COMPUTATIONAL MODELLING SAMMARY II

BASED ON COMPUTATIONAL MODELLING BY S U HULSHOFF

SAM YAN ELSLOO

APRIL - JUNE 2017

ナマケモノの戦争馬の攻撃

Preface

Just as a warning, considering the previous quiz was so easy, it's imo reasonable to expect that this quiz will be relatively harder.

Contents

6 Chapter 6: Time march methods

Unsteady problems are problems in which u is a function of time; e.g. $\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0$ is an unsteady problem as it involves time. On the other hand, $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$ (the Laplace equation in 2D), is not considered an unsteady problem. These are fundamentally different in the fact that there is a clear future and past in unsteady problems, which you have to take into account in your numerical model; for example, to find a finite-difference operator, although you can go in two directions for the x and y derivatives, you are only allowed to go in one direction for the time derivative (you can only use past points, not future points). Naturally speaking, this means that methods for unsteady problems show fundamentally different behaviour. These methods are called **time march methods**, as we will usually end up evaluating the solution at one time, and then march forward into the future using that solution. The main characteristic that's interesting to discuss is the stability of it, and that'll be the main focus of this chapter.

6.1 Accuracy of transient computations

6.1.1 The exact amplification factor

In the previous quiz, we discussed the amplification factor of a numerical solution, for which we considered a single component of the Fourier series:

$$u_i^n = U e^{at} e^{I k_m x}$$

Determining the numerical amplification factor

DETERMINING THE AMPLIFICATION FACTOR

1. Replace u_i^n with 1.
2. For the nodes in the time direction, use the following substitution:

$$u_i^{n+p} = e^{ap\Delta t} \tag{6.1}$$

3. For the nodes in spatial direction, use the following substitution:

$$u_{i+q}^n = e^{I k_m q \Delta x} \tag{6.2}$$

4. Rewrite to find an explicit expression for $\rho = e^{a\Delta t}$.
5. If required use the following equations:

$$e^{Ix} = \cos(x) + I\sin(x) \tag{6.3}$$
$$\cos(2x) = \cos^2(x) - \sin^2(2x) = 1 - 2\sin^2(x) \tag{6.4}$$
$$\sin(2x) = 2\sin(x)\cos(x) \tag{6.5}$$

6. Work out

$$|\rho| = \sqrt{(\mathrm{Re}(\rho))^2 + (\mathrm{Im}(\rho))^2} \tag{6.6}$$

Now, there is also an *exact* amplification factor, corresponding to the exact solution. Consider the linear advection equation, $\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x}$. Again, we will merely consider one Fourier component, i.e.

$$u = U e^{at} e^{I k_m x}$$

We then have

$$\begin{aligned}\frac{\partial u}{\partial t} &= U a e^{at} e^{Ik_m x} \\ \frac{\partial u}{\partial x} &= U I k_m e^{at} e^{Ik_m x}\end{aligned}$$

Substituting gives

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = U a e^{at} e^{Ik_m x} + cU I k_m e^{at} e^{Ik_m x} = 0$$

Divide by $U e^{at} e^{Ik_m x}$, and we have to solve the equation

$$\begin{aligned}a + cIk_m &= 0 \\ a &= -cIk_m\end{aligned}$$

Thus, the exact amplification factor is

$$\rho = e^{a\Delta t} = e^{-cIk_m\Delta t} = e^{-I\alpha\beta}$$

with $\alpha = \frac{c\Delta t}{\Delta x}$ and $\beta = k_m\Delta x$, where k_m is the wave number[1]. Note that the magnitude of the Fourier component is always equal to one[2]; however, its phase will always shift by $-\alpha\beta$ each time step.

DETERMINING THE EXACT AMPLIFICATION FACTOR

Starting with a PDE of the form (with r, s and t constants)

$$r\frac{\partial^p u}{\partial t^p} + s\frac{\partial^q u}{\partial x^q} + tu = 0$$

1. Replace the pth derivative of u with respect to t with a^p.
2. Replace the qth derivative of u with respect to x with $\left(Ik_m\right)^q$. Do not forget to work out I^q properly (e.g. $I^2 = -1$, $I^3 = -I$, $I^4 = 1$, $I^5 = I$, $I^6 = -1$, etc.
3. Replace u with 1.
4. Solve the remaining equation for a.
5. The exact amplification factor is then given by $e^{a\Delta t}$.
6. It is customary to rewrite this so that the term $\beta = k_m\Delta x$ is used.

For example, consider the linear diffusion equation, $u_t - vu_{xx}$. We substitute $a - v\left(Ik_m\right)^2 = a + vk_m^2 = 0$, so that we have $a = -vk_m^2$. Thus, the amplification factor is given by

$$\rho = e^{a\Delta t} = e^{-vk_m^2\Delta t} = e^{-g\beta^2}$$

[1]The concept of wave number is rather important to understand as we'll come across it a lot in both chapter 6 and 7. Let L be the length of the domain in spatial direction (e.g. if it runs between $x = 0$ and $x = \pi$, then $L = \pi$). The wavenumber then indicates the number of radians per unit distance. For example, if a certain wavenumber is 1, then that means that there is 1 radian of a wave between $x = 0$ and $x = 1$. If a certain wavenumber is 4, then there are 4 radians of a wave between $x = 0$ and $x = 1$ (and 4 radians between $x = 1$ and $x = 2$, etc.). So, higher wavenumbers mean that more waves fit within a given distance; this corresponds to higher frequency waves. If you fail to see how waves have anything to do with what we're discussing, remember that if you write out a complex exponential, you get a combination of a sine and cosine; we're talking to the waves that these terms generate.

Now, how can k_1, k_2, k_3 etc. be determined? The relation is $k_m = \frac{\pi m}{L}$. So, for example, for the case $L = \pi$, we have that $k_3 = \frac{\pi\cdot 3}{\pi} = 3$, i.e. 3 radians of a wave (where full wave would be 2π radians, obviously) fit between $x = 0$ and $x = 1$. The wave number does not need to be an integer, to be clear: suppose $L = 1$, then $k_5 = \frac{\pi\cdot 5}{1} = 5\pi$, i.e. 5π radians of wave (2.5 complete waves) fit between $x = 0$ and $x = 1$.

The final question remains, how far can you go with k_m? Should we continue until $k_{10000000}$, or should we stop somewhere before? That all depends on how large Δx is. Suppose we have $L = 1$ and $\Delta x = 0.001$. I assume you have either done the tutorial for instrumentation and signals, or already studied a bit for it. You have to see the x-values, $x = 0.000$, $x = 0.001$, $x = 0.002$, $x = 0.003$, etc. (the step is governed by $\Delta x = 0.001$) as points at which you sample the exact solution to get the discrete solution. You must have at least two samples per wave; you cannot have a wavenumber such that you have 2π radians of a wave (i.e. a full wave) that occupy less than $2\Delta x = 0.002$ distance, otherwise you'd get aliasing. If we have 2π radians per 0.002 unit distance, we have 1000π radians per unit distance (i.e. wavenumber $k_m = 1000\pi$), which we should not exceed. In general, the maximum wavenumber to be considered should be $k_m = 1000\pi$.

The main take-away is that the wavenumber states how many waves fit within a unit distance, so the higher the wavenumber, the higher the frequency.

[2]The exponent is purely imaginary. Remember from instrumentation and signals how you can easily determine the magnitude and phase of a complex exponential: e^{a+bI} with a and b both real can be written as $e^a \cdot e^{bI}$; the value of e^a is then the magnitude of the complex exponential; the phase shift is equal to b. In this case, we have $a = 0$ and $b = -\alpha\beta$.

with $g = \frac{v\Delta t}{\Delta x^2}$ and $\beta = k_m \Delta x$. Note that in this case, ρ_e is real, so there will be no change in the phase of the wave; there will only be a decrease in magnitude.

6.1.2 *Definition of amplitude error*

AMPLITUDE ERROR

The **amplitude error** is defined as

$$er_a = |\rho_e| - |\rho| \tag{6.7}$$

6.1.3 *Definition of phase error*

PHASE ERROR

The **phase error** is defined as

$$er_p = \angle\rho_e - \angle\rho \tag{6.8}$$

where $\angle\rho$ indicates the angle of the complex number ρ, i.e.

$$\angle\rho = \arctan \frac{\mathrm{Im}(\rho)}{\mathrm{Re}(\rho)} \tag{6.9}$$

When dealing with the linear advection equation, it is more common to examine the relative phase:

$$rp = \frac{\angle rp}{\angle rp_e} \tag{6.10}$$

Here, $rp > 1$ indicates the approximated wave component is moving faster than it should, while $rp < 1$ indicates that the approximated wave is lagging behind.

6.1.4 *Errors of some example methods*

In case you're totally lost right now what the amplitude and phase error are, consider the following. Consider the explicit in time, first-order upwind in space scheme for the linear advection equation. Plots of the amplification factor of the solution (*not* the amplitude error) and relative phase (*not* phase error) are shown in figure 6.1, for several values of $\alpha = \frac{c\Delta t}{\Delta x}$. The amplification plot looks familiar; we see that for $\alpha \leq 1$, the method is stable; for $\alpha > 1$, there are values of β for which the amplification factor is larger than 1. Furthermore, we see that the phase is relative phase is pretty close to 1 generally speaking, so we don't expect much phase difference (we'll see shortly what would happen if there is significant phase difference); the result for $\alpha = 0.5$ is shown in figure 6.2. After several time steps, the highest-frequency components of the initial square-wave solution have been removed; these high-frequency components are what allows for the very high gradients required to produce the square so that's why it flattens out. Since there are barely any phase errors, the wave moves at the correct speed.

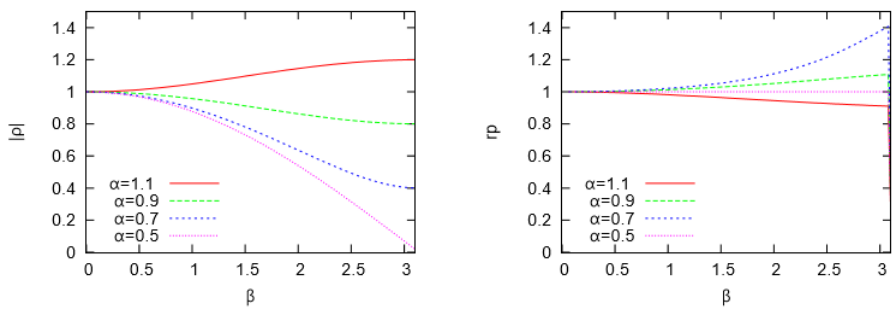


Figure 6.1: Amplficiation factor and relative phase of the explicit in time, upwind in space scheme.

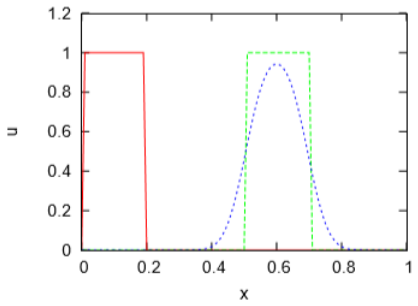


Figure 6.2: Comparison of the exact and computed solution from the explicit in time, upwind in space scheme at $\alpha = \frac{c\Delta t}{\Delta x} = 0.5$.

Now consider the implicit Euler in time, upwind in space scheme. $|\rho|$ and rp are shown in figure 6.3. Medium and high-frequency components are clearly damped out, and there is a significant relative phase for these components as well (even the low frequency components show a small relative phase). This is depicted in figure 6.4. If you look closely, you see that the blue line seems to lag the green line; this is especially visible when you compare where the blue line crosses the vertical green lines. Note that the high-frequency components, which would have had the highest relative phase, are already dampened out a bit, so essentially we only see the phase lag of the low-frequency components.

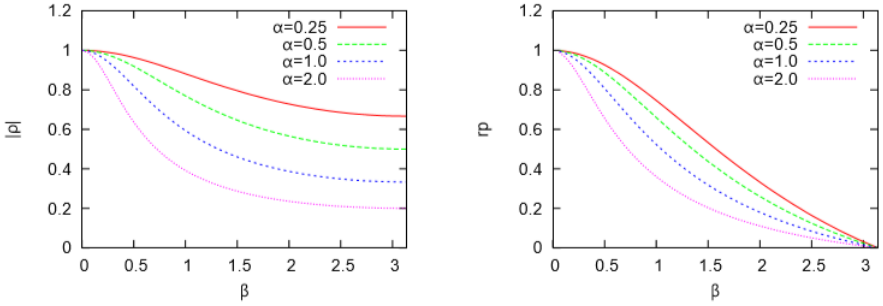


Figure 6.3: Amplficiation factor and relative phase of the implicit in time, upwind in space scheme.

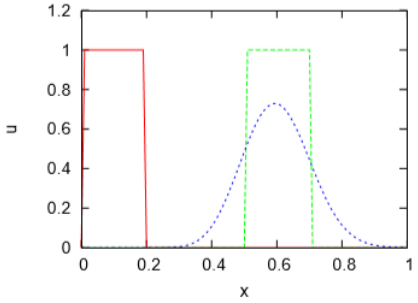


Figure 6.4: Comparison of the exact and computed solution from the implicit in time, upwind in space scheme at $\alpha = \frac{c\Delta t}{\Delta x} = 0.5$.

Now consider the implicit Euler in time, central in space scheme. $|\rho|$ and rp are shown in figure 6.5. Clearly, the medium-frequency (associated with β values around $\pi/2$) component are dampened out, even though the high and low-frequency components are almost completely unaffected. Furthermore, we see that especially the high-frequency components have a large relative phase: rp is much smaller than 1, which again suggests that the solution lags the exact solution. This is depicted in figure 6.6: we still observe high-frequency oscillations (those are preserved and not damped out after all), and we also see that the blue dotted line seems to lag the green line a bit: the relative phase.

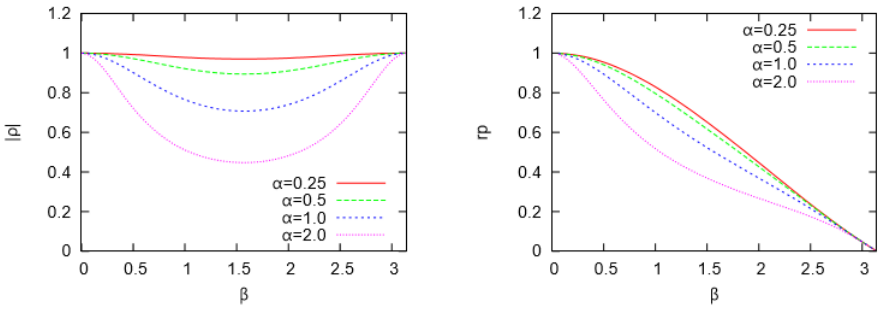


Figure 6.5: Amplficiation factor and relative phase of the implicit in time, central in space scheme.

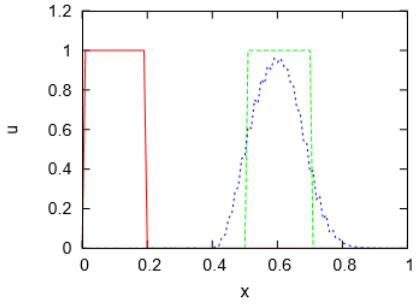


Figure 6.6: Comparison of the exact and computed solution from the implicit in time, central in space scheme at $\alpha = \frac{c\Delta t}{\Delta x} = 0.5$.

Quiz 3: Question 1

Consider the convection-reaction equation:

$$u_t + cu_x = ru$$

Defining $I = \sqrt{-1}$, $k = \frac{\pi m}{L}$ and $T = \Delta t$, the exact amplification factor of this equals $\rho_e = ...$

We replace u_t with a; u_x with Ik_m and u with 1. Then we have

$$\begin{aligned} u_t + cu_x &= ru \\ a + cIk_m &= r \\ a &= r - cIk_m \end{aligned}$$

so

$$\rho = e^{a\Delta t} = e^{(r-cIk_m)\Delta T}$$

Don't worry, you don't need to work this out as sines and cosines. Note that we now have that the magnitude of the exact amplification factor is

$$|\rho_e| = e^{r\Delta t}$$

and the exact phase is

$$\angle\rho_e = -ck_m\Delta t$$

Quiz 3: Question 2

Defining $I = \sqrt{-1}$, $k = \frac{\pi m}{L}$ and $T = \Delta t$, the exact amplification factor of the following PDE

$$u_t = q \cdot u_{xx} + r \cdot u$$

is defined by $\rho_e = ...$

Replace u_t with a, u_{xx} with $\left(Ik_m\right)^2 = -k_m^2$ and u with 1. Then, we get

$$\begin{aligned} u_t &= q \cdot u_{xx} + r \cdot u \\ a &= -q \cdot k_m^2 + r \end{aligned}$$

so that

$$\rho_e = e^{a\Delta t} = e^{\left(-q \cdot k_m^2 + r\right)\Delta t}$$

Note that ρ_e is completely real, so $\left|\rho_e\right| = \rho_e$ and $\angle\rho_e = 0$.

Quiz 2: Question 3

A finite difference approximation to the heat equation is:

$$u_i^{n+1} = u_i^n + g\left(u_{i+1}^n - 2u_i^n + u_{i-1}^n\right)$$

where $g = \frac{v\Delta t}{\Delta x^2}$. If the exact amplification factor is expressed as ρ_e and β_x, the amplitude error of the finite-difference approximation is:

- $\left|\rho_e\right| - |1 + 2g \cdot (\cos(\beta) - 1)|$
- $\left|\rho_e - 1 + 2g \cdot (\cos(\beta) - 1)\right|$
- $\left|\rho_e\right| - |1 + 2g \cdot I \cdot (\cos(\beta) - 1)|$
- $\left|\rho_e - 1 + 2g \cdot I \cdot (\cos(\beta) - 1)\right|$
- $\left|\rho_e\right| - |1 + g \cdot (\cos(\beta) - 2)|$
- $\left|\rho_e - 1 + g \cdot (\cos(\beta) - 2)\right|$
- $\left|\rho_e\right| - |1 + g \cdot I \cdot (\cos(\beta) - 2)|$
- $\left|\rho_e - 1 + g \cdot I \cdot (\cos(\beta) - 2)\right|$

The correct answer is $\left|\rho_e\right| - |\rho| = \left|\rho_e\right| - |1 + 2g(\cos(\beta) - 1)|$. Note that we do not need to compute the exact amplification factor ourselves, fortunately. The amplification factor for the numerical solution is easily found using the problem solving guide on page 7: we make the following substitutions:

$$\begin{aligned} u_i^{n+1} &= e^{at} \\ u_i^n &= 1 \\ u_{i+1}^n &= e^{Ik_m\Delta x} \\ u_{i-1}^n &= e^{-Ik_m\Delta x} \end{aligned}$$

so that we have

$$\begin{aligned} u_i^{n+1} &= u_i^n + g\left(u_{i+1}^n - 2u_i^n + u_{i-1}^n\right) \\ e^{at} &= 1 + g \cdot \left(e^{Ik_m\Delta x} - 2 + e^{-Ik_m\Delta x}\right) \\ \rho &= 1 + 2g\left(\frac{e^{I\beta} + e^{-I\beta}}{2} - 1\right) = 1 + 2g(\cos(\beta) - 1) \end{aligned}$$

Thus, the amplitude error is

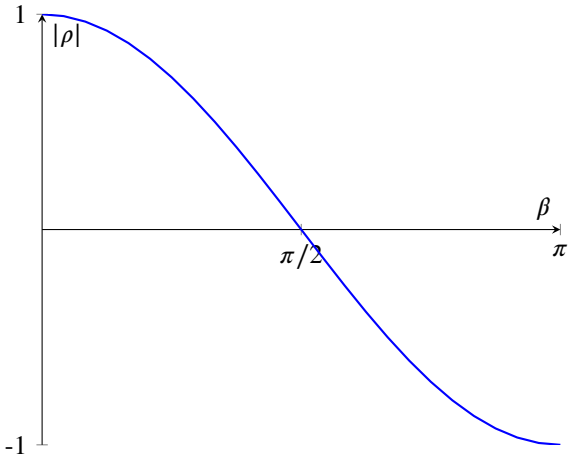
$$\left|\rho_e\right| - |\rho| = \left|\rho_e\right| - |1 + 2g(\cos(\beta) - 1)|$$

Quiz 2: Question 4

A finite-difference approximation for the linear convection equation has an amplification factor $\rho = \cos(\beta)$. One can expect that this scheme:

- amplifies waves of low frequency
- damps waves of low frequency
- amplifies waves of medium frequency
- damps waves of medium frequency
- amplifies waves of high frequency
- damps waves of high frequency

The correct answer is **damps waves of medium frequency**. On the domain $0 \leq \beta \leq \pi$, ρ looks like as shown below:



This means that $|\rho|$ is especially small for waves of medium frequency (around $\beta = \pi/2$), and thus waves of medium frequency are damped. Thus, the correct answer is **damps waves of medium frequency**.

6.2 General analysis of linear methods

6.2.1 From PDE to algebraic system

In the first quiz, you studied the fully-discrete approach for time problems; for example, for the linear advection equation,

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0$$

you'd find finite-difference operators for $\frac{\partial u}{\partial x}$ *and* $\frac{\partial u}{\partial t}$ and then combine everything into one big matrix equation, which you'd then solve. For example, if you used an implicit, central in space method for the linear advection equation, you'd end up at

$$\begin{bmatrix} 1 & \frac{c\Delta t}{2\Delta x} & & & & & -\frac{c\Delta t}{2\Delta x} \\ -\frac{c\Delta t}{2\Delta x} & 1 & \frac{c\Delta t}{2\Delta x} & & & & \\ & -\frac{c\Delta t}{2\Delta x} & 1 & \frac{c\Delta t}{2\Delta x} & & & \\ & & -\frac{c\Delta t}{2\Delta x} & 1 & \frac{c\Delta t}{2\Delta x} & & \\ & & & -\frac{c\Delta t}{2\Delta x} & 1 & \frac{c\Delta t}{2\Delta x} & \\ & & & & -\frac{c\Delta t}{2\Delta x} & 1 & \frac{c\Delta t}{2\Delta x} \\ \frac{c\Delta t}{2\Delta x} & & & & & -\frac{c\Delta t}{2\Delta x} & 1 \end{bmatrix} \mathbf{u}^{n+1} = \mathbf{u}^n$$

However, there is another approach, namely the **semi-discrete approach**, which is more popular actually. What we will do is *not* use a finite-difference operator for $\frac{\partial u}{\partial t}$, but only for the spatial derivative. For example, consider

the linear advection equation with periodic boundaries (so no need to take into account boundary conditions); we can write $\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x}$ as

$$\frac{\partial u}{\partial t} = -c\frac{\partial u}{\partial x} = -c\frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x}$$

leading to the matrix equation

$$\frac{\partial \mathbf{u}}{\partial t} = -\frac{c}{2\Delta x}\begin{bmatrix} 0 & 1 & & & & -1 \\ -1 & 0 & 1 & & & \\ & -1 & 0 & 1 & & \\ & & & & & \\ 1 & & & & -1 & 0 \end{bmatrix}\mathbf{u}$$

where (we have M nodes in spatial direction)

$$\mathbf{u} = \begin{bmatrix} u_1^n \\ u_2^n \\ \vdots \\ u_M^n \end{bmatrix}$$

Note that if you work this matrix equation out, you get a (coupled) system of ODEs. This can be solved exactly! A system of ODEs can be solved using linear algebra, and it'll give you a solution for *all* possible values of t, since you get a solution of the form (with λ_m being the eigenvalues; please don't think too much right now, just remember that this is how solutions to linear systems of ODEs look like)

$$\begin{aligned} u_1(t) &= \sum_{m=1}^{M} C_m e^{\lambda_m t} \\ u_2(t) &= \sum_{m=1}^{M} C_m e^{\lambda_m t} \\ \vdots \quad &\vdots \quad \vdots \\ u_M(t) &= \sum_{m=1}^{M} C_m e^{\lambda_m t} \end{aligned}$$

In other words, suppose you have a string between your hands. Curious as you are, you want to find a numerical solution to what happens when you vibrate it (2nd order wave equation). Using the semi-discrete approach, you then divide the string up in a few nodes (can be 2, can be 4, can be 10, can be thousands, but let M refer to the total number of nodes you use). You make a nice finite-difference operator in spatial direction, leading to some big matrix and a system of ODEs. Then, using some initial conditions, you can calculate the values of the solution at each of the nodes at $t = 0$, i.e. $u_1(0)$, $u_2(0)$, etc., allowing you to calculate the integration constants. You then have formulas for the values at the nodes for *all* values of t: you have $u_1(t)$, $u_2(t)$, etc. Using these formulas, you can directly compute $u_1(0.0001)$, $u_3(4.243542)$, everything you want. In other words, the solution is discrete in space (you only know the solution at certain points in spatial direction), but continuous in time (but, at those points, you know it as a continuous function of time), hence semi-discrete.

Compare this with the fully discrete approach you've been taught before: you only know the values of the solution at discrete points in space (separated by distances Δx), and even for those, you only know them at discrete points in time (separated a time Δt; if $\Delta t = 0.1$ for example, you can't know $u_1(0.005)$): it's therefore called fully discrete.

I hope the difference between fully discrete and semi-discrete is now clear.

6.2.2 *The exact solution of the semi-discrete system*

First, let me introduce some notation. If we have an unstable PDE, we will produce a coupled system of ODEs of the form

$$\frac{\partial \mathbf{u}}{\partial t} = [A]\,\mathbf{u} - \mathbf{f}$$

where **f** contains specified boundary and source-term values. To give you another example of how $[A]$ might look like, consider the linear diffusion equation, $\frac{\partial u}{\partial t} = \upsilon \frac{\partial^2 u}{\partial x^2}$:

$$\begin{aligned} \frac{\partial u}{\partial t} &= \upsilon \frac{\partial^2 u}{\partial x^2} = \upsilon \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} \\ \frac{\partial \mathbf{u}}{\partial t} &= \frac{\upsilon}{\Delta x^2} \begin{bmatrix} -2 & 1 & & & & 1 \\ 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & & & & \\ 1 & & & & 1 & -2 \end{bmatrix} \mathbf{u} \end{aligned}$$

In all subsequent sections, we will assume that **f** is constant over time (so the source terms and boundary conditions do not change over time); this means that we can neglect **f** alltogether and we will do so from this moment on.

Now, as I said, this is a *coupled* system of ODEs; this means that for example, u_2 appears in the equation for $\frac{du_1}{dt}$. This is not what we want, however. We prefer to have it as an uncoupled system (so that u_2 does not appear in any other equation than the one for $\frac{du_2}{dt}$, etc.), as these are extremely easy to solve. The way to do that is as follows:

Assume we have M nodes in spatial direction. Assume $[A]$, which will have size $M \times M$, has a complete set of linearly-independent eigenvectors (which it pretty much always will have), and then store these M eigenvectors in a matrix $[X]$, defined by

$$[X] = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & ... & \mathbf{x}_M \end{bmatrix}$$

In other words, the first column of $[X]$ is the first eigenvector; the second column is the second eigenvector, etc. Then, the following transformation holds:

$$[X]^{-1}[A][X] = [\Lambda] = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_M \end{bmatrix}$$

where λ_m, $m = 1, 2, ..., M$ are the eigenvalues of $[A]$. Note that $[\Lambda]$ is a diagonal matrix. Then, using the following steps (you'll agree with me that these modifications are allowed to be made):

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial t} &= [A]\mathbf{u} \\ \frac{\partial \mathbf{u}}{\partial t} &= [A][X][X]^{-1}\mathbf{u} \\ [X]^{-1} \frac{\partial \mathbf{u}}{\partial t} &= [X]^{-1}[A][X][X]^{-1}\mathbf{u} \\ [X]^{-1} \frac{\partial \mathbf{u}}{\partial t} &= [\Lambda][X]^{-1}\mathbf{u} \end{aligned}$$

Now, define $\mathbf{w} = [X]^{-1}\mathbf{u}$ (don't worry about what **w** would physically represent just yet), so that we can write

$$\frac{\partial \mathbf{w}}{\partial t} = [\Lambda]\mathbf{w}$$

which is an uncoupled system; since $[\Lambda]$ is diagonal, if we write this out, we'd get the equations

$$\begin{aligned} \frac{dw_1}{dt} &= \lambda_1 w_1 \\ \frac{dw_2}{dt} &= \lambda_2 w_2 \\ \vdots \quad &\vdots \quad \vdots \\ \frac{dw_M}{dt} &= \lambda_M w_M \end{aligned}$$

which are easily solvable! The solution for each of these equations is given by

$$w_m(t) = C_m e^{\lambda_m t}, \qquad m = 1, 2, ..., M$$

Since $\mathbf{w} = [X]^{-1}\mathbf{u}$, we have $\mathbf{u} = [X]\mathbf{w}$. Thus, the total solution is given by

$$\mathbf{u}(t) = [X]\mathbf{w}(t)$$

In case you got lost; $\mathbf{u}$ is the vector containing the solution values at each node; $\mathbf{w}$ is the vector containing w_1, w_2 etc.; it's like

$$\mathbf{u}(t) = \begin{bmatrix} u_1(t) \\ u_2(t) \\ \vdots \\ u_M(t) \end{bmatrix}, \qquad \mathbf{w}(t) = \begin{bmatrix} w_1(t) \\ w_2(t) \\ \vdots \\ w_M(t) \end{bmatrix}$$

6.2.3 *Meaning of eigenvectors and eigenvalues*

Now, I think it is rather poorly explained in the book (in the sense that it's basically completely missing), so let me try to explain to you what the significance of all this is because it's important to understand what's going on. Note that we are allowed to write

$$\mathbf{u}(t) = \sum_{m=1}^{M} w_m \mathbf{x}_m = \sum_{m=1}^{M} C_m e^{\lambda_m t} \mathbf{x}_m$$

In other words: $\mathbf{u}(t)$ is the sum you get by multiplying w_1 with eigenvector $\mathbf{x}_1$, and w_2 with multiplying with eigenvector $\mathbf{x}_2$, etc. What does this physically represent? Suppose we have a vibrating string, where we divide the length of the string in three nodes, $i = 1, 2, 3$. The spatial discretisation will result in three linearly independent eigenvectors; suppose they'll be

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \quad \mathbf{x}_2 = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \quad \mathbf{x}_3 = \begin{bmatrix} 1 \\ 3 \\ 4 \end{bmatrix}$$

Then, what this method tells you is that any time, $\mathbf{u}$ is given by a linear combination of these three vectors. Thus, the first entries in each of those eigenvectors determine the value u_1; the second entries in each determine the value u_2 and the third in each of those determine the value u_3. We can sort of 'plot' these eigenvectors as shown below in figure 6.7.

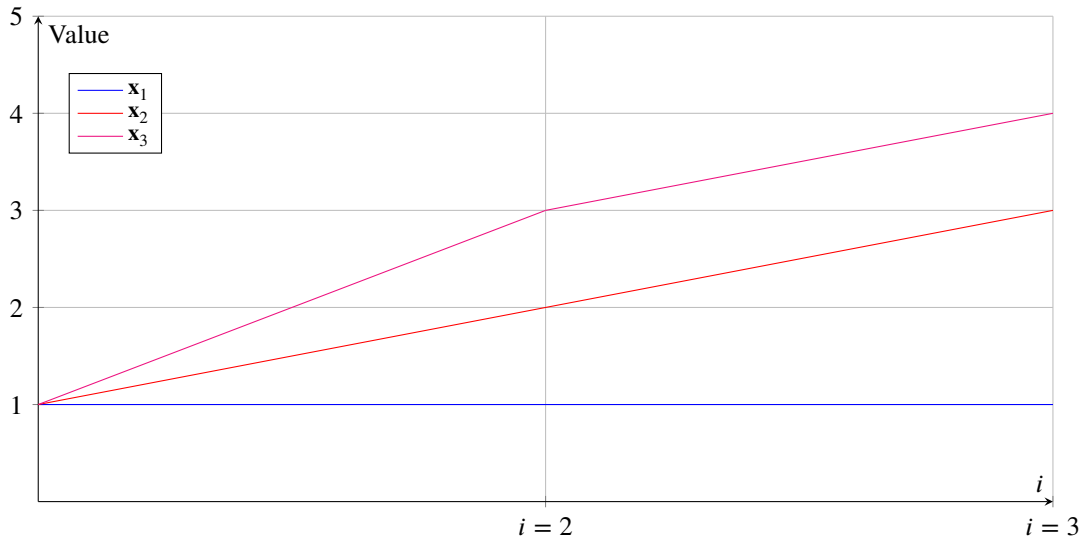


Figure 6.7: Plotting the eigenvectors.

Now, suppose that due to the initial conditions, $C_1 = 1$, $C_2 = 2$ and $C_3 = -1$; the solution at $t = 0$ is then given by (we don't have to take into account the exponential since $t = 0$, so the exponentials all equal 1)

$$\mathbf{u}(0) = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} C_1\mathbf{x}_1 + C_2\mathbf{x}_2 + C_3\mathbf{x}_3 = 1 \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} + 2 \cdot \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} - 1 \cdot \begin{bmatrix} 1 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ 3 \end{bmatrix}$$

Please note the implications of this: the solution value at $i = 1$ is a linear combination of the values of the eigenvectors at $i = 1$ (u_1); the solution value at $i = 2$ (u_2) is a linear combination of the values of the eigenvectors at $i = 2$ and the solution value u_3 is a linear combination of the values of the eigenvectors at $i = 3$. This is visualized in figure 6.8, where each of the eigenvectors is multiplied with its corresponding weight a_i; the solution $\mathbf{u}$ is then simply the summation of these three eigenvectors:

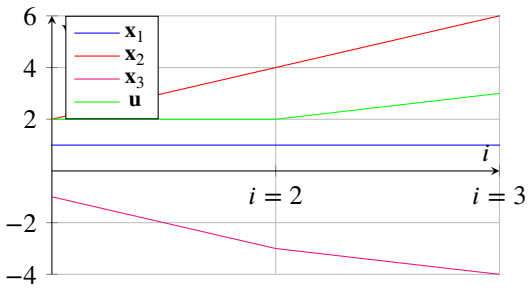


Figure 6.8: Plotting the eigenvectors multiplied by their weights ($a_1 = 1$, $a_2 = 2$ and $a_3 = -1$) and the resulting solution, found by superimposing the red, blue and neonpink line.

Now, there is this extra component that we need to take into account, $e^{\lambda_m t}$, where λ_m is the eigenvalue associated with the eigenvector. What effect does this have? Well, first, note that λ_m may be a complex value, so perhaps it's better to write

$$e^{\lambda_m t} = e^{\mathrm{Re}(\lambda_m)t} \cdot \left[\cos\left(\mathrm{Im}\left(\lambda_m\right)t\right) + I\sin\left(\mathrm{Im}\left(\lambda_m\right)t\right)\right]$$

What's the meaning of this? Depending on the values of λ_m, the weight of each of the eigenvector changes over time: it may be the case that at a later point in time, say at $t = 1$, we should use the weights

$$\mathbf{u}(1) = \frac{1}{2} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} + 3 \cdot \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + \frac{1}{4} \cdot \begin{bmatrix} 1 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} \frac{15}{4} \\ \frac{29}{4} \\ \frac{21}{2} \end{bmatrix}$$

We can then make a new figure where multiply the eigenvector with these weights, and show the resulting solution as done in figure 6.9.

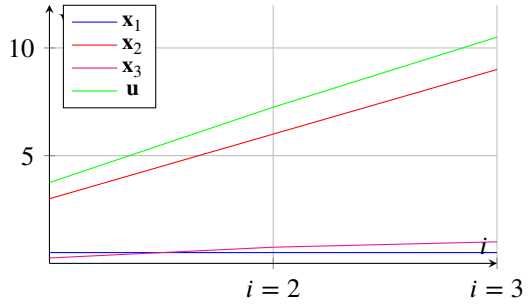


Figure 6.9: Plotting the eigenvectors multiplied by their weights ($a_1 = 1/2$, $a_2 = 3$ and $a_3 = 1/4$) and the resulting solution, found by superimposing the red, blue and neonpink line.

What is the point I'm trying to make? Look back at figure 6.9: the eigenvectors provide 'shapes' of the solution, and the solution will at all times be a linear combination of these shapes (also called 'modes'). The weights,

however, change over time, as the weights equal $C_m e^{\lambda_m t}$. It is clear that the value of λ_m greatly affects stuff: for example, consider the case where λ_1 is completely real and very negative. In that case, the presence of eigenvector $\mathbf{x}_1$ in the final solution $\mathbf{u}$ will decrease very quickly, due to the dampening effect of having a negative number in your exponent. Similarly, suppose λ_2 would be purely imaginary: then the presence of eigenvector $\mathbf{x}_2$ would oscillate over time, as you get sines and cosines in there so the weight corresponding to $\mathbf{x}_2$ oscillates a lot. And finally suppose λ_3 would be partly real (and positive) and partly imaginary; then the presence of $\mathbf{x}_3$ will amplify over time, but it'll also oscillate a bit.

Please, please try to understand what I just explained: although you could be able to do the rest of this chapter without it, it's really helpful to have an understanding of what it all means. I think you would not be surprised at all to hear this is greatly important to stability. How do we suddenly get to stability? Well, remember when I said that you'd get exact solutions for $u_1(t)$, $u_2(t)$ etc. when using a semi-discrete approach? That was actually a lie; in the end, when you have $\frac{du}{dt}$ = something, you will use a numerical method to find u_i^{n+1} (for example, you could use explicit Euler time march:

$$u_m^{n+1} = u_m^n + \Delta t \frac{du_m^n}{dt}$$

which you want to have stable behaviour, naturally). This is actually the process that is called **time marching**. Do note already: the requirement for stability will *not* be that all λ_m have real parts that are negative or zero (so there's no λ_m for which the real part is larger than 0). We'll later see what exactly the requirement is for λ_m.

Quiz 3: Question 7

Assume matrices $[A]$ and $[B]$ have the same complete set of linearly-independent right eigenvectors, which are collected into matrix $[X]$. The following system of equations:

$$([A] + [B]) \cdot \mathbf{y} = \mathbf{f}$$

can be decoupled by considering a new vector of variables defined by: $\mathbf{q} = ...$

We use a slightly different notation for this question, but other than that it's exactly the same. Remember we used the transformation $\mathbf{w} = [X]^{-1}\,\mathbf{u}$ before? Well, now $\mathbf{w}$ has become $\mathbf{q}$ and $\mathbf{u}$ has become $\mathbf{y}$, so the answer is simply $\mathbf{q} = [X]^{-1} \cdot \mathbf{y}$.

Quiz 3: Question 8

A diagonalised version of the semi-discrete system

$$\frac{\partial \mathbf{u}}{\partial t} = [A]\,\mathbf{u}$$

can be obtained using $[X]$, the matrix of eigenvectors of $[A]$, by writing:

- $[X]\,\frac{\partial \mathbf{u}}{\partial t} = [X]\,[A]\,[X]\,[X]^{-1}\,\mathbf{u}$
- $[X]^{-1}\,\frac{\partial \mathbf{u}}{\partial t} = [X]^{-1}\,[A]\,[X]^{-1}\,\mathbf{u}$
- $[X]\,\frac{\partial \mathbf{u}}{\partial t} = [X]\,[A]\,[X]\,[X]\,\mathbf{u}$
- $[X]^{-1}\,\frac{\partial \mathbf{u}}{\partial t} = [X]^{-1}\,[A]\,[X]\,[X]^{-1}\,\mathbf{u}$
- $\frac{\partial \mathbf{u}}{\partial t} = [X]^{-1}\,[A]\,\mathbf{u}$
- $[X]^{-1}\,\frac{\partial \mathbf{u}}{\partial t} = [X]\,[A]\,[X]\,\mathbf{u}$

Correct is $[X]^{-1}\,\frac{\partial \mathbf{u}}{\partial t} = [X]^{-1}\,[A]\,[X]\,[X]^{-1}\,\mathbf{u}$. Just something you should remember, the derivation of

this. In case you forgot it and don't want to go back a few pages:

$$\begin{aligned}
\frac{\partial \mathbf{u}}{\partial t} &= [A]\mathbf{u} \\
\frac{\partial \mathbf{u}}{\partial t} &= [A][X][X]^{-1}\mathbf{u} \\
[X]^{-1}\frac{\partial \mathbf{u}}{\partial t} &= [X]^{-1}[A][X][X]^{-1}\mathbf{u} \\
[X]^{-1}\frac{\partial \mathbf{u}}{\partial t} &= [\Lambda][X]^{-1}\mathbf{u}
\end{aligned}$$

6.2.4 *Values of eigenvalues and eigenvector for discretisations with periodic BCs*

Note that the eigenvalues follow strictly from the used space discretisation (as this is what determined the matrix $[A]$). Now, consider a finite-difference operator with periodic BCs; this will be of the form

$$\begin{bmatrix} b_0 & b_1 & b_2 & b_3 \\ b_3 & b_0 & b_1 & b_2 \\ b_2 & b_3 & b_0 & b_1 \\ b_1 & b_2 & b_3 & b_0 \end{bmatrix}$$

These type of matrices are called **circulant matrices**. The eigenvectors of these systems can be easily computed (don't bother with the derivation). For the mth eigenvector, i.e. $\mathbf{x}_m$, the jth entry in this mth eigenvector, i.e. $\left(x_j\right)_m$, is given by:

EIGENVECTOR OF CIRCULANT MATRIX

The eigenvectors of circulant matrices are given by

$$\mathbf{x}_m = \left(x_j\right)_m = e^{I(2\pi jm/M)} \tag{6.11}$$

with $j = 0, 1, ...M-1$ and $m = 0, 1, ..., M-1$, where m denotes the mth eigenvector, j the jth entry in that eigenvector, and size of matrix $[A]$ (which is of size $M \times M$).

For example, suppose we use four nodes in our solution, then $M = 4$. Then the third eigenvector, $\mathbf{x}_3$, would be given by[3]

$$\mathbf{x}_3 = \begin{bmatrix} e^{I(2\pi\cdot 1\cdot 3/4)} \\ e^{I(2\pi\cdot 2\cdot 3/4)} \\ e^{I(2\pi\cdot 3\cdot 3/4)} \\ e^{I(2\pi\cdot 4\cdot 3/4)} \end{bmatrix} = \begin{bmatrix} -i \\ -1 \\ i \\ 1 \end{bmatrix}$$

The eigenvalues can be expressed as

EIGENVALUES OF PERIODIC DISCRETISATION

$$\lambda_m = \sum_{j=0}^{M-1} b_j e^{I(2\pi jm/M)} \tag{6.12}$$

[3] In similar ways, we'd get

$$\mathbf{x}_1 = \begin{bmatrix} i \\ -1 \\ -i \\ 1 \end{bmatrix}, \qquad \mathbf{x}_2 = \begin{bmatrix} -1 \\ 1 \\ -1 \\ 1 \end{bmatrix}, \qquad \mathbf{x}_4 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Continuing above example, this would for example mean that the third eigenvalue ($m = 3$) would equal[4]

$$\begin{aligned}\lambda_3 &= b_0 e^{I(2\pi\cdot 0\cdot 3/4)} + b_1 e^{I(2\pi\cdot 1\cdot 3/4)} + b_2 e^{I(2\pi\cdot 2\cdot 3/4)} + b_3 e^{I(2\pi\cdot 3\cdot 3/4)} \\ &= b_0 - ib_1 - b_2 + ib_3\end{aligned}$$

So, for linear periodic boundary conditions, the eigenvectors and eigenvalues are extremely easy to calculate:

- Eigenvectors are exactly the same for circular matrices of the same size, independent of what discretisation is used.
- Eigenvalues are different for circular matrices of the same size.

Quiz 3: Question 5

Consider the following matrices

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}; \qquad B = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \\ 2 & 3 & 1 \end{bmatrix}; \qquad C = \begin{bmatrix} 5 & 4 & 2 \\ 2 & 5 & 4 \\ 4 & 2 & 5 \end{bmatrix}$$

Which statement is true?

- A and B have the same eigenvalues
- A and B have the same eigenvectors
- A and C have the same eigenvalues
- A and C have the same eigenvectors
- B and C have the same eigenvalues
- B and C have the same eigenvectors

Correct is **B and C have the same eigenvectors**. B and C are both circular 3×3 matrices. So the eigenvectors are the same (although the eigenvalues are different).

Quiz 3: Question 6

Which of the following is true?

- Circulant matrices have linearly dependent eigenvectors.
- Circulant matrices of the same size have the same eigenvectors.
- Circulant matrices of the same size have the same eigenvalues.
- Circulant matrices are always symmetric.

Correct is **Circulant matrices of the same size have the same eigenvectors.** The eigenvectors are linearly independent; circulant matrices have different eigenvalues and they are basically never symmetric.

Now, let us consider some PDEs and their spatial discretisation (as this is what determines the eigenvalues) and plot their eigenvalues in the complex plane (if we choose a specific discretisation, the values of b_j are known so we can easily plot the eigenvalues in the complex plane). Consider the following cases:

- The linear diffusion equation, $\frac{\partial u}{\partial t} = \upsilon \frac{\partial^2 u}{\partial x^2}$. Using a central discretisation for $\frac{\partial^2 u}{\partial x^2}$, i.e.[5]

$$\frac{\partial^2 u}{\partial x^2} = \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2}$$

[4] In similar fashion, we'd get

$$\begin{aligned}\lambda_1 &= b_0 + ib_1 - b_2 - ib_3 \\ \lambda_2 &= b_0 - b_1 + b_2 - b_3 \\ \lambda_4 &= b_0 + b_1 + b_2 + b_3\end{aligned}$$

[5] Using b_{-1} to denote the last b (i.e. if you have a 4×4 matrix for $[A]$, then $b_{-1} = b_3$), we realize that this means that $b_0 = -2$ and $b_1 = b_{-1} = 1$.

you can compute that you get eigenvalues that are completely real and negative[6]. This results in a transient solution which decays in time, mimicking the exact diffusion process.

- The linear advection equation, $\frac{\partial u}{\partial t} = -c\frac{u}{x}$. Using again a central discretisation, i.e.

$$\frac{\partial u}{\partial x} = \frac{u_{i+1} - u_{i-1}}{2\Delta x}$$

we now get $b_0 = 0$, $b_1 = 1$ and $b_{-1} = -1$ (the division by 2 is put in front of the matrix $[A]$); this results in eigenvalues that are purely imaginary. This means that the initial conditions propagate unattenuated, albeit their probably will be some sort of oscillation.

The eigenvalues of these two discretisations are shown in the complex plane in figure 6.10.

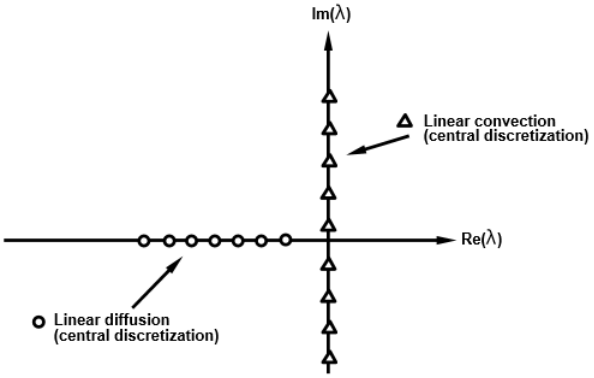


Figure 6.10: Semi-discrete eigensystems for central discretisations.

Let's focus on what different discretisations would mean for the linear advection equation:

- The linear advection equation, $\frac{\partial u}{\partial t} = -c\frac{u}{x}$, now using an *upwind* discretisation, i.e.

$$\frac{\partial u}{\partial x} = \frac{u_i - u_{i-1}}{\Delta x}$$

so that $b_0 = 1$ and $b_{-1} = -1$. The resultant eigenvalues are shown in figure 6.11.

- The linear advection equation, $\frac{\partial u}{\partial t} = -c\frac{u}{x}$, now using an *downwind* discretisation, i.e.

$$\frac{\partial u}{\partial x} = \frac{u_{i+1} - u_i}{\Delta x}$$

so that $b_0 = -1$ and $b_1 = 1$. The resultant eigenvalues are shown in figure 6.11.

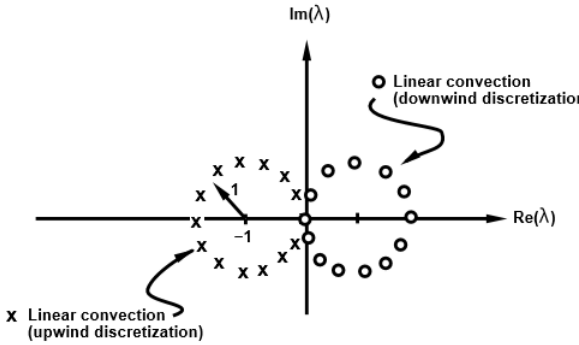


Figure 6.11: Semi-discrete eigensystems for one-sided discretisations.

Clearly, we'd prefer the upwind discretization: Re $\left(\lambda_m\right) < 0$ for those, meaning that the eigenvectors are not amplified (but quite the opposite).

[6]If you don't trust me, take a certain value for M (e.g. $M = 10$). Then, you only have $b_0 = -2$, $b_1 = b_9 = 1$, so you can easily compute the eigenvalue with the given formula.

6.2.5 *The exact solution of the fully-discrete system*

For the fully-discrete system (for $\mathbf{f} = \mathbf{0}$), we have

$$\mathbf{u}^{n+1} = [C]\,\mathbf{u}^n$$

Again, we can do some transformations with this. Let $[Y]$ be the matrix containing the eigenvectors of $[C]$, then we get

$$\begin{aligned} \mathbf{u}^{n+1} &= [C]\,\mathbf{u}^n \\ \mathbf{u}^{n+1} &= [C]\,[Y]\,[Y]^{-1}\,\mathbf{u}^n \\ [Y]^{-1}\,\mathbf{u}^{n+1} &= [Y]^{-1}\,[C]\,[Y]\,[Y]^{-1}\,\mathbf{u}^n \\ [Y]^{-1}\,\mathbf{u}^{n+1} &= [\Sigma]\,[Y]^{-1}\,\mathbf{u}^n \end{aligned}$$

where $[\Sigma]$ is the matrix containing the eigenvalues σ_m, $m = 1, 2, ..., M$. Defining $\mathbf{w}^n = [Y]^{-1}\,\mathbf{u}^n$, we get

$$\mathbf{w}^{n+1} = [\Sigma]\,\mathbf{w}^n$$

or

$$w_m^{n+1} = \sigma_m w_m^n, \qquad m = 1, 2, ..., M$$

since $[\Sigma]$ is a diagonal matrix. What is the significance of this? Since $\mathbf{u} = [Y]\mathbf{w}$, it makes sense to say the requirement for stability is

REQUIREMENT FOR STABILITY

For a time-march to be stable, the eigenvalues λ_m must be such that

$$|\sigma_m| < 1 \tag{6.13}$$

where σ_m may be complex.

6.3 *Relation between λ_m and σ_m*

That should be a logical requirement. Now, how are λ_m exactly related to σ_m? Well, suppose that for the semi-discrete system, for the time-march, we want to use an explicit Euler time march:

$$u_m^{n+1} = u_m^n + \Delta t \frac{du_m^n}{dt}$$

Since for the semi-discrete system $\mathbf{u} = [X]\mathbf{w}$, it makes sense that this basically comes down to

$$w_m^{n+1} = w_m^n + \Delta t \frac{dw_m^n}{dt}$$

However, bear in mind that in the analysis for the semi-discrete system, we got the uncoupled ODEs

$$\frac{dw_m}{dt} = \lambda_m w_m$$

Thus, we essentially get

$$w_m^{n+1} = w_m^n + \Delta t \lambda_m w_m = \left(1 + \Delta t \lambda_m\right) w_m$$

Comparing this with

$$w_m^{n+1} = \sigma_m w_m^n$$

we realize that for the explicit Euler time march

$$\sigma_m = 1 + \Delta t \lambda_m$$

In other words, if you want to use explicit Euler time march, all λ_m need to be between 0 and $-2/\Delta t$. So, clearly, the value of λ_m itself is not the only important thing for stability; it also depends on how the derivatives in the time-march (i.e. the discretisation in time) are used.

Another example, if you have the implicit Euler method,

$$u_m^{n+1} = u_m^n + \Delta t \frac{du_m^{n+1}}{dt}$$

then we first write

$$w_m^{n+1} = w_m^n + \Delta t \frac{dw_m^{n+1}}{dt}$$

then we realize that $dw_m^{n+1}/dt = \lambda_m w_m^{n+1}$ so that we get

$$\begin{aligned} w_m^{n+1} &= w_m^n + \Delta t \lambda_m w_m^{n+1} \\ \left(1 - \Delta t \lambda_m\right) w_m^{n+1} &= w_m^n \\ w_m^{n+1} &= \frac{w_m^n}{1 - \Delta t \lambda_m} \end{aligned}$$

so that from comparison with $w_m^{n+1} = sigma_m w_m^n$, we see that

$$\sigma_m = \frac{1}{1 - \Delta t \lambda_m}$$

So, we need to make our discretisation in space such that the eigenvalues are such that $\left|\sigma_m\right| < 1$.

Now, personally, I think this way of deriving the $\sigma_m - \lambda_m$ relation is a little bit complicated, especially if you are given more complicated discretisations in time. Therefore, let me introduce my way of doing it in the quickest way possible (if you don't get what I write down, look at the subsequent examples):

DERIVING THE $\lambda - \sigma$ RELATION

Starting with a discretisation of the form

$$u_m^{n+1} = \sum_{k=1}^{s} a_{s-k} u_m^{n+1-k} + \Delta t \sum_{k=0}^{s} b_{s-k} \frac{du_m^{n+1-k}}{dt}, \qquad m = 1, 2, ..., M \tag{6.14}$$

- Multiply all derivatives with λ_m. Note that it may be that higher-order derivatives appear in your discretisation; in general, multiply the pth derivative with $\left(\lambda_m\right)^p$.
- Look at what the 'oldest' node is that is used in the discretisation; e.g. suppose you use u_m^{n+1}, u_m^n or u_m^{n-1} somewhere in the discretisation (doesn't matter whether they appear normally or as derivatives), then the oldest node is u_m^{n-1}.
- Replace all instances of this node, including its derivatives, with 1.
- Replace all instances of the first node that occurs after this with σ_m.
- Replace all instances of the second node that occurs after this with σ_m^2.
- Replace all instances of the third node that occurs after this with σ_m^3, etc. etc.
- Rewrite the expression to the desired form.

For example, for the explicit Euler time march:

$$u_m^{n+1} = u_m^n + \Delta t \frac{du_m^n}{dt}$$

The oldest node is u_m^n, so instances of those are replaced by 1. u_m^{n+1} is replaced by σ_m, and the derivative is multiplied with λ_m, so that we get

$$\sigma_m = 1 + \Delta t \lambda_m$$

For the implicit Euler time march:

$$u_m^{n+1} = u_m^n + \Delta t \frac{du_m^{n+1}}{dt}$$

we replace u_m^n with 1, all instances of u_m^{n+1} (including the entire derivative) with σ_m and multiply the derivatives with λ_m; this gets us

$$\begin{aligned} \sigma_m &= 1 + \Delta t \lambda_m \sigma_m \\ \sigma_m &= \frac{1}{1 - \Delta t \lambda_m} \end{aligned}$$

Quiz 3: Question 9

Consider the Adams time marching scheme:

$$u^{n+1} = u^n + \Delta t \left(\frac{3}{4} \frac{du^{n+1}}{dt} + \frac{1}{4} \frac{du^{n-1}}{dt} \right)$$

If we define $T = \Delta t$, the $\lambda - \sigma$ relation of this time march is given by: ...

The oldest node is u^{n-1}; $\frac{du^{n-1}}{dt}$ is thus replaced with $\lambda \cdot 1$. u^n is replaced with σ; u^{n+1} with σ^2 and $\frac{du^{n+1}}{dt}$ with $\lambda\sigma^2$. Thus, we can write

$$\begin{aligned} \sigma^2 &= \sigma + \Delta t \left(\frac{3}{4} \lambda \sigma^2 + \frac{1}{4} \lambda \right) \\ \sigma^2 - \sigma - \Delta t \left(\frac{3}{4} \lambda \sigma^2 + \frac{1}{4} \lambda \right) &= 0 \end{aligned}$$

Easy as fuck as long as you remember my method.

Quiz 3: Question 10

Consider the second-order backward time-march:

$$u^{n+1} = \frac{1}{3} \left[4u^n - u^{n-1} + 2\Delta t \frac{du^{n+1}}{dt} \right]$$

If we define $T = \Delta t$, the $\lambda - \sigma$ relation of this time march is given by: ...

---

u^{n-1} is the oldest node, and is thus replaced by 1. u^n is replaced with σ, $\frac{du^{n+1}}{dt}$ with $\lambda\sigma^2$ and u^{n+1} with σ^2. Then we get

$$\begin{aligned} \sigma^2 &= \frac{1}{3} \left[4\sigma - 1 + 2\Delta t \lambda \sigma^2 \right] \\ \sigma^2 - \frac{1}{3} \left[4\sigma - 1 + 2\Delta t \lambda \sigma^2 \right] &= 0 \end{aligned}$$

Again, easy af.

Now, there's something to remark here. If we compare

$$w_m(t) = C_m e^{\lambda_m t} \qquad \begin{aligned} w_m^{n+1} &= \sigma_m w_m^n \\ w_m(t + \Delta t) &= \sigma_m w_m(t) \end{aligned}$$

then we'd expect

$$\sigma_m = \frac{w_m(t + \Delta t)}{w_m(t)} = \frac{C_m e^{\lambda_m (t + \Delta t)}}{C_m e^{\lambda_m t}} = e^{\lambda_m \Delta t}$$

and this is actually true to some extent: the Taylor expansion of the exponential function is $e^k = 1 + k + k^2/2! + k^3/3! + k^4/4! + ...$ So, our expansion would be

EXPANSION OF $e^{\lambda_m t}$

The expansion of $e^{\lambda_m t}$ is

$$\sigma_m \approx 1 + \lambda_m \Delta t + \frac{\left(\lambda_m \Delta t\right)^2}{2} + \frac{\left(\lambda_m \Delta t\right)^3}{6} + ... \tag{6.15}$$

Now, compare this with the $\lambda - \sigma$ relation for the explicit Euler time march, which was

$$\sigma_m = 1 + \Delta t \lambda_m$$

so the first two terms match exactly! This is no coincidence: explicit Euler is first-order accurate, and if a method is pth-order accurate, then the σ_m will approximate $e^{\lambda_m t}$ exactly for the first $p + 1$ term (so the first two terms for explicit Euler). To repeat:

APPROXIMATION OF σ_m

For a pth order accurate time discretisation, σ_m approximates $e^{\lambda_m t}$ for the first $p + 1$ terms.

Now, you may wonder, but implicit Euler in time (first order accurate) got us

$$\sigma_m = \frac{1}{1 - \Delta t \lambda_m}$$

what's up with that? Well, the binomial theorem states that $(1 + \epsilon)^{-1} = 1 - \epsilon + \epsilon^2 - \epsilon^3 + \epsilon^4 + ...$, which here would become

$$\sigma_m = \frac{1}{1 + \left(-\Delta t \lambda_m\right)} = 1 - \left(-\Delta t \lambda_m\right) + \left(-\Delta t \lambda_m\right)^2 + ... = 1 + \Delta t \lambda_m + \left(\Delta t \lambda_m\right)^2 +$$

and we see that only the first two terms are correct; after that it goes wrong. So even the implicit scheme satisfies the theorem that a first-order accurate scheme will approximate $e^{\lambda_m t}$ correctly for the first two terms.

Now, something that you may have found odd was that in the previous two examples (question 9 and 10 I mean), we had σ^2 in there. We thus have two roots; the thing is that only one of those will be the approximation for $e^{\lambda_m t}$; this is called the **non-spurious roots**. The other root(s) (in case you have even more roots) are unimportant and are called **spurious roots**.

Quiz 3: Question 11

A quadratic $\lambda - \sigma$ relation produces the following two roots:

$$\sigma_1 = \lambda_m \Delta t + \frac{\left(\lambda_m \Delta t\right)^2}{2} \quad \text{and} \quad \sigma_2 = 1 + \lambda_m \Delta t + \frac{\left(\lambda_m \Delta t\right)^2}{2}$$

Which of the following choices is true?

- Both σ_1 and σ_2 are non-spurious
- Both σ_1 and σ_2 are spurious
- Root σ_1 is spurious
- Root σ_2 is spurious

Correct is **Root σ_1 is spurious**. There will always be one and not more than one non-spurious root, so the first two options are incorrect anyway. Then, we see that the second root correctly approximates $e^{\lambda_m t}$, so σ_2 is non-spurious and thus σ_1 is spurious. It can btw be concluded that the method is second order accurate.

Then, final thing before we move on to the last section of this paragraph; consider the following MacCormack

predictor-correct scheme:

$$\begin{aligned}\tilde{u}^{n+1} &= u^n + \Delta t \frac{du^n}{dt} \\ u^{n+1} &= \frac{1}{2}\left(u^n + \tilde{u}^{n+1} + \Delta t \frac{d\tilde{u}^{n+1}}{dt}\right)\end{aligned}$$

Basically, what happens is that you make an initial guess for u^{n+1}; this initial guess is denoted by $\tilde{u}^{n+1}$. This value is then corrected by calculating the derivative at at this new location. It's basically shown in figure 6.12, although there y is used instead of u (but the rest is the same). How do we determine the $\lambda - \sigma$ relation for this one? Well, we need to get rid of all the tildes in the second equation: we just substitute the first equation in:

$$\begin{aligned}u^{n+1} &= \frac{1}{2}\left(u^n + \tilde{u}^{n+1} + \Delta t \frac{d\tilde{u}^{n+1}}{dt}\right) \\ &= \frac{1}{2}\left(u^n + u^n + \Delta t \frac{du^n}{dt} + \Delta t \frac{d\left(u^n + \Delta t \frac{du^n}{dt}\right)}{dt}\right) \\ &= \frac{1}{2}\left(2u^n + \Delta t \frac{du^n}{dt} + \Delta t \frac{du^n}{dt} + \Delta t^2 \frac{d^2u^n}{dt^2}\right) \\ &= u^n + \Delta t \frac{du^n}{dt} + \frac{\Delta t^2 \frac{d^2u^n}{dt^2}}{2}\end{aligned}$$

The earliest used node is u^n, so u^n is replaced with 1; $\frac{du^n}{dt}$ with λ_m, $\frac{d^2u^n}{dt^2}$ with λ_m^2 (since it's the second derivative); u^{n+1} is replaced with σ_m. In case there would have been $\frac{d^2u^{n+1}}{dt^2}$, you'd have replaced it with $\lambda_m^2\sigma_m$, to be clear. Thus, we get

$$\sigma_m = 1 + \lambda_m \Delta t + \frac{\left(\lambda_m \Delta t\right)^2}{2}$$

and clearly, the method is second order accurate as it exactly approximates the first three terms correctly.

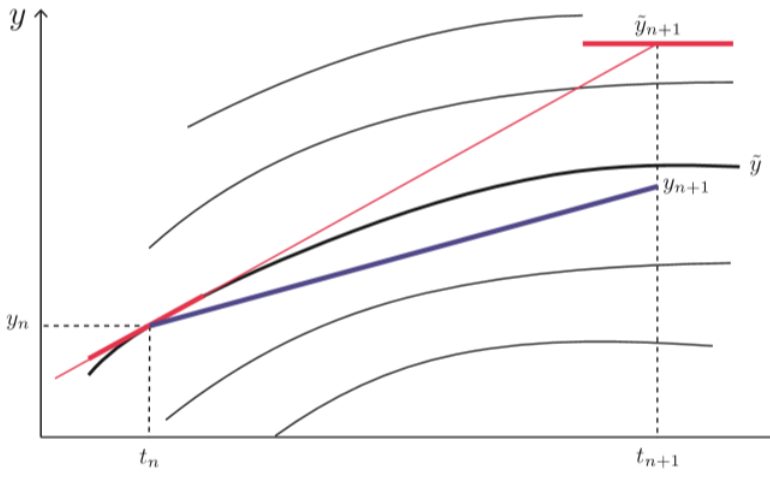


Figure 6.12: Graphical representation of the predictor-corrector in the solution space, with exact solutions plotted (block lines). A prediction is made using the forward Euler-Cauchy formula (thin red line). The solution gradient at the prediction is evaluated (thick red line). This gradient is averaged with the initial solution gradient, and on this basis a new correction step is made (blue line). The final result is much closer to the truth than the predictor.

6.4 Choosing a time march

Now, as I said before, the combination of space discretisation (which determines your eigenvalues λ_m) and the time discretisation (which determines how σ depends on λ) need to be such that $|\sigma| \leq 1$ for all eigenvalues. For example, consider an explicit Euler time march, for which we established that it was required that $\sigma_m = 1 + \Delta t \lambda_m$. In the complex plane, the circle $|\sigma_m| \leq 1$ can be drawn as shown in figure 6.13. In this figure, we also plotted the eigenvalues λ_m that you get for certain discretisations. Let's go over them one-by-one:

- For the central discretisation of the linear diffusion equation (the small circles), we see (again) that the eigenvalues lay on the negative part of the real axis. As their position scales with $\frac{\upsilon \Delta t}{\Delta x^2}$ (that means, if this quantity is halved, they also lay half so close to the origin), it's easy to imagine that there are values of Δt and Δx possible for which all eigenvalues lay within stable region.
- For the upwind discretization of the linear convection equation (the small crosses), we see that the eigenvalues all lay within the stable region; again, as the radius of circle of λ_m scales with $\frac{c \Delta t}{\Delta x}$, we can change these parameters so that the circle becomes larger or smaller (although it may not become larger than the stability region).
- For the central discretization of the linear convection equation (the small triangles), we see that there will always be eigenvalues that lay outside the stable region, so it won't be stable. However, adding an additional dissipation term will pull the λ_m to the left (it basically adds some diffusion, and we see that the eigenvalues of the diffusion equation also all lay to the left), so then it'd be stable.

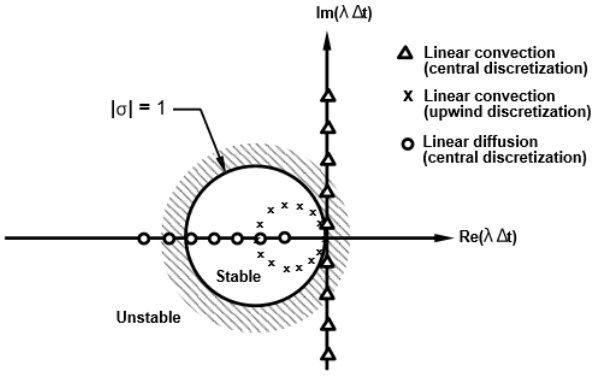
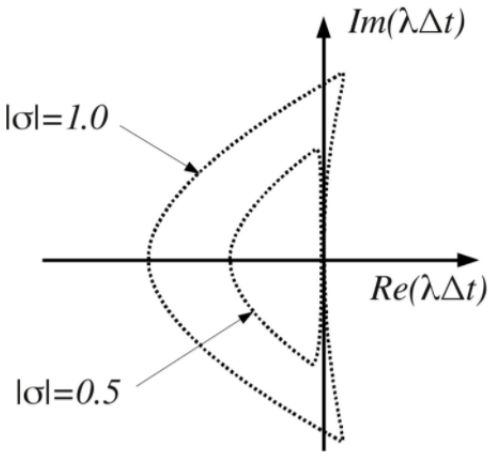


Figure 6.13: Stability region for the explicit Euler time march.

Quiz 3: Question 12

$|\sigma|$ =constant counters are shown for a time march in the figure below.



This time march is:

- Unstable for a second-order central discretisation of linear convection

- Unconditionally stable for a second-order central discretisation of linear convection
- Conditionally stable for a central discretisation of linear diffusion
- Unstable for a central discretisation of linear diffusion

Consider figure 6.13. All of the discretisations shown in figure 6.13 are conditionally stable, with the requirement that $\frac{c\Delta t}{\Delta x}$ (or $\frac{v\Delta t}{\Delta x^2}$ is sufficiently small. So, the correct answer is **Conditionally stable for a central discretisation of linear diffusion**.

6.4.1 *Implicit versus explicit time marches*

We can also draw the $|\sigma|$ contour for implicit Euler in the left part figure 6.14; please note that the region within the circle is now the *un*stable region. This case, the σ_m are far from 1 for all of previously considered eigenvalues. This may seem nice, but this essentially means that you have a very large amplitude error[7]. This is obviously also undesirable as it means that your entire solution damps out quickly while it's not supposed to happen.

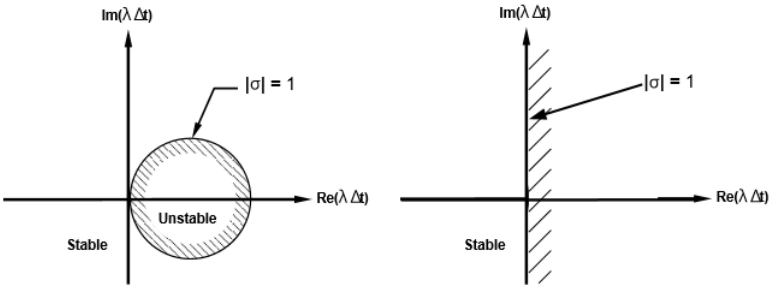


Figure 6.14: Stability regions for the Implicit Euler (left) and trapezoidal (right) time marches.

We do, however, have another implicit method, namely the second-order accurate trapezoidal time march, defined by:

$$u_i^{n+1} = u_i^n + \frac{\Delta t}{2}\left(\frac{du_i^{n+1}}{dt} + \frac{du_i^n}{dt}\right)$$

We can quickly determine the $\lambda - \sigma$ relation for this:

$$\begin{aligned} \sigma_m &= 1 + \frac{\Delta t}{2}\left(\lambda_m \sigma_m + \lambda_m\right) \\ \left(1 - \frac{\Delta t}{2}\lambda_m\right)\sigma_m &= 1 + \frac{\Delta t}{2}\lambda_m \\ \sigma_m &= \frac{1 + \frac{\Delta t}{2}\lambda_m}{1 - \frac{\Delta t}{2}\lambda_m} \end{aligned}$$

The corresponding region where $|\sigma| \leq 1$ is shown in the right part of figure 6.14; to be clear, everything to the left of (and including) the imaginary axis is in the stable region; everything to the right of it is unstable.

Implicit methods are more expensive than explicit methods, so a fair comparison is with explicit methods which are applied for more time steps, but have the same computational work. The choice between them depends on the λ_m that are used then. Typically speaking, we would want the largest $|\lambda_m|$ with the stable range of the

[7] Forgot to mention it before, but the amplitude and phase errors of a fully discrete system can also be expressed as

$$\begin{aligned} er_a &= |\rho_e| - |\sigma_m| \\ er_p &= \angle\rho_e - \angle\sigma_m \end{aligned}$$

time march, as these are normally associated with highest-frequency components of the solution; these show high phase errors and thus we want to get rid of them. However, such a time step can be much smaller than the one required to adequately resolve eigenvectors with low frequencies; these are associated with the smallest $|\lambda_m|$. These low frequency components have the disadvantages their period is rather large; for a fixed Δt, this may mean that we need to make many steps before we are able to accurately reproduce the low-frequency components. In this case, it is preferred to use implicit time marches.

An early indication of whether implicit or explicit is prefered is the stiffness of the semi-discrete system:

STIFFNESS OF SEMI-DISCRETE SYSTEM

The stiffness of the semi-discrete system is defined by

$$C_r = \frac{\max\left(|\lambda_m|\right)}{\min\left(|\lambda_m|\right)} \tag{6.16}$$

For $C_r \geq 10^4$, the system is considered "stiff", and implicit time marches should be considered.

Quiz 3: Question 13

Discretisation A produces semi-discrete eigenvalues $[-3, -4, -5, -6, -7, -8]$, while another discretisation, B, has the semi-discrete eigenvalues $[-3, -4, -5, -7, -9, -11]$.

- Discretisation B describes a stronger convective process than A
- Discretisation B describes a weaker convective process than A
- Discretisation B is less stiff than discretisation A
- Discretisation B is more stiff than discretisation A

For A, we have $C_r = 8/3 = 2.67$; for B we have $C_r = 11/3 = 3.67$. Thus, B is more stiff than A and the correct answer is **Discretisation B is more stiff than discretisation A**.

6.4.2 *Tuning a time march with free parameters*

Doubt they'll ask for this, but anyway. Runge-Kutta is basically an extension of that predictor-correcter scheme we saw before[8]:

$$\begin{aligned}
u_1^{n+1} &= u^n + \alpha^1 \Delta t \left(\frac{\partial u^n}{\partial t}\right) \\
u_2^{n+1} &= u^n + \alpha^2 \Delta t \left(\frac{\partial u_1^{n+1}}{\partial t}\right) \\
&\vdots \\
u_k^{n+1} &= u^n + \alpha^k \Delta t \left(\frac{\partial u_{k-1}^{n+1}}{\partial t}\right) \\
&\vdots \\
u^{n+1} &= u^n + \alpha^K \Delta t \left(\frac{\partial u_{K-1}^{n+1}}{\partial t}\right)
\end{aligned}$$

The associated stable regions, for several corrections K, are shown in figure 6.15. Note that this allows us to even use central in space discretisations for the linear advection equation.

Changing K and α greatly affects the stability region; for $K = 4$, contours of different $|\sigma|$ are shown for different α^k in figure 6.16. How should we determine which set of α^k we want to use? Well, in general, we want them to be such that the highest frequencies (which have a large phase error, as I mentioned in the previous subsection) are dampened; the highest frequencies are those eigenvalues with a large imaginary part. We thus want our regions where $|\sigma|$ is small to be located in those regions as well (i.e. the regions where the imaginary part is large).

[8] Please note that I do not mean α to the power 1, 2 etc. You have a list of αs; α^1, α^2, ..., α^K.

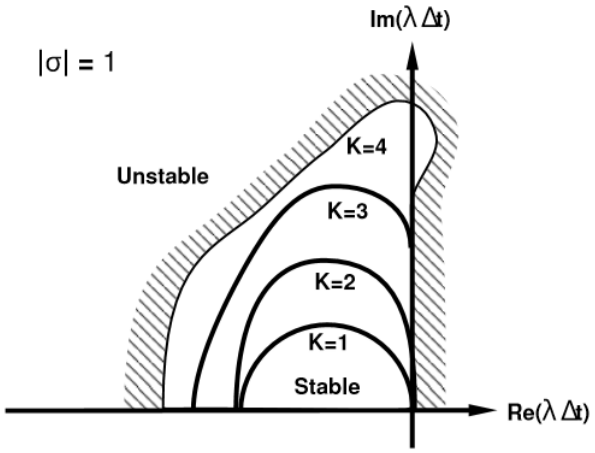


Figure 6.15: Stability regions for the low-storage Runge-Kutta time march.

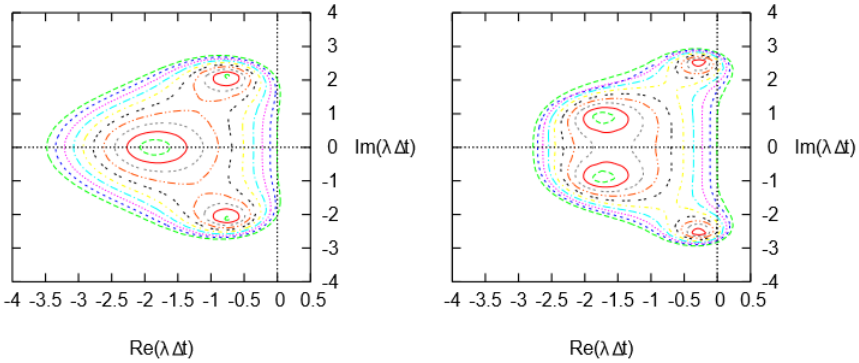


Figure 6.16: $|\sigma|$ = constant contours for four-stage low-storage Runge-Kutta time marches with different α^k. The contour values range from 0.1 to 1.0 with a 0.1 increment.

7 Iterative solvers

7.1 A brief overview of direct solvers

You already know a few methods on how to compute the solution, given some matrix equation. Let's go through them:

- Cramer's rule: this worked as follows. Suppose we have the matrix equation $[A]\mathbf{x} = \mathbf{b}$, with $[A]$ of size 5×5. Then, to find the entries of $\mathbf{x}$ (a 5×1 vector), you do the following:
  - Let $\mathbf{a}_1$, $\mathbf{a}_2$, ..., $\mathbf{a}_5$ denote the columns of $[A]$.
  - Suppose you want the second entry of $\mathbf{x}$, then this is given by substituting the second column of $[A]$ (i.e. $\mathbf{a}_2$) with $\mathbf{b}$ and calculating the determinant of the resultant matrix
  - This should then be divided by the determinant of the original matrix $[A]$; in other words, you get

$$x_2 = \frac{\det\left(\begin{bmatrix}\mathbf{a}_1 & \mathbf{b} & \mathbf{a}_3 & \mathbf{a}_4 & \mathbf{a}_5\end{bmatrix}\right)}{\det\left(\begin{bmatrix}\mathbf{a}_1 & \mathbf{a}_2 & \mathbf{a}_3 & \mathbf{a}_4 & \mathbf{a}_5\end{bmatrix}\right)}$$

  It should come at no surprise that this method is horribly inefficient; it's the fastest method for matrices smaller than 10×10, but after that, it takes forever, as it scales with $(N+1)!$. What does this scaling mean? Suppose we want to compare how much longer it takes to apply Cramer's rule for a 20×20 matrix compared to a 15×15 matrix, we get that it takes $\frac{(20+1)!}{(15+1)!} = 21 \cdot 20 \cdot 19 \cdot 18 \cdot 17$ times as long (so if it first takes 1 second, it'll now take more than 28 days. If you'd go for 21×21 matrix, it'd take almost 2 years! That's just dramatically bad.
- Gaussian elimination: you row reduce the matrix (not to echolon form), i.e. you make it

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \to \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

  You can then easily compute x_3, substitute this into the second row and calculate x_2; then you substitute both into the first row and calculate x_1. This scales with N^3, which is still rather bad but not nearly as bad as Cramer's rule; it means that if you double the size of the matrix, it'll take eight times as long to compute the solution (compared with Cramer's rule, if you'd there go from 15×15 to 30×30 and it'd take a second first, it'd now take 12.5 trillion years to get your result.
- If there are particular sparseness patterns, then Gaussian elimination can also be implemented that utilizes that. A famous example is the **Thomas algorithm** for tridiagonal systems, which applies to matrices of the form

$$\begin{bmatrix} b_1 & c_1 & & & & \\ a_1 & b_2 & c_2 & & & \\ & a_2 & b_3 & c_3 & & \\ & & a_3 & b_4 & c_4 & \\ & & & a_4 & b_5 & c_5 \\ & & & & a_5 & b_6 \end{bmatrix}$$

  Such matrices are common in second-order 1D finite-difference approximations, for example when Dirichlet or first-order Neumann boundary conditions are used. The Thomas algorithm is extremely efficient; it only scales with N. However, note that it can only be applied if the matrix is tridiagonal, if there is only one thing that fucks up then it cannot be used at all any more. Tough luck.

In this chapter, we will discuss new methods which use iteration to solve the system; it'll be shown that these scale with N^2. They are called Jacobi point/line iteration and Gauss-Seidel point/line iteration.

Quiz 3: Question 14

The ratio of work required to solve a system with a 5×5 matrix over that required to solve one with a 4×4 matrix using Cramer's rule is:

---

6. The ratio is $\frac{(5+1)!}{(4+1)!} = 6$.

Quiz 3: Question 15

A Gaussian elimination routine requires 1 second to solve a system with a 10×10 matrix. How many seconds would it take to solve one with a 20×20 matrix?

---

8. We have the ratio $\frac{20^3}{10^3} = \left(\frac{20}{10}\right)^3 = 8$, so it now takes $8 \cdot 1 = 8$ seconds to solve.

Quiz 3: Question 16

Three solvers were tested on a 10×10 linear system producing the CPU times shown in the table below:

Solver	CPU time
Cramer's rule	1
Gaussian elimination	2
Overrelaxed point Gauss-Seidel	5
Point Jacobi	8

We would like to know what solver is likely to have the lowest CPU time for a 20×20 matrix. Assuming that CPU time is a direct indication of operation count, determine the expected CPU times for the following solvers:

a) Cramer's rule:
b) Gaussian elimination:
c) Overrelaxed point Gauss-Seidel:

Let's do them one-by-one. For Cramer's rule, it's

$$1 \cdot \frac{(20+1)!}{(10+1)!} = 21 \cdot 20 \cdot 19 \cdot 18 \cdot 17 \cdot 16 \cdot 15 \cdot 14 \cdot 13 \cdot 12 = 1.2799 \cdot 10^{12}$$

You can either plug in the multiplication (not the one with factorials still in their) or the number.

For Gaussian elimination, it's simply

$$2 \cdot \frac{20^3}{10^3} = 2 \cdot \left(\frac{20}{10}\right)^3 = 16$$

For overrelaxed point Gauss-Seidel (we'll discuss later what this means), it'll be

$$5 \cdot \frac{20^2}{10^2} = 5 \cdot \left(\frac{20}{10}\right)^2 = 20$$

7.2 *Iterative solves: model problems*

Let me first define two model problems that we'll look at to explain iterative solvers. We will consider the Laplace equation in 1D and 2D. In 1D, using the mesh shown in figure 7.1, Laplace's equation is $\frac{\partial^2 u}{\partial x^2} = 0$, and

we can use the central discretisation for this (with h the mesh spacing):

$$\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} = 0 \tag{7.1}$$

In 2D, we have $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$; if we have $h = \Delta x = \Delta y$ on the mesh shown in figure 7.2, we end up at

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2} = 0 \tag{7.2}$$

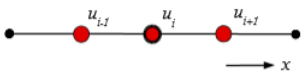


Figure 7.1: Finite-difference stencil for the 1D model problem.

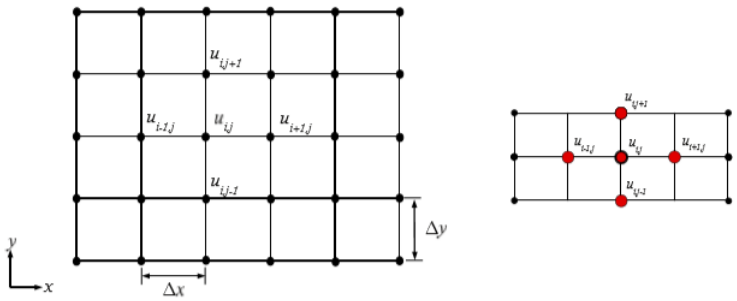


Figure 7.2: Domain and finite-difference stencil for the 2D model problem.

7.3 Point iteration methods

First we'll consider point iteration methods; later we'll discuss line iteration methods, but these are extremely similar to line iteration methods so don't worry about it yet.

7.3.1 Jacobi iteration

The simplest iterative method is Jacobi point iteration. It works as follow: you start with an initial guess for the solution. You then correct the initial guess using the discretisation. For example, we can rewrite equation (7.1) to

$$u_i^{n+1} = \frac{1}{2}\left(u_{i+1}^n + u_{i-1}^n\right)$$

In other words: we use the 'old' solution values at $i-1$ and $i+1$ to compute the updated solution at i. This process can be iterated a few times. For the Laplace equation in 2D, we can rewrite equation (7.2) to

$$u_i^{n+1} = \frac{1}{4}\left(u_{i+1,j}^n + u_{i-1,j}^n + u_{i,j+1}^n + u_{i,j-1}^n\right)$$

Note that we're basically taking the average values of the 'old' surrounding points.

I think the concept of iteration should be pretty clear: you start with an initial guess for the solution value at all points, then update it with each iteration, by using a rewritten form of the discretisation.

7.3.2 *Gauss-Seidel iterations*

Gauss and Seidel were smart fellows: if we are updating the solution in the order shown in figure 7.3, then for example for

$$u_i^{n+1} = \frac{1}{2}\left(u_{i+1}^n + u_{i-1}^n\right)$$

we already know what u_{i-1} after the new iteration will be once we start updating u_i. So why wouldn't we include it? Indeed, Gauss-Seidel iteration would look like

$$u_i^{n+1} = \frac{1}{2}\left(u_{i+1}^n + u_{i-1}^{n+1}\right)$$

Where available, you already use the updated solution values. In 2D, we get (look closely at figure 7.3):

$$u_{i,j}^{n+1} = \frac{1}{4}\left(u_{i+1,j}^n + u_{i-1,j}^{n+1} + u_{i,j+1}^n + u_{i,j-1}^{n+1}\right)$$

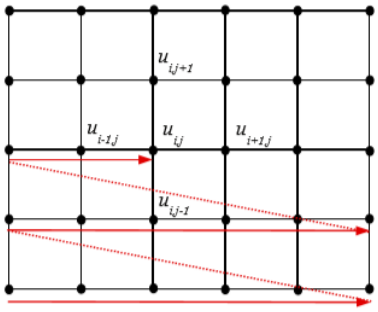


Figure 7.3: Update pattern for point iteration methods.

All of this should be pretty logical imo.

7.3.3 *Over and under relaxation*

The Jacobi and Gauss-Seidel can be seen as giving corrections to the local solution of the form $u_i^{n+1} = u_i + \Delta u_i^n$; i.e. for the 1D problem, the correction is (using Jacobi iteration):

$$\Delta u_i^n = u_i^{n+1} - u_i^n = \frac{1}{2}\left(u_{i+1}^n + u_{i-1}^n\right) - u_i^n$$

Since we expect Δu_i^n to move u_i^n in the direction of the exact solution (so that it converges), why would we not try to move a little faster by adding a little more of the correction? Or, if we keep overshooting the exact solution, why not add a little less? To provide these options, we multiply Δu_i^n by the factor ω and re-arrange to obtain

$$u_i^{n+1} = u_i^n + w\Delta u_i^n = u_i^n + \frac{\omega}{2}\left(u_{i+1}^n - 2u_i^n + u_{i-1}^n\right)$$

Note that for Gauss-Seidel iteration, we'd use u_{i-1}^{n+1} instead of u_{i-1}^n.

RELAXATION

If $\omega < 1$, we can expect slower convergence, but less tendency to overshoot. This is called **underrelaxation**.

If $\omega > 1$, we can expect faster convergence. This is referred to as **overrelaxation**.

The term **successive overrelaxation**(SOR) is used for the the fastest-convering combination, using $\omega > 1$ with the Gauss-Seidel method.

Quiz 3: Question 17

A finite difference approximation for $\frac{\partial u}{\partial x} + \frac{\partial^2 u}{\partial x^2} = f$ is defined by:

$$\frac{u_i - u_{i-1}}{h} + \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} = f_i$$

Assuming we proceed from smaller to larger values in i, the point Gauss-Seidel update formula is given by:

$$u_i^{n+1} =$$

First, the Jacobi-iteration would be derived as follows:

$$\begin{aligned}
\frac{u_i - u_{i-1}}{h} + \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} &= f_i \\
hu_i^{n+1} - hu_{i-1}^n + u_{i+1}^n - 2u_i^{n+1} + u_{i-1}^n &= f_i h^2 \\
(h-2)\,u_i^n &= f_i h^2 + hu_{i-1}^n - u_{i+1}^n - u_{i-1}^n \\
u_i^{n+1} &= \frac{f_i h^2 + hu_{i-1}^n - u_{i+1}^n - u_{i-1}^n}{h-2}
\end{aligned}$$

However, as we would already have u_{i-1}^{n+1}, this becomes

$$u_i^{n+1} = \frac{f_i h^2 + hu_{i-1}^{n+1} - u_{i+1}^n - u_{i-1}^{n+1}}{h-2}$$

Do check out in Maple how to plug this in, this is not something you'd wanna discover on your exam.

7.4 Analysis of iterative methods

Let's analyse first the stability of the iteration, and secondly how fast it converges.

7.4.1 Stability analysis

Consider again the Jacobi method including relaxation:

$$u_i^{n+1} = u_i^n + \omega \Delta u_i^n = u_i^n + \frac{\omega}{2}\left(u_{i+1}^n - 2u_i^n + u_{i-1}^n\right)$$

Let's determine the behaviour of the error, $\epsilon_i^n = u_i^n - u_i^*$, where u_i^* is the final converged solution to the system of equations. If we substitute in $u_i^n = \epsilon_i^n + u_i^*$, we get[1]

$$\epsilon_i^{n+1} = \epsilon_i^n + \frac{\omega}{2}\left(\epsilon_{i+1}^n - 2\epsilon_i^n + \epsilon_{i-1}^n\right)$$

Again, we assume the form of the error

$$\epsilon_i^n = E\rho^n e^{Ikx}$$

[1] u_i^* disappears completely: if you iterate u_i^*, you get u_i^* again; this follows from the definition of the final converged solution. Thus, although you would actually get

$$\epsilon_i^{n+1} + u_i^* = \epsilon_i^n + u_i^* + \frac{\omega}{2}\left(\epsilon_{i+1}^n + u_{i+1}^* - 2\left(\epsilon_i^n + u_i^*\right) + \epsilon_{i-1}^n + u_{i-1}^*\right)$$

since $u_{i+1}^* + u_{i-1}^* = 2u_i^*$, this completely reduces to the equation shown.

wher k is the wave number of the error component being considered. We thus get

$$E\rho^{n+1}e^{Ikx} = E\rho^n e^{Ikx} + \frac{\omega}{2}\left(E\rho^n e^{Ik(x+\Delta x)} - 2E\rho^n e^{Ikx} + E\rho^n e^{Ik(x-\Delta x)}\right)$$

Divide by $E\rho^n e^{Ikx}$ and write $\beta = \Delta x$ to get

$$\rho = 1 + \frac{\omega}{2}\left(e^{I\beta} - 2 + e^{-I\beta}\right) = 1 + \omega(\cos(\beta) - 1)$$

This amplification factor represents the ratio of the new error over the old error $\epsilon_i^{n+1}/\epsilon_i^n$. Obviously, this must be less than 1 for convergence.

FINDING THE AMPLIFICATION FACTOR OF THE ERROR

Starting with a given finite-difference operator:

1. Rewrite the scheme to be an explicit formula for $u_{i,j}^{n+1}$.
2. Let all terms be of the form $u_{i+p,j+r}^{n+q}$, where p, q and r are both integers (possibly negative or zero).
3. Each of those terms must be substituted with $\rho^r e^{Ik_x \cdot p\Delta x} e^{Ik_y \cdot q \cdot \Delta y}$.
4. Rewrite to obtain an explicit expression for ρ if desired.

We can plot ρ for several values of ω in the left plot of figure 7.4. Please note that the interpretation of these plots is vastly different from before. Before, we wanted ρ to be as close as possible to ρ_e. Now, we simply want the amplification factor to be as small as possible, so that the error dampens out as quickly as possible.

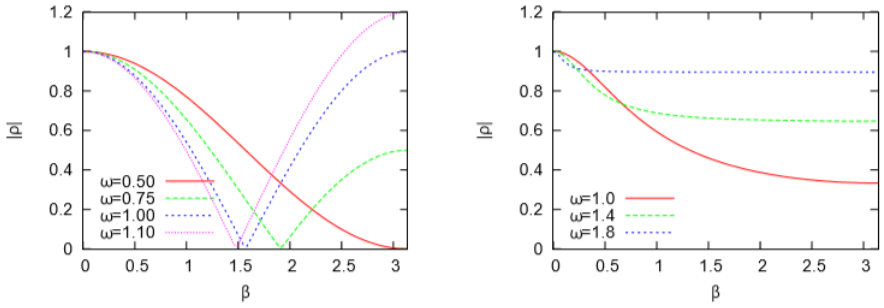


Figure 7.4: Amplification factors for the relaxed point-Jacobi (left) and Gauss-Seidel (right) methods with different values of ω.

Looking at the left plot of figure 7.4, we realize that overrelaxation is not an option for the point-Jacobi method, because then $|\rho| > 1$ so the iteration would diverge. In fact, a slight amount of underrelaxation would clean up the high-wave number components of the error; however, you cannot go for too low a underrelaxation as $|\rho|$ for the lowest wavenumbers (low values of β) increases as the relaxation is decreased.

For the Gauss-Seidel method,

$$u_i^{n+1} = u_i^n + \frac{\omega}{2}\left(u_{i+1}^n - 2u_i^n + u_{i-1}^{n+1}\right)$$

we get

$$\begin{aligned}
\rho &= 1 + \frac{\omega}{2}\left(e^{Ik\Delta x} - 2 + \rho e^{-Ik\Delta x}\right) \\
\left(1 - \frac{\omega}{2}e^{-Ik\Delta x}\right)\rho &= 1 + \frac{\omega}{2}\left(e^{Ik\Delta x} - 2\right) \\
\rho &= \frac{1 + \frac{\omega}{2}\left(e^{I\beta} - 2\right)}{1 - \frac{\omega}{2}e^{-I\beta}} = \frac{2 + \omega(\cos(\beta) + I\sin(\beta) - 2)}{2 - \omega(\cos(\beta) - I\sin(\beta))}
\end{aligned}$$

The magnitude of ρ is plotted in the right plot of figure 7.4; in this case the basic $\omega = 1$ method has good high-wavenumber error reduction, while the low-wavenumber error reduction can be improved by increasing ω. Once ω exceeds 2, however, the method will become unstable.

Quiz 3: Question 18

An iterative solver has the following amplification factor:

$$\rho = 1 - \frac{\omega}{3}\sin(\beta)$$

What is the maximum relaxation factor ω that can be used?

We must have that $-1 \leq \rho \leq 1$ for $\beta \in [0, \pi]$. This means that $0 \leq \frac{\omega}{3}\sin(\beta) \leq 2$ for $\beta \in [0, \pi]$. It should be obvious that this is true for $0 \leq \omega \leq 6$, so 6 is the maximum relation factor ω that can be used.

Quiz 3: Question 19

Consider the following approximation for $u - c\frac{\partial u}{\partial x} = 0$ on a periodic domain:

$$u_i - c\frac{u_{i+1} - u_{i-1}}{2h} = 0$$

Defining $I = \sqrt{-1}$ and $b = k_m \Delta x$, the amplification factor of a point Jacobi method as a function of b equals $\rho = ...$

First, using point Jacobi, we'd get the following iterative formula:

$$u_i^{n+1} = c\frac{u_{i+1}^n - u_{i-1}^n}{2h}$$

Using the problem solving guide, this becomes

$$\rho = c\frac{e^{Ik\Delta x} - e^{-Ik\Delta x}}{2h} = c\frac{e^{Ib} - e^{-Ib}}{2h} = \frac{cI\sin(b)}{h}$$

Note that you may also just plug in the exponentials (once you've substituted b instead of $k\Delta x$ of course.

Quiz 3: Question 21

Consider a point Jacobi iteration scheme for u_i, derived from the following finite-difference operator:

$$\frac{u_{i+1} - u_{i-1}}{2h} + \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} = 0$$

Defining $I = \sqrt{-1}$ and $b = k_m \Delta x$, the amplification factor of this scheme (as a function of b) equals:

$$\rho =$$

First, we write it to an explicit expression for u_i^{n+1}:

$$\begin{aligned} \frac{h}{2} \cdot \left(u_{i+1}^n - u_{i-1}^n\right) + u_{i+1}^n - 2u_i^{n+1} + u_{i-1}^n &= 0 \\ u_i^{n+1} &= \frac{h}{4} \cdot \left(u_{i+1} - u_{i-1}\right) + u_{i+1}^n + u_{i-1}^n \end{aligned}$$

Then, use the problem solving guide to write

$$\rho = \frac{h}{4} \cdot \left(e^{Ik_m\Delta x} - e^{-Ik_m\Delta x}\right) + \frac{e^{Ik_m\Delta x} + e^{-Ik_m\Delta x}}{2} = \frac{h}{4} \cdot \left(e^{Ib} - e^{-Ib}\right) + \frac{e^{Ib} + e^{-Ib}}{2}$$

7.4.2 *Estimating the rate of convergence*

With every iteration, the magnitude of the error $|\epsilon|$ at a given wavenumber is reduced by a factor ρ. Therefore, after n iterations we can expect

$$\left|\epsilon_i^n\right| = \rho^n \left|\epsilon_i^o\right|$$

where $|\epsilon_i^o|$ is the initial error. In practice, we are interested in how many order of magnitude m the error is reduced after n iterations, i.e.

$$\frac{\left|\epsilon_i^n\right|}{\left|\epsilon_i^o\right|} = 10^{-m} = \rho^n$$

which can be rewritten to

$$\begin{aligned} -m\ln(10) &= n\ln(\rho) \\ \frac{m\ln(10)}{n} &= -\ln(\rho) = \ln\left(\frac{1}{|\rho|}\right) \end{aligned}$$

and this is how the rate of convergence is defined:

RATE OF CONVERGENCE

The rate of convergence is defined as

$$R_c = \frac{m\ln(10)}{n} = \ln\frac{1}{|\rho|} \tag{7.3}$$

It can be quite a bitch to derive the formula for R_c, although ρ is very easy to determine.

The first thing you have to realize that a good approximation requires low error at the lowest wavenumbers (these correspond to the largest features in the solution[2]). Furthermore, as visible from figure 7.4, $|\rho|$ tends to be largest for low-frequency components. Therefore, in subsequent calculations, we will assume β to be small, which allows for some Taylor expansions.

Consider the rate of convergence of the 2D Laplace operator with Jacobi iteration:

$$u_{i,j}^{n+1} = \frac{1}{4}\left(u_{i+1,j}^n + u_{i-1,j}^n + u_{i,j+1}^n + u_{i,j-1}^n\right)$$

Again, we write

$$\rho = \frac{1}{4}\left(e^{Ik_x\Delta x} + e^{-Ik_x\Delta x} + e^{Ik_y\Delta y} + e^{-Ik_y\Delta y}\right)$$

which can be rewritten as (using $\beta_x = k_x\Delta x$ and $\beta_y = k_y\Delta y$):

$$\rho = \frac{1}{2}\left(\cos\left(\beta_x\right) + \cos\left(\beta_y\right)\right)$$

With β_x and β_y small, we can use the Taylor expansion

$$\cos(\beta) = 1 - \frac{\beta^2}{2!} + \frac{\beta^4}{4}! - \ldots$$

Let's include the first two terms of this expansion, so that we get

$$\rho = \frac{1}{2}\left(1 - \frac{\beta_x^2}{2} + 1 - \frac{\beta_y^2}{2}\right) = 1 - \frac{\beta_x^2 + \beta_y^2}{4}$$

[2] If you don't get what I mean: low-frequency components determine the shape of the solution far more than high-frequency components as those are more like little details in the solution. If you'd plot the solution, and you'd walk a bit back (or squish your eyes) then you see the general shape of the solution as formed by the low-frequency waves; you don't really see the small details that high-frequency oscillations form.

We thus have a rate of convergence

$$R_c = \ln\left(\frac{1}{1-\frac{\beta_x^2+\beta_y^2}{4}}\right)$$

but this is an ugly as fuck expression. Thus, let's first use the expansion

$$\frac{1}{1-x} = 1 + x + x^2 + ...$$

and let's include only the first two terms of this; this means we get

$$R_c = \ln\left(\frac{1}{1-\frac{\beta_x^2+\beta_y^2}{4}}\right) = \ln\left(1+\frac{\beta_x^2+\beta_y^2}{4}\right)$$

Then, we use the Taylor series expansion

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3}$$

which we again will restrict ourselves to only the first term:

$$R_c = \ln\left(1+\frac{\beta_x^2+\beta_y^2}{4}\right) \approx \frac{\beta_x^2+\beta_y^2}{4} = \frac{h^2\left(k_x^2+k_y^2\right)}{4}$$

where we assume $h = \Delta x = \Delta y$ (so uniform spacing).

FINDING AN EXPRESSION FOR THE RATE OF CONVERGENCE

Starting with a finite-difference operator:

1. Obtain an explicit expression for ρ, using the previous problem solving guide.
2. Replace all complex exponentials in the expression of ρ with sines and cosines.
3. Make use of the following expansions:

$$\begin{aligned}\cos(x) &= 1 - \frac{x^2}{2!} + \frac{x^4}{4!} + ... \\ \sin(x) &= x - \frac{x^3}{3!} + \frac{x^5}{5!} + ...\end{aligned}$$

   For the cosine, only use the first two terms in this expansion; for the sine, only use the first term in this expansion.
4. Write out

$$R_c = \ln\frac{1}{\rho}$$

   Use the expansion:

$$\frac{1}{1-x} = 1 + x + x^2 + ...$$

   Only use the first two terms in this expansion.
5. Rewrite the remaining natural logarithm by using the expansion

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} + ...$$

   Only use the first term in this expansion. You have now found your rate of convergence.

Please note that in general, for each expansion, you only use up until (including) the first term that uses x; this is why $\cos(x)$ and $1/(1-x)$ require the inclusion of two terms but $\sin(x)$ and $\ln(1+x)$ only one.

Let's analyse the effect of decreasing our mesh spacing h by 2:

- This will introduce four times as many nodes in our stencil, meaning we get four times as many unknowns.
- This will *reduce* the rate of convergence by a factor 4: indeed, if we decrease our mesh spacing by a factor 2, you'll need 4 times as many iterations to get results of the same accuracy.
- For each iteration, since we have four times as many unknowns, you need to evaluate 4 times as many equations.

In other words: if we increase the number of unknowns by a factor n, then the required computational effort will be n^2 as much[3]; in other words, the work required for Jacobi iteration scales with N^2, with N the number of unknowns. This is worse than the Thomas algorithm, but Jacobi iteration can always be applied and scales better than Gaussian elimination. Furthermore, the Gauss-Seidel, and especially overrelaxed Gauss-Seidel methods can provide much faster convergence for a given N than the point Jacobi method, although these also scale with N^2.

Quiz 3: Question 22

Which of the following is not true?

- The converged solution u_i^* satisfies the update equation
- The error ϵ_i^n satisfies the update equation
- The rate of convergence is normally determined by the highest wavenumbers
- Overrelaxation is used to increase the convergence rate

The correct answer is **The rate of convergence is normally determined by the highest wavenumbers**. Let's go through the statements one-by-one. The converged solution indeed satisfies the update equation. The error also satisfies the update equation. Overrelaxation is indeed used to increase the convergence rate. However, it is not true that the rate of convergence is determined by the highest wavenumbers; quite the opposite in fact.

Quiz 3: Question 23

An iterative solver has the amplification factor $\rho = 1 - \sin(\beta)$. Given the following expansions:

$$\begin{aligned} \cos(x) &= 1 - \frac{x^2}{2!} + ... \\ \sin(x) &= x - \frac{x^3}{3!} + ... \\ \frac{1}{1-x} &= 1 + x + x^2 + ... \end{aligned}$$

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3}$$

an approximation for the solver's convergence rate R_c (as funtion of b) is:

Let's use the problem solving guide for this. First, we use only the first term of the expansion of $\sin(x)$ to write

$$\rho = 1 - \sin(b) = 1 - b$$

so that

$$R_c = \ln \frac{1}{|\rho|} = \ln \frac{1}{1-b}$$

Then, we use the first two terms of the expansion for $1/(1-x)$ to write

$$R_c = \ln \frac{1}{1-b} = \ln(1+b)$$

[3] In above example: we introduce 4 times as many unknowns. However, we must then perform 4 times as many iterations, and for each iteration, 4 times as many equations need to be solved (thus you need 4^2 the number of computations compared to before). Clearly, the relation is n^2.

Finally, we use the first term of the expansion for $\ln(1+x)$ to write

$$R_c = \ln(1+b) = b$$

7.5 Line iteration methods

Line iteration basically relies on the concept of updating all points on a line simultaneuously. In 2D, this can either be lines along constant j or lines along constant i, as shown in figure 7.5.

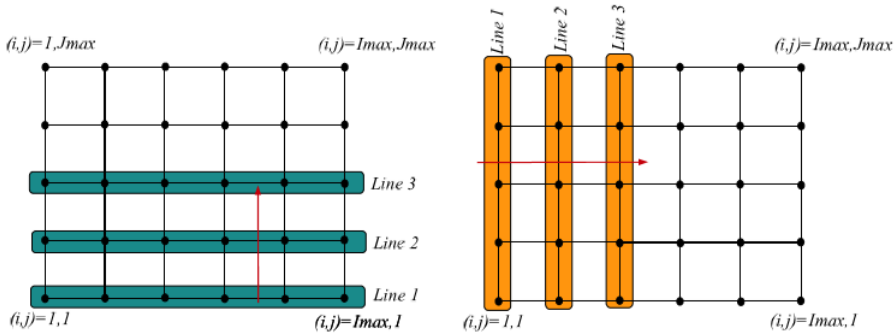


Figure 7.5: Line iteration methods. Updates with j = constant lines (left) and with i = constant lines (right).

How does this look like practically? Let's take a loot at that.

7.5.1 Line Jacobi

Consider the j = constant sweep shown in the left of figure 7.5. Originally, we would have had

$$u_{i,}^{n+1} = \frac{1}{4}\left(u_{i+1,j}^{n} + u_{i-1,j}^{n} + u_{i,j+1}^{n} + u_{i,j-1}^{n}\right)$$

However, now we'll update $u_{i+1,j}$ and $u_{i-1,j}$ simultaneously as well. Thus, we actually get the system of equations

$$u_{i+1,j}^{n+1} - 4u_{i,j}^{n+1} + u_{i-1,j}^{n+1} = -\left(u_{i,j+1}^{n} + u_{i,j-1}^{n}\right), \qquad i = 2, Imax - 1$$

since at the boundaries we'll need different equations (since you'd then have $u_{-1,j}$ appearing etc., which does not exist). Thus, if we impose 2 boundary conditions, we get a system of $Imax$ equations; this system will be tridiagonal (as long as the boundary conditions are either Dirichlet or first-order Neumann) and thus can be solved very quickly. One then proceeds to the next line, and once all lines are done, you start over. You could use a method which first updates sets of lines with j is constant, then sets of lines with i is constant; these are known as **alternating direction implicit** (ADI) **methods** .

7.5.2 Line Gauss-Seidel

Line Gauss-Seidel is similarly modified. First we'd have the equation

$$u_{i,j}^{n+1} = \frac{1}{4}\left(u_{i+1,j}^{n} + u_{i-1,j}^{n+1} + u_{i,j+1}^{n} + u_{i,j-1}^{n+1}\right)$$

However, we will now update $u_{i-1,j}$ and $u_{i+1,j}$ simultaneously with $u_{i,j}$, so actually we should write

$$\widetilde{u_{i+1,j}^{n+1}} - 4\widetilde{u_{i,j}^{n+1}} + \widetilde{u_{i-1,j}^{n+1}} = -\left(u_{i,j+1}^{n} + u_{i,j-1}^{n+1}\right), \qquad i = 2, Imax - 1$$

where again boundary conditions are needed for $i = 1$ and $i = Imax$, leading to a tridiagonal matrix equation which can be solved very efficiently. Why the wide tildes? Because we can implement the overrelaxation as follows:

$$u_{i,j}^{n+1} = u_{i,j}^{n} + \omega \left(\widetilde{u_{i,j}^{n+1}} - u_{i,j}^{n} \right) \qquad i = 2, Imax - 1$$

This can be combined with an ADI approach for maximum effectiveness.

Quiz 3: Question 20

Consider the following finite-difference approximation for the Laplace equation:

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2} = 0$$

When we use a line Gauss-Seidel iterative method to solve for $u_{i,j}^{n+1}$ (considering j = constant lines and with the matrix elements increasing from smaller to larger values in i and j), the updating scheme looks like

$$4u_{i,j}^{A} = u_{i+1,j}^{B} + u_{i-1,j}^{C} + u_{i,j+1}^{D} + u_{i,j-1}^{E}$$

with time levels (A, B, C, D, E) equal to ...

When we're updating the line j, we already know the solution value $u_{i,j-1}^{n+1}$. Furthermore, as we are updating the line j, we will simultaneously update $u_{i,j}^{n+1}$, $u_{i+1,j}^{n+1}$ and $u_{i-1,j}^{n+1}$ too. The only 'old' solution that we are using is $u_{i,j+1}^{n}$. Thus, from comparison, we see that $A = n+1$, $B = n+1$, $C = n+1$, $D = n$ and $E = n+1$. This means you have to answer (n+1,n+1,n+1,n,n+1).

Quiz 3: Question 24

The following questions concern the matlab code segment (below) for the iterative solution of Laplace's equation on a Cartesian mesh with dimensions imax×jmax, using a second-order central finite-difference operator and Dirichlet conditions on all boundaries.

```
67  while( delta > tolerance )
68
69     it = it + 1;
70
71     for j = 2:jmax-1
72       for i = 2:imax-1
73
74          % load indices
75          ij  = imax*(j-1)+i;
76          ip1j = ij+1;
77          im1j = ij-1;
78          ijp1 = ij+imax;
79          ijm1 = ij-imax;
80
81          % Update value
82          delu(ij) = (un(ip1j)+un(im1j)+un(ijp1)+un(ijm1))/4 - un(ij);
83          unp1(ij) = un(ij) + om*delu(ij);
84
85       end
86     end
87
88     delta=norm(delu);
89     un = unp1;
90
91  end
```

1. The update method defined by the code above can be best described as:

- Line Jacobi
- Point Jacobi
- Point Gauss Seidel
- Line Gauss Seidel

2. (second part)
   - with relaxation
   - without relaxation
3. Storage of the solution occurs in:
   - unp1
   - im1j
   - ip1j
   - delu
4. order in groups off all
   - i
   - j

   values for a given
   - i
   - j
5. The iteration procedure stops when
   - all of the interior points are updated
   - a measure of the solution is below a specified value
   - a measure of changes of the solution is below a specified value
   - the number of iterations reaches a maximum value

It's a bit ugly code, I agree, but the answers are pretty straightforward. First of, regarding how the variables are called: ip1j for example means i plus 1, j (i.e. $i+1, j$). im1j means $i-1, j$. $unp1$ means u^{n+1}, i.e. the solution vector at the next iteration. Now let's discuss the answers.

First of all, it's clearly point iteration as we don't see any matrix solvers applied. Then the question remains, point Jacobi or point Gauss-Seidel? It's point Jacobi: in line 82, we only take entries from the vector un; this vector is only updated at the very end of the while-loop. This means that apparently, we will not be using solution values that were updated just before (which would be what Gauss-Seidel is). You'd have to write un(ip1j)+unp1(imj)+un(ijp1)+unp1(ijm1) in line 82; this would correspond to

$$\Delta u_{i,j} = \frac{u^n_{i+1,j} + u^{n+1}_{i-1,j} + u^n_{i,j+1} + u^{n+1}_{i,j-1}}{4} - u^n_{i,j}$$

which would be the Gauss-Seidel point iteration. However, the correct answer to the first question is **point Jacobi**. From line 83, you can clearly see that this is **with relaxation**.

The storage of the solution occurs in unp1 (this is u^{n+1}); they are ordered in groups of i for a given j (that means, the vector basically looks like

$$u^{n+1} = \begin{bmatrix} u^{n+1}_{2,2} \\ u^{n+1}_{3,2} \\ u^{n+1}_{4,2} \\ \vdots \\ u^{n+1}_{2,3} \\ u^{n+1}_{3,3} \\ u^{n+1}_{4,3} \\ \vdots \end{bmatrix}$$

i.e. you first increase i; only once you've finished doing that you update j. This is visible from how the for-loops are run. Note that due to the Dirichlet boundary conditions, we are only updating the interior nodes and not the nodes at the boundary (Dirichlet boundary conditions mean that the exact value of

those are already known, so they don't need to be iterated every time).

Finally, the iteration procedure stops when **a measure of change of the solution is below a specified value**; compare line 67 with line 88.

Index