147
 - hat functions, 136
 - piecewise linear function, 136
 - weak form in 1D, 136
- finite element solution, 136
- first order accurate, 15
- FM spaces on rectangles, 265
- forward Euler method
 - 1D parabolic, 81
 - 2D heat equation, 97
 - in FEM for 2D parabolic PDE, 274
 - stability, 94
- Fourier transform (FT), 89
- fourth order BVP in 1D, 209
- fourth order compact scheme in 2D, 67
- fourth-order compact scheme, 69
- functional spaces, 158

- Galerkin method, 143
- Gauss–Seidel iterative method, 63
- Gaussian points and weights, 196
- Gaussian quadrature formulas, 195
- global basis functions, 238
- grid refinement analysis, 16
- growth factor, 95

- hat functions, 145

- immersed finite element method (IFEM), 269
- implicit Euler method, 274
- initial value problem (IVP), 2, 279
- inner product in H^m , 166
- inner product in L^2 , 162
- interpolation function in 2D, 241
- inverse Fourier transform, 72, 89

- Jacobi iterative method, 62

- $L^2(\Omega)$ space, 160
- Lax–Friedrichs method, 110
- Lax–Milgram Lemma, 214, 215, 231
- Lax–Wendroff scheme, 117
- leap-frog scheme, 114
 - for heat equation, 96
- linear form, 137
- linear transform in 2D, 246
- local load vector, 149
- local stiffness matrix, 149
- local stiffness matrix and load vector, 204

- local truncation error
 - 1D parabolic, 82

- maximum principle in 2D elliptic PDE, 55
- mesh generation, 233
- mesh parameters, 233
- mesh size in 1D, 143
- method of line (MOL), 85
- method of undetermined coefficients, 19
- minimization form, 220
- mixed (Robin) boundary condition, 34
- modified PDE, 115
 - Lax–Wendroff method, 118
- multi-index notation, 159

- natural boundary condition, 182, 210
- natural jump conditions
 - in 1D, 40, 221
 - in 2D, 269
- natural ordering, 52
- neutral stability, 114
- nine-point discrete Laplacian, 69
- nonconforming IFE space, 271
- numerical boundary condition, 120
- numerical dissipation, 116
- numerical solutions, 2

- one-sided finite difference, 19
- one-way wave equations, 108
- ordinary differential equation (ODE), 1

- partial differential equation (PDE), 1
- piecewise linear basis function
 - in 2D H^1 , 234
- piecewise quadratic function in 2D H^1 , 258
- Poincaré inequality, 217, 231
- pole singularity, 71
- p -th order accurate, 15

- quadratic basis function in 1D H^1 , 190
- quadrature formula in 2D, 247
- quintic function, 264

- red-black ordering, 52
- Ritz method, 143, 146
- round-off errors, 26, 55
- Runge–Kutta methods RK(k), 283

- second Green’s theorem, 228
- shape function, 199, 212
- simplified FE algorithm in 2D, 251
- Sobolev embedding theorem, 167
- Sobolev space, 164
- SOR(ω) method, 64
- stability
 - Lax–Wendroff scheme, 117
- staggered grid in polar coordinates, 71
- steady state solution, 79, 105

Index

293

- Sturm–Liouville problem in 1D, 182
- symmetric positive definite (SPD), 54
- Taylor expansion, 14
- time marching method, 81, 281
- triangulation, 233
- truncated Fourier series, 71
- unconditionally stability, 96
- upwind scheme for 1D advection equation, 111
- upwinding discretization, 29
- von Neumann stability analysis, 89, 94
- wave equation, 108
- weak derivative, 164
- weak form
 - 1D Sturm–Liouville BVP, 183
 - 2D elliptic PDE, 231

PROOF

PROOF