Contents

List of Figures

List of Tables

Part I

Introduction to Computational Mechanics

Chapter 1

Introduction

Arguments covered in this course (roughly):

- Numerical methods for ODEs.

- Rigid-body motion with applications to Nav. Eng. (G. Bulian).
- Network models.
- PDES: hyperbolic, elliptic, parabolic (with applcations).
- Numerical techniques specific for Fluid Mechanics.
- Hands-on applications with STAR-CCM+
- Introduction to numerical techniques for Structure Mechanics.

Part II

Ordinary differential equations

Chapter 2

Introduction to ODEs

A nice course on ODEs: https://ocw.mit.edu/courses/18-03-differential-equations-spring-2010/video_galleries/video-lectures/

Chapter 3

First-order ODE's

3.1 Linear first-order ODE's

$$u_x + a(x)\,u = b(x) \tag{3.1}$$

3.1.1 The integrating factor

Definition 3.1.1 (Integrating factor) *Integrating factor $M(x)$: any expression that a differential equation is multiplied by to facilitate integration.*

Key idea: convert equation (3.1) into separable form using an integrating factor. Thus, we look for $M(x)$ such that:

$$M(x)\,u_x \,+\, M(x)\,a(x)\,u \,=\, \left[M(x)\,u\right]_x$$

Expanding it is readily found:

$$\frac{M_x}{M} \,=\, a(x) \,\Longrightarrow\, M(x) \,=\, C\,e^{\int a(x)\,\mathrm{d}x}$$

The constant C is arbitrary, thus let it be 1:

$$M(x) = e^{\int a(x)\,\mathrm{d}x} \tag{3.2}$$

Equation (3.1) is converted into:

$$[M(x)\,u]_x = M(x)\,b(x) \Longrightarrow u(x) = \frac{1}{M}\int M(x)\,b(x)\,\mathrm{d}x + C' \tag{3.3}$$

The constant C' is to be determined using an assigned initial condition.

Example 3.1.1

$$y' - 2\,\frac{y}{x} = 0$$

Solve separating variables:

$$y(x) = C\,x^2$$

Solve with integrating factor:

$$M(x) = e^{\int -2/x\,dx} = \frac{1}{x^2}$$

$$[M\,y]' = 0 \Longrightarrow y = \frac{C}{M} = C\,x^2$$

3.1.2 Homeworks / Class Exercises

Solve by the method of the integrating factor

A small metallic sphere (Stainless steel AISI 316, $r = 5$mm), initially in thermal equilibrium at $T(0) = 250°$C, is placed in a quenching bath at $T_w = 20°$C. The convection coefficient on the surface

of the sphere is $h = 35$ W/m^2 K. A uniform, time-dependent heat generation $g(t)$ [W/m^3]takes place within the sphere by electric Joule effect, with $g(t) = 3$ W/m^3. Compute the temperature distribution as function of time using the method of the integrating factor. In developing the physical model, assume that the temperature is uniform within the sphere.

Chapter 4

Linear homogeneous ODEs with constant coefficients

DA FARE....

Chapter 5

Non-homogeneous linear ODEs: variation of parameters

We aim to find a particular solution of a non-homogeneous, linear ODE. The method of *variation of parameters* is introduced considering a non-homogeneous, second-order linear ODE:

Assume we know a basis of the solution space of the associated homogeneous ODE. Two basis functions are $y_1(t)$ and $y_2(t)$. Let's look for a particular tentative solution in the form:

$$y(t) = u_1(t)\, y_1 + u_2(t)\, y_2 \tag{5.2}$$

where u_1 and u_2 are two undetermined functions, to be found. Substituting (5.2) in (5.1) results is:

$$p(t)\, [u_1(t)\, y_1 + u_2(t)\, y_2]'' + q(t)\, [u_1(t)\, y_1 + u_2(t)\, y_2]' + r(t)\, [u_1(t)\, y_1 + u_2(t)\, y_2] = g(t) \tag{5.3}$$

In order to simplify the calculations let's introduce a further constraint on the undetermined functions u_1 and u_2:

$$[u_1(t)\, y_1 + u_2(t)\, y_2]' = u_1(t)\, y_1' + u_2(t)\, y_2' \Longrightarrow u_1'\, y_1 + u_2'\, y_2 = 0 \tag{5.4}$$

Notice that there is no a-priori reason to believe that (5.4) indeed works: we just try and see what happens.

Using (5.4):

$$\begin{aligned} y' &= u_1\, y_1' + u_2\, y_2' \\ y'' &= u_1'\, y_1' + u_1\, y_1'' + u_2'\, y_2' + u_2\, y_2'' \end{aligned} \tag{5.5}$$

Using (5.5) in (5.3), gathering terms containing u_1 and u_2 and excising the same terms by recognizing that they multiply vanishing factors (since y_1 and y_2 are solutions of the associated homogeneous equation) we are left with:

$$p\,[u_1'\, y_1' + u_2'\, y_2'] = g \tag{5.6}$$

Defining $h(t) := g(t)/p(t)$ we get (5.6) in the form:

$$u_1' y_1' + u_2' y_2' = h \tag{5.7}$$

Let's solve (5.4) and (5.7) for u_1' and u_2':

$$u_1' = -\frac{y_2 h}{W(y_1, y_2)} \tag{5.8}$$

$$u_2' = \frac{y_1 h}{W(y_1, y_2)} \tag{5.9}$$

where the Wronskian $W(y_1, y_2)$[1] is non-zero since y_1 and y_2 are linearly independent by assumption. Integrating (5.8) and (5.9) yields a particular solution of (5.1):

$$y(t) \;=\; -\int \frac{y_2\, h}{W}\,\mathrm{d}t\, y_1 \;+\; \int \frac{y_1\, h}{W}\,\mathrm{d}t\, y_2 \;-\, c\, y_1 \;+\; k\, y_2 \tag{5.10}$$

where c and k are constants of integration. Once substituted into the differential equation, the term involving the constants c and k does not bear any contribution and vanish identically. As a

[1]Recall that the Wronskian associated with a homogeneous n-th order linear ODE is defined as

$$W(y_1, \ldots, y_n) \;:=\; \det \begin{bmatrix} y_1 & y_2 & y_n \\ y_1' & y_2' & y_n' \\ \cdots\cdots\cdots\cdots & & \\ \cdots\cdots\cdots\cdots & & \\ \cdots\cdots\cdots\cdots & & \\ y_1^{(n-1)} & y_2^{(n-1)} & y_n^{(n-1)} \end{bmatrix}$$

consequence, we may safely assume $c = k = 0$, yielding

$$y(t) = -\int \frac{y_2\, h}{W}\, \mathrm{d}t\, y_1 + \int \frac{y_1\, h}{W}\, \mathrm{d}t\, y_2 \tag{5.11}$$

Example 5.0.1

$$a\, y' = -c\, y + g(t)$$

$$y_1 = e^{-c\, t/a}$$

$$y = u_1(t)\, y_1$$

$$u_1'\, y_1 + u_1\, y_1' = -\frac{c}{a}\, u_1\, y_1 + \frac{g}{a}$$

$$u_1' \, e^{-ct/a} \;=\; \frac{g}{a} \;\Longrightarrow\; u_1(t) \;=\; \int e^{ct/a} \, \frac{g(t)}{a} \, dt \;+\; C$$

Assume $C \,=\, 0$ *to get the sought particular solution:*

$$y(t) \;=\; \frac{1}{a} \, e^{-ct/a} \int g(t) \, e^{ct/a} \, dt$$

Let's now face the problem of finding a particular solution of an n-th order linear ODE:

$$y^{(n)} \,+\, \sum_{j=0}^{n-1} a_j(t) \, y^{(j)} \;=\; b(t) \tag{5.12}$$

Let $y_1(t)$, $y_2(t)$, $\ldots$, $y_n(t)$ form a basis of the solution space of the associated homogeneous ODE.

Let's look for a tentative solution in the form:

$$y(t) = \sum_{j=1}^{n} u_j(t)\, y_j \tag{5.13}$$

where the u_j's are the unknown (function) parameters.

It is readily verified that the calculatiomns yielding the derivatives of y up to order n can be greatly simplified provided we assume

$$\sum_{j=1}^{n} u_j'\, y_j^{(k)} = 0 \quad \text{for } k = 0, \ldots, n-2 \tag{5.14}$$

With these assumptions, we are left with

$$y^{(k)} = \sum_{j=1}^{n} u_j \, y_j^{(k)} \quad \text{for } k = 0, \ldots, n-1 \tag{5.15}$$

and

$$y^{(n)} = \sum_{j=1}^{n} u_j' \, y_j^{(n-1)} + \sum_{j=1}^{n} u_j \, y_j^{(n)} \tag{5.16}$$

Substituting in the ODE, while recognizing that the terms including $u_j(t)$ add to zero since the y_j's are solutions of the associated homogeneous problem, yields:

$$\sum_{j=1}^{n} u_j' \, y_j^{(n-1)} = b(t) \tag{5.17}$$

Equations (5.14) and (5.17) form a system of n equations in the n unknowns u'_j, $j = 1, \ldots, n$:

$$\boldsymbol{W}\,\boldsymbol{u}' = \boldsymbol{b} \tag{5.18}$$

where $\boldsymbol{W}$ denotes the Wronskian matrix, $\boldsymbol{u}'$ denotes the column vector of the first derivatives of the unknown parameters, $\boldsymbol{b}$ denotes the column vector with all zero components but the last one that is $b(t)$. Using Cramer's rule, the solution of this system returns:

$$u'_j = \frac{W_j}{W} \tag{5.19}$$

where W denotes the Wronskian, i.e. the determinant of the Wronskian matrix, while W_j denotes the determined of the matrix obtained by substituting the j-th column of $\boldsymbol{W}$ with $\boldsymbol{b}$. The unknown

parameters and the particular solution sought for can be readily obtained:

$$u_j(t) = \int \frac{W_j}{W}\,\mathrm{d}t \tag{5.20}$$

$$y(t) = \sum_{j=1}^{n} y_j(t) \int \frac{W_j}{W}\,\mathrm{d}t \tag{5.21}$$

5.1 Class Application

$$x'' + 2\,\xi\,\omega_0\,x' + \omega_0^2\,x = F\,\cos(\omega\,t)$$

Solution of the associated homogeneous problem, with the characteristic polynomial:

$$\lambda^2 + 2\,\xi\,\omega_0\,\lambda + \omega_0^2 = 0$$

$$\lambda_{1,2} = -\xi\,\omega_0 \pm \omega_0\,\sqrt{\xi^2 - 1}$$

Consider $\xi < 1$ (low damping).

$$\begin{aligned} x_1(t) &= \mathrm{e}^{-\xi\,\omega_0\,t}\,\mathrm{e}^{i\,\omega_0\,\sqrt{1-\xi^2}\,t} \\ x_2(t) &= \mathrm{e}^{-\xi\,\omega_0\,t}\,\mathrm{e}^{-i\,\omega_0\,\sqrt{1-\xi^2}\,t} \end{aligned}$$

or, alternatively,

$$\begin{aligned} x_1(t) &= \mathrm{e}^{-\xi\,\omega_0\,t}\,\cos\omega_0\,\sqrt{1-\xi^2}\,t \\ x_2(t) &= \mathrm{e}^{-\xi\,\omega_0\,t}\,\sin\omega_0\,\sqrt{1-\xi^2}\,t \end{aligned}$$

Particular solution of the inhomogeneous equation:

$$x(t) \;=\; u_1(t)\,x_1(t) \;+\; u_2(t)\,x_2(t)$$

$$h(t) \;=\; F\,\cos\left(\omega\,t\right)$$

$$x(t) \;=\; -\int_0^t \frac{h\,x_2}{|W|}\,\mathrm{d}t\,x_1 \;+\; \int_0^t \frac{h\,x_1}{|W|}\,\mathrm{d}t\,x_2$$

Using the Symbolic Toolbox by MATLAB:

$$|W| \;=\; \begin{vmatrix} x_1 & x_2 \\ \\ x_1' & x_2' \end{vmatrix} \;=\; x_1\,x_2' \;-\; x_1'\,x_2 \;=\; \omega_0^2\,\mathrm{e}^{-2\,\omega_0\,\xi\,t}\,\sqrt{1\;-\;\xi^2}$$

The actual expression for $x(t)$ can now be easily derived using the Symbolic Toolbox by MATLAB and can be substituted into the differential equation, verifying that it satisfies the equation.

A simpler approach to figure out a solution of the considered inhomogeneous ODE consists in using the Euler formula to represent $\cos \omega\, t$ as sum of complex exponentials while obtaining by direct substitution two particular (and complex conjugate of each other) solutions in the form

$$x_a(t) \;=\; A\, \mathrm{e}^{j\,\omega\, t}; \quad x_b(t) \;=\; A^*\, \mathrm{e}^{-j\,\omega\, t}$$

Eventually, a particular solution of the original problem is obtained as

$$x(t) \;=\; \frac{A\, \mathrm{e}^{j\,\omega\, t} \;+\; A^*\, \mathrm{e}^{-j\,\omega\, t}}{2}$$

Letting

$$A \equiv R + j\,I$$

yields

$$x(t) = R\cos\omega\,t - I\sin\omega\,t$$

There is no a-priori reason why the two alternative approaches should result in the same particular solution.

Chapter 6

Non-homogeneous linear ODEs: undetermined constants

The method of *undetermined constants* is an alternative approach to derive a particular solution of a non-homogeneous, linear ODE. Its peculiarities are:

- Can be applied only with some particular right-hand sides (in most cases, with right-hand sides that are eigenfunctions of the linear operator),
- a solution basis of the associated homogeneous problem is not required.

The essence of the method is as follows:

- look at the form of $g(t)$,
- based on that, guess a tentative solution involving some undetermined constants,
- plug the tentative solution into the ODE and from that derive the conditions required to compute the constants .

Example 6.0.1

$$y'' - 4\,y' - 12\,y = 3\,e^{5\,t}$$

Notice that $e^{5\,t}$ *is an eigenfunction for the linear differential operator appearing on the left-hand side.*

Thus a reasonable guess for the tentative solution is:

$$y(t) \;=\; A\,e^{5\,t}$$

Plugging into the ODE yields:

$$(-7\,A)\,e^{5\,t} \;=\; 3\,e^{5\,t} \;\Longrightarrow\; A \;=\; -\frac{3}{7}$$

and

$$y(t) \;=\; -\frac{3}{7}\,e^{5\,t}$$

Considering an ODE with the same linear differential operator but having $t\,e^{4\,t}$ *as right-hand side, a reasonable guess for* $y(t)$ *is*

$$y(t) = (A\,t + B)\,e^{4\,t}$$

yielding, upon substitution into the ODE,

$$A = -\frac{1}{12}, \quad B = -\frac{1}{36}$$

Many other *special* right-hand side terms can be dealt with by the method of undetermined constants. Refer to the pertinent literature for further details.

Chapter 7

Method of Manufactured Solutions (MMS)

The Method of Manufactured Solutions (MMS) provides a general procedure for generating an analytical solution for code accuracy verification.

The basic idea of the procedure is to simply manufacture an exact solution, without being concerned about its physical realism. (The *realism* or lack thereof has nothing to do with the mathematics, and Verification is a purely mathematical exercise.) In the original, most straightforward and most universally applicable version of the method, one simply includes in the code a general source term, $Q(x, y, z, t)$ and uses it to generate a non-trivial but known solution structure.

We first pick a continuum solution. Interestingly enough, we can pick a solution virtually independent of the code or of the hosted equations. That is, we can pick a solution, then use it to verify an incompressible Navier-Stokes code, a Darcy flow in porous media code, a heat conduction code, an electrode design code, a materials code, etc.

We want a solution that is non-trivial but analytic, and that exercises all ordered derivatives in the error expansion and all terms, e.g., cross-derivative terms. For example, chose a solution involving

tanh. This solution also defines boundary conditions, to be applied in any (all) forms, i.e., Dirichlet, Neumann, Robin, etc. Then the solution is passed through the governing PDEs to give the production term $Q(x, y, z, t)$ that produces this solution. Since this description sounds circular, we will demonstrate with one concrete example.

To emphasize the generality of the concept, we pick the first example solution before we specify the governing equations. Then we have the opportunity to use this same solution for different problems, i.e., set of governing PDEs and boundary conditions. The chosen solution $U(t, x)$ is the following.

$$U(t, x) \; = \; A + sin(B), B \; = \; x + C\, t \tag{7.1}$$

First, let us apply this 1-D transient solution to the non-linear Burgers equation, often taken as a

model for CFD algorithm development.

$$u_t \;=\; u\,u_x \;+\; \alpha\,u_{xx} \tag{7.2}$$

Incidentally, this specified solution $U(t,x)$ is the exact solution for the constant velocity advection equation (i.e., $\alpha = 0$) with boundary condition of

$$u(t,0) \;=\; A \;+\; sin(C\,t),$$

so for the high Reynolds number problem (small α) it may look *realistic* in some sense, but it is not a solution to our governing Eq. (7.2), and its *realism* or lack thereof is irrelevant to the task of Code Verification. We determine the source term $Q(t,x)$ which, when added to the Burgers equation

for $u(t,x)$, produces the solution $u(t,x) = U(t,x)$. We write the Burgers equation as an operator (non-linear) of u,

$$L(u) \equiv u_t + u\, u_x \alpha\, u_{xx} = 0$$

Then we evaluate the Q that produces U by operating on U with L.

$$Q(t,x) = L(U(t,x)) = U_t + U\, U_x\ \alpha\, U_{xx}$$

By elementary operations on the manufactured solution $U(t,x)$ stated in Eq. (7.1),

$$Q(t,x) = C\, \cos(B) + [A + \sin(B)]\, \cos(B) + \alpha\, \sin(B)$$

If we now solve the modified equation

$$L(u) \equiv u_t + u\,u_x\ \alpha\,u_{xx} = Q(t,x)$$

or

$$u_t = u\,u_x + \alpha\,u_{xx} + Q(t,x)$$

with **compatible** initial and boundary conditions, the exact solution will be $U(t,x)$ given by Eq. (7.1).

The initial conditions are obviously just $u(0,x) = U(0,x)$ everywhere. The boundary conditions are determined from the manufactured solution $U(t,x)$ of Eq. (7.1). Note that we have not even specified the domain of the solution as yet. If we want to consider the usual model $0 \leq x \leq 1$ or

something like $-10 \leq x \leq 100$, the same solution Eq. (7.1) applies, but of course the boundary values are determined at the corresponding locations in x. Note also that we have not even specified the type of boundary condition as yet. This aspect of the methodology has often caused confusion. Everyone knows that different boundary conditions on a PDE produce different answers; not everyone recognizes immediately that the same solution $U(t, x)$ can be produced by more than one set of boundary condition types. The following combinations of inflow (left boundary, e.g., $x = 0$) or outflow (e.g., $x = 1$) boundary conditions will produce the same solution $U(t, x)$ over the domain $0 \leq x \leq 1$.

- DirichletDirichlet:

$$u(t, 0) = U(t, 0) = A + \sin(Ct), \quad u(t, 1) = A + \sin(1 + Ct)$$

- DirichletOutflow Gradient (Neumann):

$$u(t,0) \;=\; U(t,0) \;=\; A + \sin\left(Ct\right), \;\; u_x(t,1) \;=\; \cos\left(1 + Ct\right)$$

- Robin (mixed)Outflow Gradient (Neumann) at $x = \pi$:

$$a\,u \;+\; b\,u_x \;=\; c \;\text{ at }\; (t,0) \longrightarrow \text{given } a \text{ and } b, \;\text{ select } c \;=\; \left[A \;+\; \sin\left(C\,t\right)\right] \;+\; b\,\cos\left(C\,t\right)$$

$$u_x(t,\pi) \;=\; \cos\left(\pi \;+\; C\,t\right)$$

For this time-dependent solution, the boundary values are time-dependent. It also will be possible to manufacture time-dependent solutions with steady boundary values, if required by the code.

A very simple application of the MMS

Motion of a point-particle through a stationary fluid:

$$v_t \;=\; -C_D\;|v|\;v; \quad v(0) \;=\; 0;$$

Numerical solution:

$$\begin{aligned} -C_D\;|v(t\;=\;n\,\Delta t)|\;v(t\;=\;n\,\Delta t) \;=\; v_t(t\;=\;n\,\Delta t) \;&\approx\; \frac{v(t\;=\;(n+1)\,\Delta t)\;-\;v(t\;=\;n\,\Delta t)}{\Delta t} \\ &\equiv\; \frac{v^{n+1}\;-\;v^n}{\Delta t} \end{aligned}$$

Thus:

$$v^{n+1} \;=\; v^n \;-\; C_D\,\Delta t\;|v^n|\;v^n$$

$$v^0 \;=\; 0$$

Let $U(t)$ be

$$U(t) \;=\; \mathrm{e}^{-b\,t}$$

for a given constant b. Then:

$$U_t \;=\; -b\,\mathrm{e}^{-b\,t}$$

$$-\,C_D\;|U|\;U \;=\; -\,C_D\,\mathrm{e}^{-2\,b\,t}$$

Thus:

$$Q(t) \;=\; U_t \;+\; C_D\;|U|\;U \;=\; -b\,\mathrm{e}^{-b\,t} \;+\; C_D\,\mathrm{e}^{-2\,b\,t}$$

Thus, we will solve numerically the equation

$$v_t = -C_D \, |v| \, v + Q(t)$$

with the initial condition

$$v(0) = U(0) = 1$$

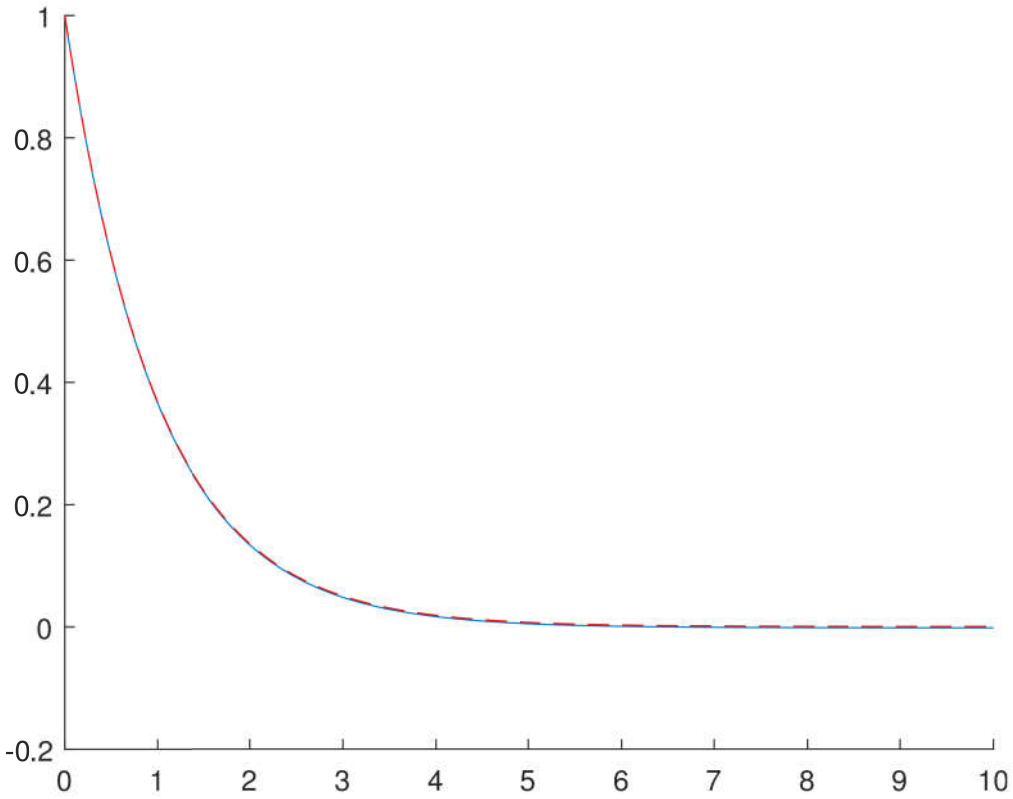
Try out the following MATLAB code. Experiment with the time steps $\Delta t = 0.2, 0.1, 0.05, 0.01$.

```
clear all

close all

clc
```

```
b  = 1;

CD = 1;

T  = 10;

syms t;

U = exp(-b*t);

Q = diff(U,t)+CD*abs(U)*U;


dt = 0.01;

u = nan(ceil(T/dt),1);

tau = zeros(ceil(T/dt),1);
```

```
u(1) = subs(U,t,0);

count = 1;

while 1

    if (tau(count)>=T-dt/1000)

        break;

    elseif tau(count)>(T-dt-dt/1000)

        dt = T-tau(count);

    end

    tau(count+1) = tau(count) + dt;

    count = count + 1;
```

```
    u(count) = u(count-1) - CD*dt*abs(u(count-1))*u(count-1) + dt*subs(Q,t,tau(count)-
dt);
end
tau = tau(1:count);
u   = u(1:count);


figure
hold on
h=plot(tau,u);
h=plot(tau,subs(U,t,tau),'r--');
```


1
0.8
0.6
0.4
0.2
0
-0.2
0
1
2
3
4
5
6
7
8
9
10

Chapter 8

Difference equations

These notes propose a very unformal, incomplete, often not sufficiently detailed introduction to *difference equations*. It is intended to provide students with the (minimal) background required to

deal with the stability of numerical methods for Initial Value Problems (IVP).

Difference equations are recurrence relations, whose solutions are sequences of numbers. Difference equations are the discrete equivalent of IVP's. The general form of a difference equation (DE) is:

$$x_{n+1} = F\left(x_n, x_{n-1}, \ldots, x_{n-k}\right) \tag{8.1}$$

As to get a well-posed (possibly vectorial) Cauchy problem requires an initial condition, so does it to get a well-posed DE problem. Thus, equation (8.1) must be complemented with the specification of some initial values for the sought sequence. So, equation (8.1) requires:

$$x_0 = a_0; \quad x_1 = a_1; \ldots; \quad x_k = a_k \tag{8.2}$$

Definition 8.0.1 (Order of a DE) *The* order *of a DE is the difference between the position in the sequence of the most updated iterate and the position of the oldest iterate appearing the the DE itself. So, in the DE (8.1), the newest iterate is* x_{n+1} *while the oldest is* x_{n-k}*. There are* $(n+1) - (n-k) = k+1$ *steps in between: thus the order of the DE is* $k+1$*.*

Definition 8.0.2 (Fixed Point) *Let's consider a first-order DE*

$$x_{n+1} = F(x_n) \tag{8.3}$$

A fixed point x^* *for (8.3) satisfies the condition:*

$$x^* = F(x^*) \tag{8.4}$$

Thus, the sequence does not moves from a fixed point.

Example 8.0.1

$$x_{n+1} = x_n + 1$$

with

$$x_0 = a$$

Let's try to compute a few iterates:

n	0	1	2	3	4	5	6	7	8	9
x_n	a	$a+1$	$a+2$	$a+3$	$a+4$	$a+5$	$a+6$	$a+7$	$a+8$	$a+9$

It is rather evident that a solution (it can be proved that this is the unique *solution) is*

$$x_n = a + n$$

Theorem 8.0.1 (General solution of linear, homogeneous DE with constant coefficients)

Consider the following linear, homogeneous DE with constant coefficients:

$$\sum_{j=0}^{k} \alpha_j \, x_{n-j} = 0 \tag{8.5}$$

It can be proved that the set of solutions forms a linear vector space of dimension k. A basis of the space can be obtained as follows:

1. *Find the roots* $r_1, \ldots, r_k$ *of the* characteristic polynomial*:*

$$p(r) := \sum_{j=0}^{k} \alpha_j \, r^{k-j}$$

2. *For all roots* r_j *with multiplicity* 1*, consider the basis element*

$$x_n^{(j)} = r_j^n$$

3. *when a root* r_j *has multiplicity* m*, consider the basis elements*

$$x_n^{(j,0)} = r_j^n; \; x_n^{(j,1)} = n \, r_j^n; \; x_n^{(j,2)} = n^2 \, r_j^n; \; \ldots x_n^{(j,m-1)} = n^{m-1} \, r_j^n$$

Example 8.0.2 (Second-order, linear, homogeneous differenceequation with constant coefficients)

$$x_{n+1} = \alpha\, x_n + \beta\, x_{n-1}$$

with

$$x_0 = a; \quad x_1 = b$$

Characteristic polynomial:

$$p(r) = r^2 - \alpha\, r + \beta$$

Roots:

$$s_1; \quad s_2$$

General solution, with different roots:

$$x_n = c_1 s_1^n + c_2 s_2^n$$

General solution, with coincident roots:

$$x_n = c_1 s_1^n + c_2 n s_2^n$$

Get coefficients from the initial conditions:

$$a = c_1 + c_2$$

$$b = c_1 s_1 + c_2 s_2$$

or

$$\begin{aligned} a &= c_1 \\ b &= c_1\, s_1 + c_2\, s_2 \end{aligned}$$

Example 8.0.3 (Fibonacci's sequence)

$$\begin{aligned} F_0 &= 0 \\ F_1 &= 1 \\ F_{n+1} &= F_n + F_{n-1} \end{aligned}$$

The roots of the characteristic polynomial are:

$$s_{\pm} = \frac{1 \pm \sqrt{5}}{2}$$

Enforcing the initial conditions we get the actual solution:

$$F_n = \frac{1}{\sqrt{5}} \left[\left(\frac{1 + \sqrt{5}}{2} \right)^n - \left(\frac{1 - \sqrt{5}}{2} \right)^n \right]$$

Let's compute the following limit:

$$\lim_{n \longrightarrow +\infty} \frac{F_{n+1}}{F_n} = \lim_{n \longrightarrow +\infty} \frac{c_1 \, s_+^{n+1} + c_2 \, s_-^{n+1}}{c_1 \, s_+^n + c_2 \, s_-^n}$$

Notice that $s_+ > 1$, *while* $s_- < 1$. *Therefore,*

$$\lim_{n \longrightarrow +\infty} \frac{F_{n+1}}{F_n} = s_+ = \frac{1 + \sqrt{5}}{2}$$

The number

$$\frac{1 + \sqrt{5}}{2}$$

is known as the Golden Ratio.

Example 8.0.4 (A second-order DE with repeated roots)

$$x_n = 2\,x_{n-1} - x_{n-2}$$

Characteristic polynomial:

$$p(r) = r^2 - 2r + 1$$

Roots:

$$r_{1,2} = 1 \quad \textit{with multiplicity } m = 2$$

General solution:

$$x_n = c_1 1^n + c_2 n 1^n = c_1 + c_2 n$$

Example 8.0.5 (First-order inhomogeneous equation)

$$\begin{aligned} x_n &= 2x_{n-1} + n \\ x_0 &= 1 \end{aligned}$$

The solution of a linear inhomogeneous equation can be written as the sum of a particular solution $x_n^{(p)}$ *of the inhomogeneous equation (disregarding the initial conditions) with the general solution* $x_n^{(g)}$ *of the associated homogeneous equation:*

$$x_n^{(g)} = c\,2^n$$

As for the particular solution, considering that the inhomogeneous term is a first-order polynomial in n*, we shall try a tentative solution of the same kind:*

$$x_n^{(p)} = \alpha\,n + \beta$$

Substituting in the DE:

$$\alpha\, n + \beta = 2\left[\alpha\,(n-1) + \beta\right] + n \iff \alpha\, n + \beta = (2\,\alpha + 1)\, n + (-2\,\alpha + 2\,\beta)$$

Since the identity must hold $\forall\, n$*, we end up with:*

$$\alpha = 2\,\alpha + 1 \implies \alpha = -1$$

and

$$\beta = -2\,\alpha + 2\,\beta \implies \beta = -2$$

A particular solution is therefore:

$$x_n^{(p)} = -n - 2$$

and the solution of the inhomogeneous DE becomes:

$$x_n = c\,2^n - n - 2$$

Substituting the initial condition yields the undetermined constant:

$$1 = c\,2^0 - 0 - 2 \Longrightarrow c = 3$$

and the solution of the problem is

$$x_n = 3\,2^n - n - 2$$

As an exercise, enforce the initial conditions $x_0 = 0$, $x_1 = 1$ *and verify that the unique solution of the considered D.E. is the sequence* $x_n = n$.

Example 8.0.6 (Second-order inhomogeneous equationwith two repeated *characteristic* roots)

$$\begin{aligned} x_n - 4\,x_{n-1} + 4\,x_{n-2} &= n + 1 \\ x_0 &= 1 \\ x_1 &= 5 \end{aligned}$$

$r = 2$ *is the unique root of the characteristic polynomial, with multiplicity* $m = 2$. *The general*

solution of the associated homogeneous DE is thus:

$$x_n^{(g)} = c_1\, 2^n + c_2\, n\, 2^n$$

The particular solution is sought in the form of a linear polynomial in n. Direct substitution yields:

$$x_n^{(p)} = n + 5$$

Henceforth the general solutioon of the inhomogeneous DE:

$$x_n = c_1\, 2^n + c_2\, n\, 2^n + n + 5$$

Substituting the initial conditions and rearranging yields:

$$x_n = (-4)\,2^n + 7\,n\,2^{n-1} + n + 5$$

Definition 8.0.3 (Stability of a first-order difference equation) *A recurrence relation*

$$x_{n+1} = F(x_n)$$

is said to be:

1. Linearly stable *at a fixed point* x^* *if* $|F'(x^*)| < 1$.

2. Linearly unstable *at a fixed point* x^* *if* $|F'(x^*)| > 1$.

3. Linearly neutral *at a fixed point* x^* *if* $|F'(x^*)| = 1$.

The motivation behind the aforementioned definition of linear stability stems from the linearization of $F(x)$ about x^*:

$$x_{n+1} \approx F(x^*) + F'(x^*)\,(x_n - x^*) \tag{8.6}$$

Since x^* is a fixed point, equation (8.6) can be recast as:

$$\delta_{n+1} = F'(x^*)\,\delta_n$$

where

$$\delta_n := x_{n+1} - x_n$$

Thus, the linear stability of a first-order recurrence relation is about the propagation of a small, initial disturbance on the initial condition.

The example 8.0.7 shows what happens to δ_n when $|F'(x^*)| < 1$ (stable case) or $|F'(x^*)| > 1$ (unstable case).

Example 8.0.7 (Stability of a first-order linear D.E.)

$$\begin{aligned} x_{n+1} &= \lambda\, x_n \\ x_0 &= \varepsilon \end{aligned} \tag{8.7}$$

Solution:

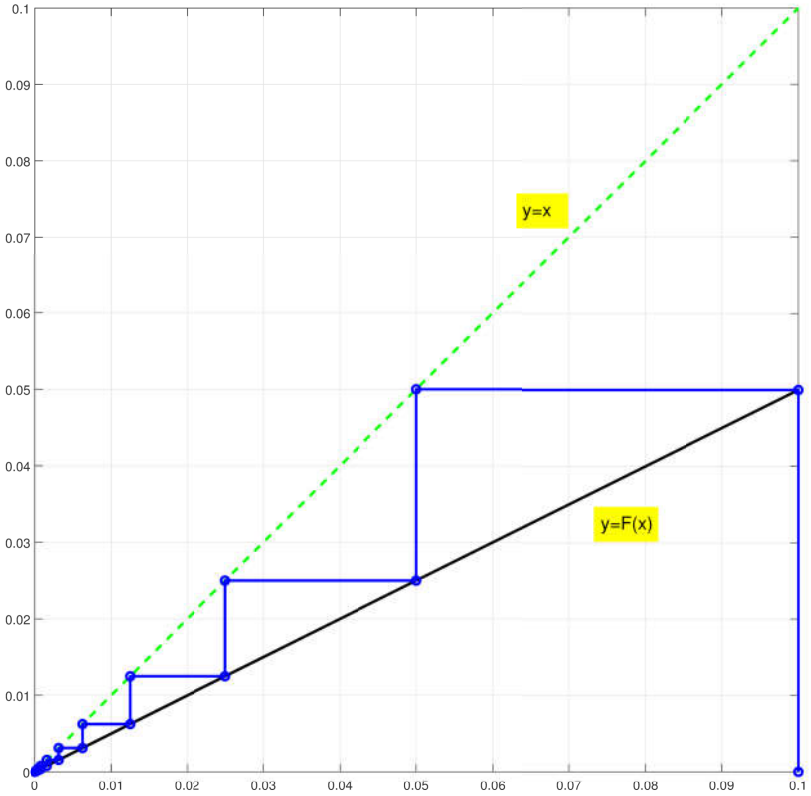
$$x_n = \varepsilon\, \lambda^n$$

Notice that if $|\lambda| < 1$ *the sequence tends towards the fixed point* $x^* = 0$ *for any value of* ε*, while for* $|\lambda| > 1$ *the solution diverges, so that it moves progressively farther from* x^**. Eventually, if* $|\lambda| = 1$*, the iterates do not move from the initial solution* $x_0 = \varepsilon$*.*

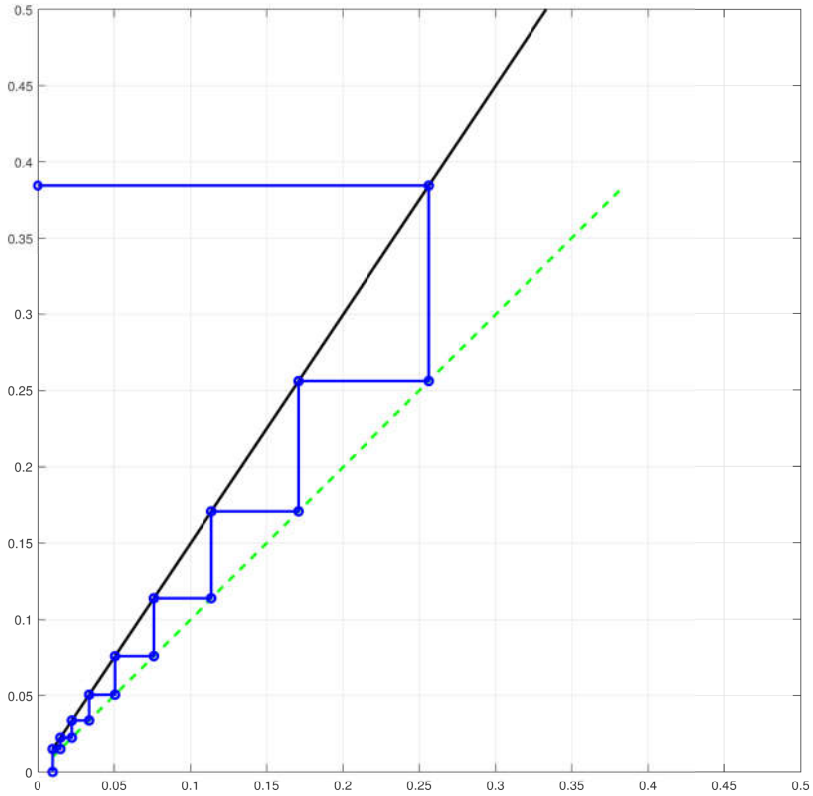
The solution process of a first-order, homogeneous recursion relation $x_{n+1} = F(x_n)$ *can be represented graphically on the phase-plane* $x - F(x)$*: the idea is to follow the path*

$$(x_n, 0) \rightarrow (x_n, F(x_n)) \rightarrow (F(x_n), F(x_n))$$

Figure 8.1a shows the solution with $\lambda = 0.5$ *and* $\varepsilon = 0.1$*. Figure 8.1b shows the solution with* $\lambda = 1.5$ *and* $\varepsilon = 0.01$*.*



(a) $\lambda = 0.5$, $\varepsilon = 0.1$.



(b) $\lambda = 1.5$, $\varepsilon = 0.01$.

Figure 8.1: Solution of the recursion relation (8.7).

8.1 Logistic map and chaos

Consider modelling a population of rabbits. Let's assume that the yearly growth rate is given by a positive, real constant r: in the absence of competing factors, such as predators, the population of rabbits would increase by a factor r every year. So, if $r = 2$, the population doubles every year.

Nevertheless, there are always competing factors as, e.g., limited alimentary resources.

Denoting by x_n the population of rabbits at year n, scaled by a maximum admissible population so that $0 \leq x_n \leq 1$, a reasonable recurrence relation representing the change of x_n on a yearly basis, in absence of competing factors, is

$$x_{n+1} = r\, x_n$$

while if competing factors are present, we may conjecture that thy tend to reduce the population's

growth rate more and more as the actual population gets closer to the maximum admissible population ($x_{max} = 1$):

$$x_{n+1} = \underbrace{r\, x_n}_{\text{Unconstrained growth}} \underbrace{(1 - x_n)}_{\text{Environ. constraint}} \tag{8.8}$$

Equation (8.8) is referred to as the *logistic map*. It is used as one of the simplest models to study chaotic phenomena, where *chaotic means (very, very roughly)* strongly dependent on the initial conditions. *Indeed, John von Neumann had suggested using the logistic map with $r = 4$ as a random number generator in the late 1940s.*

The logistic map is non-linear and the $F(x_n)$ function is a convex parabola.

Notice that the logistic recurrence relation has a well-known differential counterpart, the logistic differential equation*:*

$$\frac{dx}{dt} = r\, x\, (1 - x)$$

A Bifurcation Diagram *is often used to get an overview of the behavior of a non-linear, first-order D.E., as the Logistic Map. A Bifurcationn Diagram of the logistic map is obtained by plotting as a function of* r *a series of values for* x_n *obtained by starting with a random value* x_0*, iterating many times, and discarding the first points corresponding to values before the iterates converge to the attractor. In other words, the set of fixed points of* x_n *corresponding to a given value of* r *are plotted for values of* r *increasing to the right.*

Let's experiment with the logistic map.

Experiments on the logistic map using MATLAB

- *TEST 1:* $r = 0.5$ *(<1)*
- *TEST 2: BIFURCATION DIAGRAM*

- *TEST 3: SKETCH THE F(x) FUNCTION*

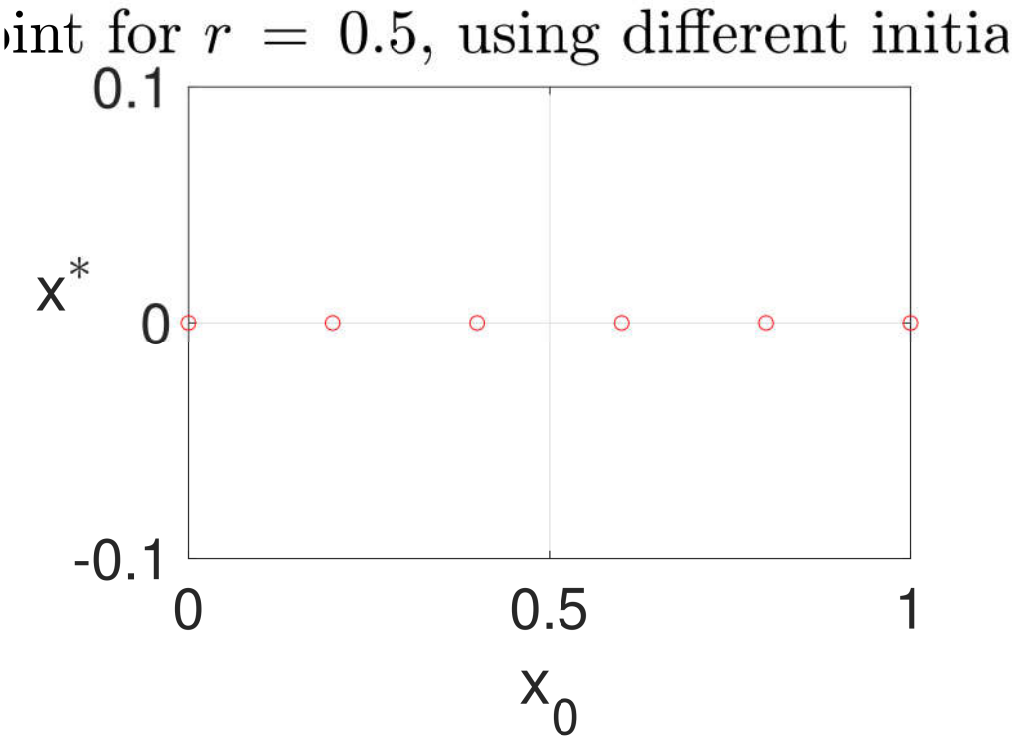
```
function testLogisticMap

close all;

clc;
```

TEST 1: r = 0.5 (<1)

```
disp('TEST 1: r = 0.5, SEVERAL VALUES OF x0 in [0,1]');


x0 = 0:0.2:1.0;

r  = 0.5;

N  = 20;
```

```
xlim = nan(length(x0),1);


for j=1:length(x0)

    X=logisticMap(r,x0(j),N,@logisticF);

    xlim(j) = X(end);

end


figure

h=plot(x0,xlim,'ro');

grid on
```

```
box on

axis([0 1 -0.1 0.1]);

set(gca,'fontname','arial','fontsize',24);

xl=xlabel('x_0');

yl=ylabel('x^\ast');

set(yl,'rotation',0);

ht = title('Fixed point for $r\,=\,0.5$, using different initial conditions');

set(ht,'interpreter','latex');



TEST 1: r = 0.5, SEVERAL VALUES OF x0 in [0,1]
```



TEST 2: BIFURCATION DIAGRAM

%{

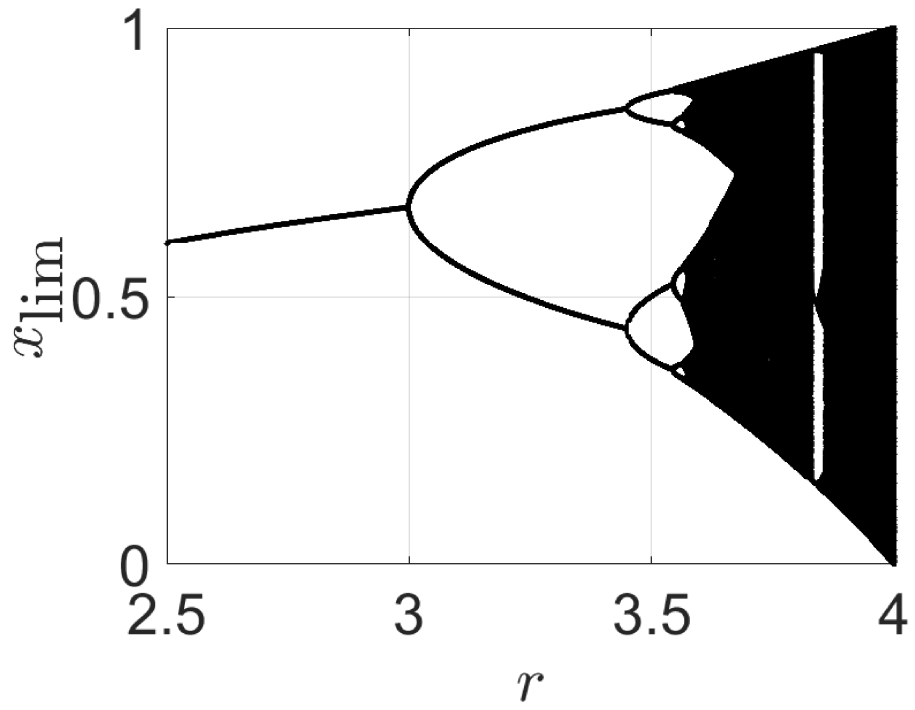
```
      Now, we simulate this system for 10000 values of r
 linearly spaced between 2.5 and 4, and vectorize the simulation
 by considering a vector of independent systems
(one dynamical system per parameter value).
We use 1000 iterations of the logistic map and keep the last 100
 iterations to display the bifurcation diagram.
The initial condition is always x0 = 0.00001.
%}


clc;
disp(['TEST 2: DRAW THE BIFURCATION DIAGRAM FOR r IN [2.5,4]'])
```

```
x0 = 0.00001;

r  = linspace(2.5,4,10000);

N  = 1000;

last = 100;


xlim = nan(last,N);


hf=figure;

hold on;set(hf,'Visible','off');

grid on;
```

```
box on;

axis([min(r) max(r) 0 1]);

set(gca,'fontname','arial','fontsize',24);

xl=xlabel('$r$','interpreter','latex');

yl=ylabel('$x_{\textrm{lim}}$','interpreter','latex');

set(yl,'rotation',90);


for j=1:length(r)

    X=logisticMap(r(j),x0,N,@logisticF);

    xlim(:,j) = X(N-last+1:N);

    h=plot(r(j)*ones(last,1),xlim(:,j),'k.');
```

```
end

set(hf,'Visible','on');
```

```
TEST 2: DRAW THE BIFURCATION DIAGRAM FOR r IN [2.5,4]
```

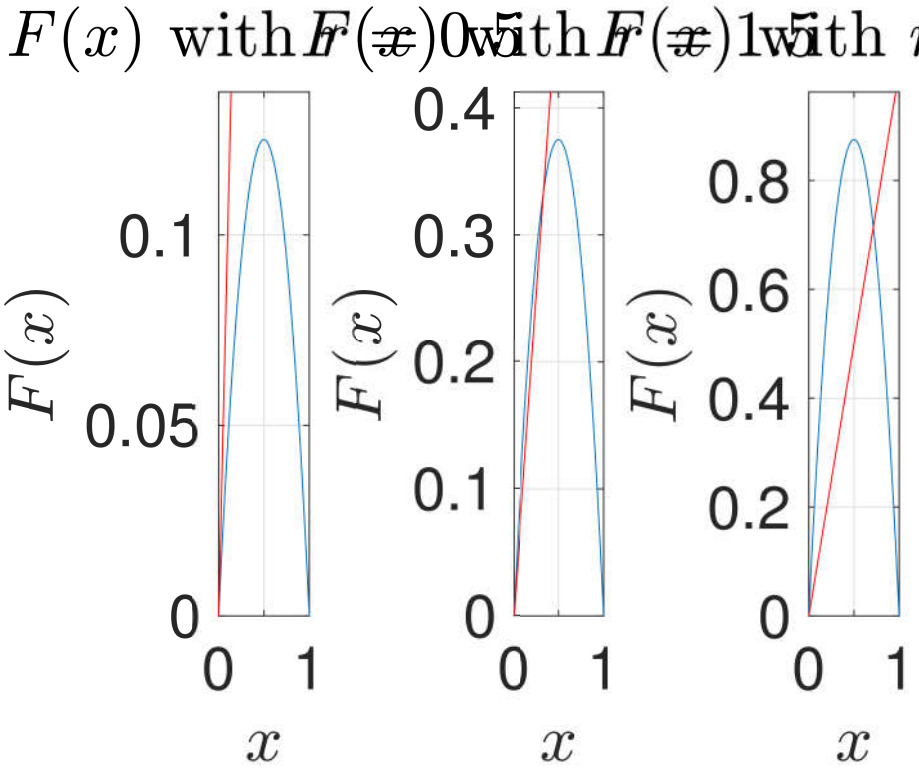


TEST 3: SKETCH THE F(x) FUNCTION

```
disp(['TEST 3: SKETCH THE F(x) FUNCTION'])


figure; hold on;

r = [0.5,1.5,3.5];

for j=1:length(r)

    subplot(1,length(r),j);

    hold on

    h=plot(0:0.01:1,logisticF(0:0.01:1,r(j)));

    h=plot(0:0.01:1,0:0.01:1,'r-');

    grid on;
```

```
    box on;

    axis([0 1 0 1.1*r(j)/4]);

    set(gca,'fontname','arial','fontsize',24);

    xl=xlabel('$x$','interpreter','latex');

    yl=ylabel('$F(x)$','interpreter','latex');

    set(yl,'rotation',90);

    ht=title(['$F(x)$ with $r=',num2str(r(j)),'$']);

    set(ht,'interpreter','latex');

end
```

TEST 3: SKETCH THE F(x) FUNCTION


$F(x)$ with $r = 0.5$
$F(x)$ with $r = 1$
$F(x)$ with $r = 1.5$
0.1
0.05
0
0.4
0.3
0.2
0.1
0
0.8
0.6
0.4
0.2
0
0 1
0 1
0 1
x
x
x

```
end
```

```
function y=logisticF(x,r)

y = r*x.*(1-x);

end


function x=logisticMap(r,x0,N,F)


x(N+1) = nan;

x(1)   = x0;


for j=1:N

    x(j+1) = F(x(j),r);
```

```
end


end
```

The case $r < 1$

It is evident that $x_{n+1} < x_n$. *Therefore, the population of rabbits should decay progressively and eventually disappear. Notice that the logistic map has either one or two equilibria.* **The student should try to show why this is the case using the Web diagram.** *The value* $x^* = 0$ *is invariably a linearly-stable fixed point for the logistic map whenever* $r < 1$. *Indeed,* $F'(0) = r$.

Let's check that indeed the iterates converge to 0 *for large values of* n, *irrespective of the initial value* x_0. *We can use MATLAB, to this end.*

The case $r > 1$

Are there limitations on r *such that* x_n *lies in* $[0, 1]$*? Notice that the maximum value of* $F(x) = r\,x\,(1 - x)$ *is attained at* $x = 0.5$ *and its value is* $r/4$*. Thus, physically reasonable solutions are found only for* $r \leq 4$*.*

Furthermore, let's try to calculate the iterates occurring in the stable 2-cycles (periodic solutions with a period of 2*):*

$$F^2(x) = x \tag{8.9}$$

For sure the fixed points are roots of (8.9). Thus, it can be recognized (using polynomial deflation) that the true 2-cycle iterates are the roots of

$$x^2 - (1 + r^{-1})\,x + r^{-1}\,(1 + r^{-1})$$

and result in

$$x_{\pm}^{(2)} = \frac{1}{2}\left[(1+r^{-1}) \pm r^{-1}\sqrt{(r-3)(r+1)}\right] \tag{8.10}$$

Notice that the discriminant in (8.10) is non-negative only for $r \geq 3$: therefore, real 2-cycles may exist only for $r \geq 3$.

8.2 Homeworks and Class Exercises

Investigate the reported non-linear map

Discuss the following non-linear map. Do the best you can....

$$x_{n+1} = r\, x_n \left(1 - x_n^2\right) \tag{8.11}$$

Chapter 9

Numerical methods for initial-value problems

9.1 Motivations

Unsteady problems in Fluid- or Solid-Mechanics involve the time coordinate. The major difference

location may (in elliptic problems) influence a flow anywhere else, forcing at a given instant will affect the flow only in the future - there is no backward influence. Unsteady problems are, therefore, parabolic-like in time. This means that no conditions can be imposed on the solution - except at the boundaries - at any time after the initiation of the calculation, which has a strong influence on the choice of the solution strategy. To be faithful to the nature of time, essentially all solution methods advance in time in a step-by-step or marching manner. These methods are very similar to ones applied to initial value problems for ordinary differential equations.

9.2 Methods for first-order IVPs

Notice that higher-order IVPs can be easily converted into systems of first-order problems. We will focus on the following model problem:

$$y' = f(t, y) \tag{9.1}$$

$$y(0) = y_0 \tag{9.2}$$

9.2.1 A gentle introduction: two-level, one-stage methods

*Two-level methods involve the values of the unknown at only two time steps (*levels*) and jump from the previous to the next time-step in a single instance (*one-stage*). They can be derived by following*

different alternative approaches.

Derivation of a time-stepping scheme by Taylor expansion

Assume y denotes the exact solution of the IVP. Taylor's theorem with the Lagrange formulation of the reminder states:

$$y(t+h) = y(t) + h\,y'(t) + \frac{h^2}{2}\,y''(t^*)$$

with $t < t^* < t+h$. So fare, we have been using just Maths. Let's include the Physics in our formulation, where Physics is inherent to the ODE under investigation:

$$y'(t) = f(t,y) \Longrightarrow y(t+h) = y(t) + h\,f(t,y) + \frac{h^2}{2}\,y''(t^*)$$

Neglecting higher-order terms yields the Explicit Euler Method*:*

$$\widehat{y} = y + h\, f(t, y)$$

where $\widehat{y}$ *denotes the approximate solution at time* $t + h$ *obtained by the Euler Method.*

Derivation of a time-stepping scheme by integration

Integrating (9.1) through a time step while applying the Fundamental Theorem of Integral Calculus yields:

$$y(t+h) - y(t) = \int_t^{t+h} f(\tau,\, y(\tau))\, d\tau \tag{9.3}$$

In general, the integral must be approximated numerically. Different two-level methods result from the application of different quadrature rules. All those methods admit a common, general representation:

$$y^{n+1} - y^n = \Delta t \left[\alpha f(t_n, y^n) + (1-\alpha) f(t_{n+1}, y^{n+1})\right] \tag{9.4}$$

Choosing either alpha $= 0, 0.5$ *or* 1 *yields three widely used methods:*

1. Explicit Euler scheme*: the integral is approximated using only the value of* y *at the left-hand endpoint of the interval:*

$$\alpha = 1 \tag{9.5}$$

Deriving an expression for the LTE is relatively easy, as for the Euler scheme, even following this integral approach. To this end, the integrand function is expanded in Taylor series about

the time instant t*, to get:*

$$\begin{aligned} y(t+h) - y(t) &= f(t,y)\,h + \int_t^{t+h} \left.\frac{df}{dt}\right|_{\tau=t^*} (\tau - t)\, d\tau \\ &= f(t,y)\,h + \left.\frac{df}{dt}\right|_{\tau=t^*} \frac{h^2}{2} \end{aligned}$$

Thus, the LTE for the numerical solution $\widehat{y}(t+h)$ *at time* $t+h$*, obtained by the Explicit Euler Method, is:*

$$\begin{aligned} \widehat{y}(t+h) - y(t+h) &= [y(t) + f(t,y)\,h] - \left[y(t) + f(t,y)\,h + \left.\frac{df}{dt}\right|_{\tau=t^*} \frac{h^2}{2}\right] \\ &= -\left.\frac{df}{dt}\right|_{\tau=t^*} \frac{h^2}{2} \end{aligned}$$

2. Implicit Euler scheme*: the integral is approximated using only the value of* y *at the right-hand*

endpoint of the interval:

$$\alpha = 0 \tag{9.6}$$

It can be verified that the LTE scales as $O(h^2)$ also for this method:

3. Trapezoidal rule or Crank-Nicolson scheme*: the integral is approximated using the values at both endpoints of the interval, assuming a linear dependence on time of y:*

$$\alpha = \frac{1}{2} \tag{9.7}$$

The LTE for the Crank-Nicolson scheme scales as $O(h^3)$.

Derivation of a time-stepping scheme by finite-differences

$$y'(t) \approx \frac{y(t+h) - y(t)}{h}$$

Thus,

$$\frac{y(t+h) - y(t)}{h} \approx f(t, y(y(t))$$

that is exactly the Explicit Euler Method.

A similar approach yields the Crank-Nicolson scheme:

1. *Forward-finite-difference approximation of y' at t:*

$$y'(t) = f(t, y(t)) \approx \frac{y(t+h/2) - y(t)}{h/2} \tag{9.8}$$

2. *Backward-finite-difference approximation of* y' *at* $t+h$*:*

$$y'(t+h) \;=\; f(t+h, y(t+h)) \;\approx\; \frac{y(t+h) - y(t+h/2)}{h/2} \tag{9.9}$$

3. *Sum (9.9) with (9.8):*

$$f(t+h, y(t+h)) \;+\; f(t, y(t)) \;\approx\; \frac{y(t+h) - y(t+h/2)}{h/2} \;+\; \frac{y(t+h/2) - y(t)}{h/2} \;=\; \frac{y(t+h) - y(t)}{h/2}$$

resulting in

$$y(t+h) \;=\; y(t) \;+\; \frac{h}{2}\,[f(t+h, y(t+h)) \;+\; f(t, y(t))]$$

Local truncation error for the Explicit Euler Method

Let's consider a first-order IVP:

$$y' = f(t, y) \tag{9.10}$$

$$y(t_0) = y_0 \tag{9.11}$$

Let's denote by $\phi(t_n)$ *the exact solution of the first-order IVP at time* $t = t_n \equiv n\,h$*, while* y_n *represents the corresponding numerical solution, obtained with* n *steps of the Explicit Euler Method, without round-off errors.*

The Local Truncation Error (LTE) of the method is defined as

$$LTE \equiv |\phi(t + n\,h) - y_n| \tag{9.12}$$

assuming that $\phi(t + (n-1)\,h) = y_{n-1}$. *In other words, the LTE is the error accumulated on a single time-step, moving from an exact solution at the previous time. The LTE can be readily derived*

by Taylor expansion, assuming $\phi(t)$ is (at least) twice differentiable:

$$\begin{aligned} LTE &\equiv |\phi(t+n\,h) - y_n| \\ &= \left|\left[\phi(t+(n-1)\,h) + h\,f(t+(n-1)\,h, \phi(t+(n-1)\,h)) \right.\right. \\ &+ \left.\left. \frac{h^2}{2}\,\phi''(t^*)\right] \right. \\ &- \left. \left[\phi(t+(n-1)\,h) + h\,f(t+(n-1)\,h,\, \phi(t+(n-1)\,h))\right]\right| \\ &= \frac{h^2}{2}\left|\phi''(t^*)\right| \end{aligned}$$

Thus, if the analytical solution $\phi(t)$ has a bounded second-order derivative,

$$|\phi''| \leq M$$

with $0 \leq M$*, we end-up with:*

$$LTE = O(h^2)$$

Global truncation error for the Explicit Euler Method

The global truncation error (GTE) at time $t_n \equiv t_0 + n\,h$ *results from the accumulation of the LTE at each time-step. The GTE at time* t_n*, denoted by* ϵ_n*, is defined as:*

$$\epsilon_n \equiv \phi(t_n) - y_n \tag{9.13}$$

where, in this, case, the numerical solution matches the analytical solution only at the initial time t_0*:*

$$y_0 = \phi(t_0) \tag{9.14}$$

A bound on ϵ_n can be derived in two stages: first, an upper bound for ϵ_{n+1} will be derived in terms of ϵ_n. In the second stage, the upper bound found in the first stage will be repeatedly applied starting from $n = 0$.

First stage:

$$\begin{aligned}
\epsilon_{n+1} &= \phi(t_{n+1}) - y_{n+1} \\
&= \phi_{n+1} - \phi_n + \phi_n - [y_n + h\, f\,(t_n, y_n)] \\
&= \phi_{n+1} - \phi_n + \underbrace{(\phi_n - y_n)}_{\epsilon_n} - h\, f\,(t_n, y_n) \\
&= \underbrace{\{\phi_{n+1} - [\phi_n + h\, f\,(t_n, \phi_n)]\}}_{LTE} + \epsilon_n \\
&+ h\,[f\,(t_n, \phi_n) - f\,(t_n, y_n)]
\end{aligned} \tag{9.15}$$

Assuming that ϕ'' is bounded for all t of interest,

$$|\phi''| \leq M, \quad 0 \leq M$$

we may bound the LTE as derived in section 9.2.1:

$$LTE \leq M \frac{h^2}{2} \tag{9.16}$$

The third term in (9.15) can be recast in a different form, invoking the fundamental theorem of

integral calculus and the mean-value theorem:

$$\begin{aligned} h\left[f\left(t_n, \phi_n\right) - f\left(t_n, y_n\right)\right] &= h \int_{y_n}^{\phi_n} \frac{\partial f}{\partial y}\left(t_n, z\right)\, dz \\ &= h \frac{\partial f}{\partial y}\left(t_n, y^*\right)\left(\phi_n - y_n\right) \\ &= h \frac{\partial f}{\partial y}\left(t_n, y^*\right)\epsilon_n \end{aligned} \tag{9.17}$$

for some y^* between y_n and ϕ_n. Assuming f is Lipschitz-continuous in y with Lipschitz constant L, the following error bound on the third term in (9.15) can be derived:

$$h\left|f\left(t_n, \phi_n\right) - f\left(t_n, y_n\right)\right| \leq h\, L\, \left|\epsilon_n\right| \tag{9.18}$$

We may now turn back to equation (9.15) and derive the following upper bound for ϵ_{n+1}*:*

$$|\epsilon_{n+1}| \leq |\epsilon_n| + M\,\frac{h^2}{2} + h\,L\,|\epsilon_n| = (1 + h\,L)\,\epsilon_n + M\,\frac{h^2}{2} \tag{9.19}$$

The green term in (9.19) is due to the error accumulation from the previous time-steps, while the term in red is to ascribed to the local truncation error arising when stepping *from time* t_n *to time* t_{n+1}*, which contributes to the GTE even in the circumstance that* $\epsilon_n = 0$*.*

Second stage: *Let's repeatedly apply the inequality (9.19) for* $n = 0, 1, 2, \ldots$*:*

1.

$$|\epsilon_0| = 0$$

2.

$$|\epsilon_1| \leq (1 + h\,L)\,\epsilon_0 + M\,\frac{h^2}{2} = M\,\frac{h^2}{2}$$

3.

$$|\epsilon_2| \leq (1 + h\,L)\,\epsilon_1 + M\,\frac{h^2}{2} = (1 + h\,L)\,M\,\frac{h^2}{2} + M\,\frac{h^2}{2}$$

4.

$$|\epsilon_3| = (1 + h\,L)\,\epsilon_2 + M\,\frac{h^2}{2} = (1 + h\,L)^2\,M\,\frac{h^2}{2} + (1 + h\,L)\,M\,\frac{h^2}{2} + M\,\frac{h^2}{2}$$

5. ...

6.

$$|\epsilon_n| \leq M\,\frac{h^2}{2}\,\sum_{k=0}^{n-1}(1 + h\,L)^k$$

Let's recall the following result, holding for any real value $r \neq 1$*:*

$$\sum_{k=0}^{n-1} r^k = \frac{1-r^n}{1-r}$$

Thus:

$$|\epsilon_n| \leq M \frac{h^2}{2} \frac{1-(1+hL)^n}{1-(1+hL)} = \frac{Mh}{2L}\left[(1+hL)^n - 1\right] \tag{9.20}$$

Notice that:

$$n = \frac{t_n - t_0}{h}$$

We aim to figure out what happens to the GTE under two conditions:

- *Keeping* h *fixed while increasing the final time* t_n*, or, alternatively...*

- *keeping the final time fixed while reducing the time-step h.*

In order to make it easier to understand the behavior of the GTE under the two aforementioned conditions, we modify equation (9.20) as follows:

- *Define* $x \equiv L\,h \geq 0$

- *Notice:* $1 + x \leq 1 + x + x^2/2 + \ldots = e^x$

- *Therefore:*

$$(1 + h\,L)^n \leq e^{n\,h\,L}$$

- *Notice:*

$$n\,h\,L = (t_n - t_0)\,L$$

- *Eventually:*

$$|\epsilon_n| \leq \frac{M\,h}{2L}\left[e^{(t_n - t_0)\,L} - 1\right]$$

From the last expression we recognize that keeping h fixed while increasing t_n results in an exponential growth of the GTE. Conversely, keeping t_n fixed while reducing h yields a linear reduction of the GTE, as expected.

Local truncation error for the Implicit Euler Method

$$LTE := \phi(t+h) - y_{n+1}$$

$$y_{n+1} = \phi(t) + h\,f(t+h, y_{n+1})$$

Let's expand ϕ backwards about time $t+h$:

$$\phi(t) \;=\; \phi(t+h) \;-\; h\;\dot{\phi}\,(t+h) \;+\; \frac{1}{2}\,h^2\;\ddot{\phi}\,(\xi)$$

Substituting into the expression for LTE:

$$LTE \;=\; \phi(t) \;+\; h\;\dot{\phi}\,(t+h) \;-\; \frac{1}{2}\,h^2\;\ddot{\phi}\,(\xi) \;-\; [\phi(t) \;+\; h\,f(t+h, y_{n+1})]$$

Simplifying yields:

$$LTE \;=\; h\;\dot{\phi}\,(t+h) \;-\; \frac{1}{2}\,h^2\;\ddot{\phi}\,(\xi) \;-\; h\,f(t+h, y_{n+1})$$

Let's expand $f(t+h, y_{n+1})$ *in Taylor series about* $\phi(t+h)$*:*

$$f(t+h, y_{n+1}) \;=\; f(t+h, \phi(t+h)) \;+\; f_y(t+h, \Psi)\;(y_{n+1} \;-\; \phi(t+h))$$

Substituting yields:

$$LTE \;=\; -\frac{1}{2}\,h^2\;\ddot{\phi}\,(\xi) \;-\; h\,f_y(t+h, \Psi)\;(y_{n+1} \;-\; \phi(t+h))$$

Notice that $(y_{n+1} \;-\; \phi(t+h)) =: LTE$*. Thus:*

$$LTE \;=\; -\frac{1}{2}\,h^2\;\ddot{\phi}\,(\xi) \;-\; h\,f_y(t+h, \Psi)\,LTE$$

Solving for LTE yields:

$$LTE = -\frac{1}{2}\,h^2\,\frac{\ddot{\phi}\,(\xi)}{1\,-\,h\,f_y(t+h,\Psi)}$$

Letting $\beta := f_y(t,\Psi)$ *and Taylor-expanding the fraction about* $h = 0$ *yields:*

$$LTE = -\frac{1}{2}\,h^2\;\ddot{\phi}\,(\xi)\,\left[1\,+\,\beta\,h\,+\,O(h^2)\,+\ldots\right]$$

Notice that the Taylor expansion must account for the dependence of $f(t+h,\Psi)$ *on* h*.....*

Local truncation error for the Crank-Nicolson Method

The LTE for the Crank-Nicolson method can be derived by different approaches. In these note we combine the LTEs obtained for the Explicit-Euler and for the Implicit-Euler schemes.

The basic idea is that the numerical solution y_{n+1} obtained with the Crank-Nicolson scheme corresponds to the arithmetic average of the numerical solutions obtained with the Explicit- and Implicit-Euler schemes. Thus:

$$y_{n+1}^{CN} = \frac{1}{2}\left(y_{n+1}^{EE} + y_{n+1}^{IE}\right)$$

As for the LTE:

$$\begin{aligned} LTE^{CN} &= \phi(t+h) - y_{n+1}^{CN} \\ &= \frac{1}{2}\left(\phi(t+h) - y_{n+1}^{EE}\right) + \frac{1}{2}\left(\phi(t+h) - y_{n+1}^{IE}\right) \\ &= \frac{1}{2}\,LTE^{EE} + \frac{1}{2}\,LTE^{IE} \end{aligned}$$

Using the results derived in the previous sections we end-up with:

$$LTE^{CN} = \frac{h^2}{4}\left[\ddot{\phi}(\xi) - \ddot{\phi}(\xi')\right] - \frac{h^3}{4}\ddot{\phi}(\xi')\left[f_y(t,\Psi) + O(h)\right]$$

It is readily proved by Taylor expansion that

$$\ddot{\phi}(\xi) - \ddot{\phi}(\xi') = O(h)$$

since both ξ and ξ' lie in the interval $(t, t+h)$. Thus, it is proved that the LTE for the Crank-Nicolson scheme is $O(h^3)$.

9.2.2 Some comments about the simple time-stepping schemes derived so far

1. *Only the Explicit Euler scheme uses only the value of y at t_n to approximate the integral on right-hand side: as for this method, the value of y^{n+1} can be read from (9.4) without invoking any further approximation (e.g., linearization of the right-hand side) and/or without iterating. Methods featuring this property are called* explicit. *All other methods are called* implicit. *So, the Implicit Euler scheme and the Crank-Nicolson scheme are implicit.*

2. *All (consistent) methods yield accurate approximations of the solution at the next time step t_{n+1} when the time step Δt is small. Nevertheless, when a physical problem involves a wide range of time scales (*stiff *problems), the behaviour of the time-stepping scheme when Δt be-*

comes large also matters. Indeed, we might (and often, are) interested only in the long-term (possibly, stationary) behavior of the solution, while the short-term (transitory) behavior is just a nuisance. In that case, time-stepping with large Δt during the transitory phase allows to save a lot of computer time.

3. *The behaviour of time-stepping schemes when Δt becomes large raises the issue of* stability.

4. *As a general rule, implicit schemes are more stable than explicit schemes of comparable accuracy. Yet, most implicit schemes are not absolutely stable.*

9.2.3 Class Application

Model the dynamics of a simple pendulum without invoking the small-oscillation assumption. Use all of the one-step methods proposed so far to solve the resulting non-linear second-order ODE.

The conservation of mechanical energy for the considered system states:

$$K = \frac{1}{2} m v^2; \quad U = m g z$$

$$v = l \dot{\vartheta}; \quad z = l (1 - \cos \vartheta)$$

$$\frac{d}{dt} [K + U] = 0 \Longrightarrow \ddot{\vartheta} + \omega_0^2 \sin \vartheta = 0$$

with

$$\omega_0^2 \equiv \frac{g}{l}$$

As initial conditions, assume

$$\vartheta(0) = \vartheta_0; \quad \dot{\vartheta}(0) = 0$$

Considering both small and large oscillations, compare the numerical solution with the analytical solution.

Stability and energy conservation of one-step methods applied to the simple-pendulum problem

$$\begin{cases} \ddot{\vartheta} + \omega_0^2 \sin(\vartheta) = 0 \\ \vartheta(0) = \vartheta_0 \\ \dot{\vartheta}(0) = 0 \end{cases}$$

Using non-dimensional time $\tau \equiv \omega_0 t$ *yields:*

$$\begin{cases} \ddot{\vartheta} + \sin(\vartheta) = 0 \\ \vartheta(0) = \vartheta_0 \\ \dot{\vartheta}(0) = 0 \end{cases}$$

Considering the linearized problem (small oscillations):

$$\begin{cases} \ddot{\vartheta} + \vartheta = 0 \\ \vartheta(0) = \vartheta_0 \\ \dot{\vartheta}(0) = 0 \end{cases}$$

Convert into system of first-order ODEs:

$$\begin{cases} \dot{v}_1 = v_2 \\ \dot{v}_2 + v_1 = 0 \\ v_1(0) = \vartheta_0 \\ v_2(0) = 0 \end{cases}$$

The discrete version of this problem, where either one of the Eulers methods or the Crank-Nicolson method is used, can be cast in matrix form as:

$$\boldsymbol{v}^{n+1} = \boldsymbol{H}\,\boldsymbol{v}^{n} \tag{9.21}$$

where

$$\boldsymbol{v} \equiv \begin{bmatrix} v_1 \\ \\ v_2 \end{bmatrix}$$

$$\boldsymbol{A} \equiv \begin{bmatrix} 0 & 1 \\ \\ -1 & 0 \end{bmatrix}$$

$$\boldsymbol{B} \equiv \boldsymbol{I}_2 - (1-a)\,h\,\boldsymbol{A}$$

$$\boldsymbol{C} \equiv \boldsymbol{I}_2 + a\,h\,\boldsymbol{A}$$

where a is a parameter that determines what kind of one-step scheme is used ($a = 1$ for EE, $a = 0.5$

for CN, $a = 0$ for IE).

$$\boldsymbol{H} \equiv \boldsymbol{B}^{-1}\,\boldsymbol{C}$$

Then, notice that the non-dimensional mechanical energy of the system reads:

$$\widetilde{E} = \frac{E}{m\,g\,l} = \frac{1}{2}\,\boldsymbol{v}^T\,\boldsymbol{v} \tag{9.22}$$

Indeed, consider:

$$\begin{aligned} E &= \frac{1}{2} m v^2 + m g z \\ &= \frac{1}{2} m \left(l \dot{\vartheta} \right)^2 + m g \left[l - l \cos \vartheta \right] \\ &= \frac{1}{2} m \left(l \dot{\vartheta} \right)^2 + m g \left[l - l \left(1 - \frac{1}{2} \vartheta^2 + \ldots \right) \right] \\ &\approx \frac{1}{2} m \left(l \dot{\vartheta} \right)^2 + \frac{1}{2} m l \vartheta^2 \end{aligned}$$

where the last approximation is consistent with the assumption of small-amplitude oscillations. Scaling with $m g l$ *yields expression (9.22).*

Since $\boldsymbol{v}$ *is real, we can substitute the transpose operation* ^{T} *with the conjugate-transpose operation* * *without affecting the result:*

$$\widetilde{E} = \frac{E}{m g l} = \frac{1}{2} \boldsymbol{v}^* \boldsymbol{v}$$

In order to figure out the stability and energy-conservation properties of the method it is convenient to separate the equations for v_1 and v_2. This can be achieved as follows:

1. *Diagonalization of $\boldsymbol{H}$:*

$$\boldsymbol{H} = \boldsymbol{P}\,\boldsymbol{D}\,\boldsymbol{P}^{-1}$$

where $\boldsymbol{D}$ is the diagonal matrix of eigenvalues, which are denoted by λ_1 and λ_2.

2. *It turns out that, for the problem at hand,*

$$\boldsymbol{P} = \begin{bmatrix} -i & i \\ \\ 1 & 1 \end{bmatrix}$$

and

$$\boldsymbol{S} \equiv \boldsymbol{P}\,\boldsymbol{P}^* = \boldsymbol{P}^*\,\boldsymbol{P} = 2\,\boldsymbol{I}_2$$

which in turn yields

$$\boldsymbol{P}^{-1} = \frac{1}{2}\,\boldsymbol{P}^*$$

3. *Recast the system of equations (9.21) as:*

$$\boldsymbol{w}^{n+1} = \boldsymbol{D}\,\boldsymbol{w}^n$$

where

$$\boldsymbol{w} \equiv \boldsymbol{P}^{-1}\,\boldsymbol{v} = \frac{1}{2}\,\boldsymbol{P}^*\,\boldsymbol{v}$$

and

$$\boldsymbol{v} = \boldsymbol{P}\,\boldsymbol{w}$$

Since $\boldsymbol{D}$ *is diagonal, the resulting system of equations is uncoupled and the two equations can be solved independently of each other. Therefore, the stability of the method is governed by the eigenvalues of* $\boldsymbol{D}$*.*

4. *Choosing* $a = 1$*, i.e. EE, yields* $\lambda_1 = 1 + ih$ *while* $\lambda_2 = 1 - ih$*. Thus, the method is inherently unstable, no matter how small we take* h*.*

5. *Choosing* $a = 0$*, i.e. IE, yields* $\lambda_1 = 1\,/\,(h + i)$ *while* $\lambda_2 = 1\,/\,(1 + ih)$*. Thus, the method is inherently stable, no matter how large we take* h*. Yet, the energy of the solution decreases progressively, while it should stay constant.*

6. *Choosing* $a = 0.5$, *i.e. CN, we get*

$$\lambda_1 = \frac{\left|\frac{h}{2} - i\right|}{\left|\frac{h}{2} + i\right|}$$

$$\lambda_2 = \frac{\left|1 - i\frac{h}{2}\right|}{\left|1 + i\frac{h}{2}\right|}$$

Both eigenvalues have unit magnitude. Therefore, the energy of the system is conserved.

7. *Let's work out the expression for the dimensionless mechanical energy of the system:*

$$\widetilde{E} = \frac{1}{2}\,\boldsymbol{v}^*\,\boldsymbol{v} = \frac{1}{2}\,\boldsymbol{w}^*\,\boldsymbol{P}^*\,\boldsymbol{P}\,\boldsymbol{w} = \frac{1}{2}\,\boldsymbol{w}^*\,2\,\boldsymbol{I}_2\,\boldsymbol{w} = \boldsymbol{w}^*\,\boldsymbol{w}$$

$$\begin{aligned}
\widetilde{E}^{n+1} - \widetilde{E}^{n} &= (\boldsymbol{w}^{n+1})^{*}\, \boldsymbol{w}^{n+1} - (\boldsymbol{w}^{n})^{*}\, \boldsymbol{w}^{n} \\
&= (\boldsymbol{w}^{n})^{*}\, \boldsymbol{D}^{*}\, \boldsymbol{D}\, \boldsymbol{w}^{n} - (\boldsymbol{w}^{n})^{*}\, \boldsymbol{w}^{n} \\
&= (\boldsymbol{w}^{n})^{*}\, (\boldsymbol{D}^{*}\, \boldsymbol{D} - \boldsymbol{I}_2)\, \boldsymbol{w}^{n}
\end{aligned}$$

or, equivalently:

$$\frac{\widetilde{E}^{n+1} - \widetilde{E}^{n}}{\widetilde{E}^{n}} = (\boldsymbol{u})^{*}\, (\boldsymbol{D}^{*}\, \boldsymbol{D} - \boldsymbol{I}_2)\, \boldsymbol{u}$$

where $\boldsymbol{u}$ is a vector with unit 2-norm.

8. *For CN, it can be verified that $(\boldsymbol{D}^{*}\, \boldsymbol{D} - \boldsymbol{I}_2) = 0$ and energy is conserved, indeed.*

9.2.4 Stability

There exist several definitions of stability. *We invoke a rough one, though sufficient for our purposes.*

Definition 9.2.1 (Stability) *A numerical method for the integration of IVPs is called* stable *if it produces a bounded solution whenever the solution of the underlying differential equation is also bounded.*

The stability of (9.1) is often studied on an associated linearized autonomous problem:

$$y' = \frac{\partial f}{\partial y}(\bar{t}, \overline{y})\,(y - \overline{y}) \tag{9.23}$$

Investigating the stability of two-level methods applied to (9.23) is relatively straightforward, in general, as we are going to appreciate.

Stability of the Explicit Euler scheme

$$y' = \lambda y; \quad \lambda \in \mathbb{C},\ \mathcal{R}(\lambda) \leq 0$$

$$y^{n+1} - y^n = \lambda \Delta t\, y^n \implies \frac{y^{n+1}}{y^n} = 1 + \lambda \Delta t$$

The amplification factor *is defined as:*

$$r_n := \left| \frac{y^{n+1}}{y^n} \right|$$

As for the explicit Euler scheme:

$$r_n = |1 + \lambda \Delta t|$$

Stability requires $r_n \leq 1\ \forall\, n$. *Thus:*

$$|1 + \lambda \Delta t| \leq 1$$

If λ is purely real, the stability limit is given by

$$|\lambda\,\Delta t| \leq 2$$

Otherwise, $\lambda\,\Delta t$ must lie within a unit circle centred at $(-1, 0)$ in the complex plane.

Since for a given value of λ there is a maximum time-step width Δt that guarantees stability, the method is called conditionally stable.

Stability of the Implicit Euler scheme

$$y' = \lambda\, y; \quad \lambda \in \mathbb{C},\ \mathcal{R}(\lambda) \leq 0$$

$$y^{n+1} - y^n = \lambda\,\Delta t\, y^{n+1} \Longrightarrow \frac{y^{n+1}}{y^n} = \frac{1}{1 - \lambda\,\Delta t}$$

The amplification factor *is:*

$$r_n = \frac{1}{|1 - \lambda \Delta t|}$$

Stability requires:

$$\frac{1}{|1 - \lambda \Delta t|} \leq 1$$

For obtaining a stable solution, $\lambda \Delta t$ *must lie outside a unit circle centred at* $(1, 0)$ *in the complex plane.*

Since for a given value of λ*, with* $\mathcal{R}(\lambda) \leq 0$ *the method is stable irrespective of the time-step width* Δt*, the method is called* unconditionally stable*. Notice that the method yields a bounded solution even when* $\mathcal{R}(\lambda) > 0$*, i.e., when the ODE does not have a bounded solution.*

Stability of the Crank-Nicolson scheme

$$y' = \lambda y; \quad \lambda \in \mathbb{C},\ \mathcal{R}(\lambda) \leq 0$$

$$y^{n+1} - y^n = \frac{\lambda \Delta t}{2}\left[y^{n+1} + y^n\right] \Longrightarrow \frac{y^{n+1}}{y^n} = \frac{2 + \lambda \Delta t}{2 - \lambda \Delta t}$$

The amplification factor *is:*

$$r_n = \left|\frac{2 + \lambda \Delta t}{2 - \lambda \Delta t}\right|$$

Stability requires:

$$\left|\frac{2 + \lambda \Delta t}{2 - \lambda \Delta t}\right| \leq 1$$

For obtaining a stable solution, $\lambda \Delta t$ *must lie within the half-plane with negative real part of the complex plane.*

Since for a given value of λ*, with* $\mathcal{R}(\lambda) \leq 0$ *the method is stable irrespective of the time-step width* Δt*, the method is called* unconditionally stable. *Though the Crank-Nicolson method yields bounded solutions for* $\lambda \in \mathcal{R}^-$*, the numerical solution might manifest a step-to-step oscillation with sign change at every step: this behaviour is not akin the solution of the analytical problem, which decays monotonically towards zero. Indeed, consider the ODE*

$$y' = -a\,y, \quad a \in \mathcal{R}^+$$

Discretization by the Crank-Nicolson method results in the following change of the solution at each time step:

$$\frac{y^{n+1}}{y^n} = \frac{2 - a\,\Delta t}{2 + a\,\Delta t}$$

It is evident that the sign of the numerical solution switches at every time step provided a $\Delta t > 2$. *Conversely, the numerical solution returned from the Implicit Euler scheme tends monotonically to zero irrespective of how large is* Δt. *The Implicit Euler scheme is, under this perspective, incredibly* robust. *This is the main reason why the the Implicit Euler scheme is the default time-stepping scheme of most commercial simulation packages of Computational Fluid Mechanics.*

9.2.5 Linear *Multipoint* (or Linear *multistep*) methods

It can be proved that the maximum (global) accuracy that can be attained with two-level methods is second order. In order to attain higher-order approximations one must use information available at more than two points. The additional points may be ones at which data has already been computed or points between t_n *and* t_{n+1}, *which are used strictly for computational convenience. The former*

are called multipoint *methods, the latter* Runge-Kutta *methods. Here we present some multipoint methods. Multistep methods are explicit in nature, therefore only conditionally stable.*

A multi-step method for the first-order IVP is obtained by integrating the ODE over the interval $[t, t+h]$*:*

$$y_{n+1} - y_n = \int_t^{t+h} f(\tau, y(\tau))\, d\tau$$

Then, an interpolating polynomial $p(\tau)$ *substitutes the integrand function* $f(\tau, y(\tau))$ *and the integration is carried out analytically. The interpolating polynomial through the* nodes $(t_{n-k}, y_{n-k}, \ldots, (t_{n+1}, y_{n+1})$ *depends linearly on* $y_{n-k}, \ldots, y_{n+1}$*. Therefore, a general formulation for a* $k+1$*-point multi-step method is:*

$$y_{n+1} - y_n = h \sum_{j=-k}^{1} \alpha_j\, f(t + j\, h, y_j) \tag{9.24}$$

Different choices of the interpolating polynomial $p(\tau)$ yield different multi-step methods. If the interpolation error is such that

$$f(t, y(t)) - p(t) = \alpha(t)\, h^m + O(h^{m+1})$$

then the LTE for the method is $O(h^{m+1})$. Indeed:

$$\phi(t_n + h) = \phi(t_n) + \int_{t_n}^{t_n+h} f(\tau, \phi(\tau))\, d\tau$$

Using the mean-value theorem:

$$\begin{aligned} y_{n+1} &= \phi(t_n) + \int_{t_n}^{t_n+h} [f(\tau, \phi(\tau)) - \alpha(\tau)h^m + O(h^{m+1})]\, d\tau \\ &= \phi(t_n + h) - \alpha(\xi)\, h^{m+1} + O(h^{m+2}) \end{aligned}$$

Adams-Bashforth methods

The integral at the right-hand side of (9.4) is approximated by integrating anallytically an approximate representation of f obtained by polynomial interpolation through points t_{n-m}, t_{n-m+1}, ..., t_n: this results in an explicit method of order $m + 1$.

- *First-order Adams-Bashforth scheme: this is the Explicit Euler scheme.*

- *Second-order Adams-Bashforth scheme:*

$$y^{n+1} - y^n = \frac{\Delta t}{2}\left[3\,f^n - f^{n-1}\right]$$

- *Third-order Adams-Bashforth scheme:*

$$y^{n+1} - y^n = \frac{\Delta t}{12}\left[23\,f^n - 16\,f^{n-1} + 5\,f^{n-2}\right]$$

- *Fourth-order Adams-Bashforth scheme:*

$$y^{n+1} - y^n = \frac{\Delta t}{24}\left[55\,f^n - 59\,f^{n-1} + 37\,f^{n-2} - 9\,f^{n-2}\right] \tag{9.25}$$

Adams-Moulton methods

Similar to Adams-Bashforth, but the interpolation polynomial includes data at t_{n+1}, thus yielding a class of implicit schemes.

- *First-order Adams-Moulton scheme: this is the Implicit Euler scheme.*

- *Second-order Adams-Moulton scheme: this is the Crank-Nicolson scheme.*

- *Third-order Adams-Moulton scheme:*

$$y_{n+1} - y_n = \frac{\Delta t}{12} \left[5\, f_{n+1} + 8\, f_n - f_{n-1} \right] \tag{9.26}$$

- *Fourth-order Adams-Moulton scheme:*

$$y_{n+1} - y_n = \frac{\Delta t}{24} \left[9\, f_{n+1} + 19\, f_n - 5, f_{n-1} + f_{n-2} \right] \tag{9.27}$$

Drawing the boundary of the region of stability of multistep methods

Let's consider the simple linear ODE:

$$y' = \lambda\, y \quad \lambda \in \mathbb{C} \tag{9.28}$$

Applying a multi-step method to equation (9.28) yields a difference equation, whose general solution at time-step n can be written as a linear combination of the n-th powers of the roots ϑ of the associated

characteristic polynomial $P(\vartheta)$. It is readily recognized that for multi-step methods $P(\vartheta)$ takes the general form

$$P(\vartheta) = \rho(\vartheta) - z\,\sigma(\vartheta) \tag{9.29}$$

where ρ and σ are polynomials and $z \equiv \lambda\,\Delta t$.

The region of stability of the method is the set of complex numbers z such that all roots ϑ satisfy the condition $|\vartheta| \leq 1$. On the boundary of the stability region,

$$|\vartheta| = 1 \Longrightarrow \vartheta = e^{i\varphi}$$

A result on multistep methods states that on the boundary of stability there exists only a single root with $|\vartheta| = 1$ (see section 18.1.1 for more information). Therefore, we look for the boundary ∂S of

the stability region S letting $P(e^{i\varphi}) = 0$ or, using (9.29):

$$\rho(e^{i\varphi}) - z\,\sigma(e^{i\varphi}) = 0 \Longrightarrow z = \frac{\rho(e^{i\varphi})}{\sigma(e^{i\varphi})} \tag{9.30}$$

Thus, ∂S can be drawn by running φ from 0 to $2\,\pi$, using (9.30) to identify the corresponding $\zeta \in \partial S$. Figure 9.1 shows the boundaries of the stability regions for three widely used multistep methods: the 3- and 4-point Adams-Bashforth methods (AB3 and AB4, resp.) are explicit, while the 3-point Adams-Moulton scheme (AM3) is implicit. Notice some relevant facts:

1. *The S region of (implicit) AM3 is considerably wider than those of (explicit) AB3 and AB4.*

2. *AM3 is implicit, yet not A-stable. A result about multistep methods states that multistep methods of order higher than 2 cannot be A-stable (see section 18.1.1 for more information).*

3. *The S region of AB3 is wider than that of AB4. A general drawback of multistep methods is that their domain of stability shrinks when increasing the order of accuracy. This is opposite to what happens with Runge-Kutta methods.*

4. *AM3 and AB3 are unstable for purely imaginary λ, while AB4 is conditionally-stable. Therefore, among the three considered methods, only AB4 can be used to solve problems with purely imaginary λ.*

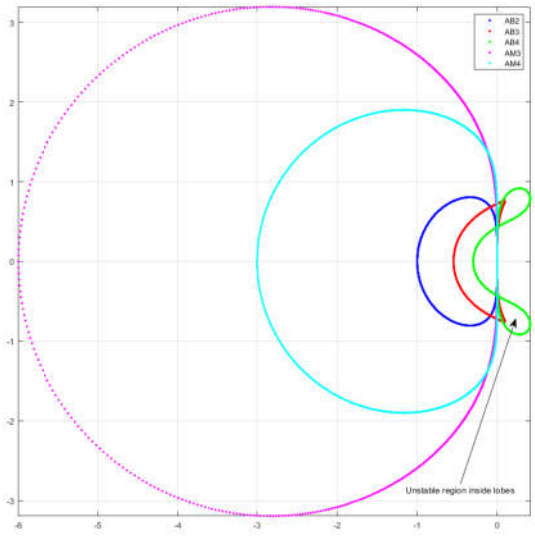


Figure 9.1: Boundary of stability for several common multistep methods. AB2, AB3, AB4, AM3, AM4.

Multistep methods: *spurious* roots

The numerical solution of the Cauchy problem

$$\begin{aligned} y' &= \lambda\, y \\ y(0) &= y_0 \end{aligned} \tag{9.31}$$

can be derived analytically (or nearly so) by the solution method for difference equations, proposed in chapter 8. To this end, the roots of the characteristic polynomial must be computed. For multistep methods,the characteristic polynomial has order ≥ 2 *and, as a consequence, has more than a single root, in general. Most often, one of the roots corresponds to an approximation of the analytical solution, while the remaining roots are* spurious*: their growth must be kept under control to limit the global (i.e., long-time) error of the method. Periodic re-starting of a multistep method is a heuristic strategy commonly adopted to limit the growth of the spurious roots.*

Let's apply the aforementioned considerations to the third-order Adams-Moulton method.

$$y^{n+1} - y^n = \frac{z}{12}\left[5\,y^{n+1} + 8\,y^n - y^{n-1}\right] \tag{9.32}$$

where

$$z \equiv \lambda \Delta t$$

The roots of the associated characteristic polynomial are:

$$\begin{aligned} \sigma_1 &= \frac{b + \sqrt{b^2 - 4\,a\,z}}{2\,a} \\ \sigma_2 &= \frac{b - \sqrt{b^2 - 4\,a\,z}}{2\,a} \end{aligned}$$

with

$$a := 12 - 5\,z; \qquad b = 12 + 8\,z$$

The analytical solution of (9.31) reads:

$$Y^n = y_0 \, (e^z)^n$$

while the analytical solution of (9.32) is

$$y^n = a_1 \, \sigma_1^n + a_2 \, \sigma_2^n$$

where the coefficients a_1 and a_2 should derive from the initial conditions. Nevertheless, as multistep methods are not self-starting, we need an additional, unphysical initial condition to calculate a_1 and a_2. As for the third-order Adams-Moulton method, an additional initial condition is required, which can be obtained by applying a one-step method (e.g., Implicit Euler or Explicit Euler) to advance

the solution through the first time-step, or by enforcing a further unphysical - and totally arbitrary - initial condition as, e.g., $y^1 = y^0$. *The starting strategy determines the weights (the a's coefficients) of the spurious roots.*

As for the considered method, notice that the Taylor expansions of σ_1 *and* σ_2 *are, respectively:*

$$\sigma_1 = \frac{5\,z^5}{144} + \frac{z^4}{12} + \frac{z^3}{6} + \frac{z^2}{2} + z + 1 + h.o.t. \tag{9.33}$$

$$\sigma_2 = -\frac{z\,(515\,z^4 + 1236\,z^3 - 5328\,z^2 + 12096\,z - 20736)}{248832} + h.o.t. \tag{9.34}$$

whereas

$$e^z = \frac{z^4}{4!} + \frac{z^3}{3!} + \frac{z^2}{2!} + z + 1 + h.o.t. \tag{9.35}$$

Thus, σ_1 *is a fourth-order approximation to* e^z, *while* σ_2 *is the spurious root, tending to zero as*

$z \longrightarrow 0$.

Letting

$$\frac{y_1}{y_0} = -\frac{6 + 4\,z + \sqrt{3}\,\alpha}{5\,z - 12}; \quad \alpha := \sqrt{7\,z^2 + 12\,z + 12} \tag{9.36}$$

yields $a_2 = 0$. *Nevertheless, this strategy is far from being general, as it works only for equation (9.31) and does not account for the effect of round-off errors.*

Homework 9.2.1 (Experiencing with multistep methods) *Solve the problem*

$$\begin{aligned} \dot{z} &= \dot{\vartheta}\ z \\ z(0) &= z_0 \end{aligned}$$

using the third-order Adams-Moulton and the third-order Adams-Bashforth method. Try different

strategies to control the growth of the parasitic root:

1. *Set* $y^1 = y^0$.

2. *Set* y^1 *according to (9.36).*

3. *Compute* y^1 *by either the Implicit or the Explicit Euler method or by the Crank-Nicolson method.*

4. *Compute* y^1 *by the Explicit Euler method and restart the time-stepping scheme every* 10 *time-steps.*

Assume $\vartheta(t) = \omega\, t$ *with* $\omega = 2\,\pi$. *Use* $|z| \equiv |\lambda|\ \Delta t = 0.5$ *or* $|z| = 0.25$ *and simulate the behaviour of the system till the final time* $T = 5$ *(5 complete turns of the systems, according to the analytical solution).*

Comments the results. Notice that the stability region of the implicit Adams-Moulton method does not include any point on the imaginary axis, while the stability region of the explicit third-order Adams-Bashforth method includes an interval of the imaginary axis about the origin. Thus, as for the specific considered problem, it is expected that the explicit scheme is conditionally stable while the implicit scheme is unconditionally unstable. The second-order Adams-Bashforth scheme should be unconditionally unstable for the considered problem. Try it!

Other common pitfalls of multistep methods

- *For $k > 1$, multistep methods are not self-starting.*

- *Low-order methods and small time-steps must be used in the early stages of the calculation, while enough data become available to use the desired high-order multistep method.*

- *standard multistep methods are designed under the assumption of constant* Δt*: therefore, the time-step cannot be changed during the calculation without degrading the accuracy of the method. If the time-step has to be changed due to stability issues, than the rate-of-change of* Δt *must be rather slow, say no more than* 10% *per time-step.*

- *Spurious roots might grow unexpectedly (real-life simulations do not deal with the simple equation (9.23), in general). Spurious roots correspond to non-physical solutions. Their growth can be kept under control (usually) by restarting the method every so often.*

- *The AB4 and AM4 methods are commonly combined to result in a predictor-corrector scheme (see section 9.2.6), where AB4 works as predictor stage while AM4 serves as corrector stage. Notice that this predictor-corrector method requires only two functions evaluations per time-step: therefore, the method is less computationally-expensive than the (standard) fourth-order*

Runge-Kutta method.

Exercises

Exercise 9.2.1 (Rotation of a point in the plane) *Solve by AB3, AB4 and AM3 the following ODE, representing the circular motion of a point in the plane, about the origin:*

$$\frac{dz}{dt} = i\omega\, z; \quad z(0) = \rho\, e^{i\vartheta_0} \tag{9.37}$$

where $z \in \mathbb{C}$, $\rho \in \mathbb{R}^+$, , $\omega \in \mathbb{R}^+$. *Compare the numerical solutions (for different values of* $\omega\,\Delta t$*) with the analytical solution. Comment the results considering the stability regions reported in figure 9.1.*

9.2.6 Predictor-corrector methods

- *Motivation: try to combine the best of explicit and implicit methods.*

- *Strategy: Use an explicit scheme to compute an approximate value of* y^* *at some intermediate time between* t_n *and* t_{n+1}*. Then, use* y^* *in an implicit scheme to evaluate a* corrected *solution at* t_{n+1}*.*

A widely used predictor-corrector method combines an Explicit Euler predictor stage with a Crank-Nicolson corrector stage: this method is named Improved Euler method *or* Heun's method*.*

$$\begin{aligned} y^*_{n+1} &= y_n + h\, f\left(t_n,\, y_n\right) \\ y_{n+1} &= y_n + \frac{h}{2}\left[f\left(t_n,\, y_n\right) + f\left(t_n + h,\, y^*_{n+1}\right)\right] \end{aligned}$$

This method can be recast in the general form of Runge-Kutta methods:

$$\begin{aligned} y_{n+1} &= y_n + h \sum_{j=1}^{K} c_j K_j \\ K_j &\equiv f\left(t_n + a_j h, y_n + h \sum_{m=1}^{K} b_{j,m} K_m\right) \end{aligned}$$

As for the Improved Euler method:

$$c_1 = c_2 = \frac{1}{2}$$

$$a_1 = 0; \quad a_2 = 1$$

$$b_{1,1} = b_{1,2} = 0; \quad b_{2,1} = 1; \quad b_{2,2} = 0$$

The coefficients of Runge-Kutta methods can be conveniently arranged in the so-called Butcher table

that, for the Improved Euler method, reads:

0	0	0
1	1	0
	1/2	1/2

The stability of a predictor-corrector method can be investigated by considering the simple linear ODE $y' = \lambda\, y$ *and by recognizing that the application of the method results in a difference equation. For instance, the Improved Euler method results in:*

$$y_{n+1} = y_n + \frac{h}{2}\left[\lambda\, y_n + \lambda\,(y_n + h\,\lambda\, y_n)\right]$$

Defining $z \equiv \lambda\, h$ *we end up with:*

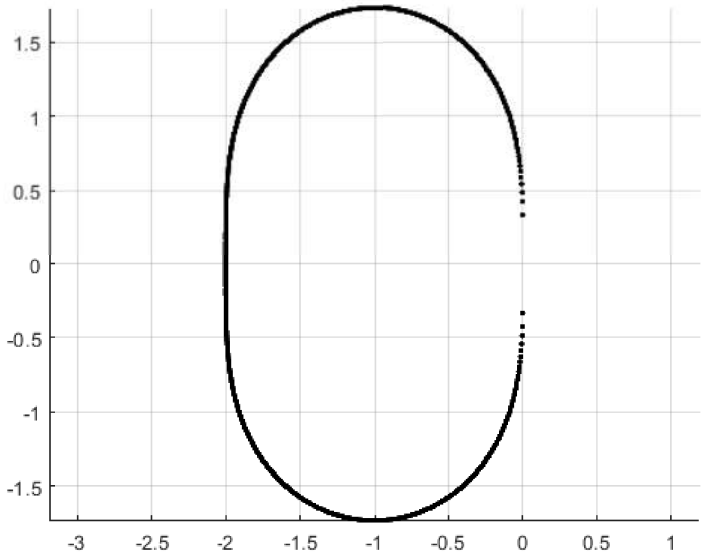
$$y_{n+1} = \left[1 + z + \frac{z^2}{2}\right] y_n \equiv p(z)\, y_n$$

The method is stable if

$$\left|\frac{y_{n+1}}{y_n}\right|, leq, 1 \implies |p(z)| \leq 1$$

A practical approach for drawing the stability boundary is presented in section 9.2.7 and applied to the Heun's method. A MATLAB code for drawing the stability boundary of Heun's method is reported below.

```
clear all
clc
close all

syms rho real positive
syms phi real positive

z = rho*(cos(phi)+1i*sin(phi));
p = 1+z+0.5*z^2;
q = simplify(real(p*conj(p)));

dPhi = 2*pi/1440;
Phi = 0:dPhi:2*pi-dPhi;

hf = figure; hold on; axis equal; grid on;
for j=1:length(Phi)
  Q = subs(q,'phi',Phi(j));
  RHO{j} = solve(Q-1, 'Real', true);
  figure(hf);
  for k=1:length(RHO{j})
    plot(RHO{j}(k)*cos(Phi(j)),RHO{j}(k)*sin(Phi(j)),'k.');
  end
end
```



Published with MATLAB® R2021b

Exercise 9.2.2 (Application of predictor corrector methods) *Derive the predictor-corrector method based on AB4+AM4. Use the stability analysis outlined for multistep schemes and draw the stability boundary for this method. Then, use the method to solve (9.37).*

9.2.7 Runge-Kutta methods

- *Self-starting methods.*
- *Bear changes of* Δt*.*
- *In general, wide stability region whose extent increases with increasing order of accuracy.*
- *Use points between* t_n *and* t_{n+1} *rather than earlier points: this, in turn, requires several computations of the derivative* $f(t, y)$ *at each intermediate time-step, which increase the computational*

work per time-step compared to multi-step methods of the same order.

- *Automatic error assessment is relatively easy: to this end, just add one more step to a Runge-Kutta method and compare the truncation errors of the last two steps (consider, e.g., the Runge-Kutta-Merson method [**?**]).*

- *Formal accuracy p (i.e., GTE) shows* jumps *with the minimum admissible number s of sub-steps:*

p	1	2	3	4	5	6	7	8
$min\ s$	1	2	3	4	6	7	9	11

In these section we propose a very compact introduction to Runge-Kutta methods, without entering into technicalities. A second-order and a fourth-order Runge-Kutta methods are outlined.

Heun's method (second-order)

Heun's method can be conceived as a predictor-corrector scheme, made of an explicit-Euler predictor over $(\Delta t)/2$ *followed by a mid-point rule corrector over a full-length step. The method is globally second-order accurate.*

$$y^{n+1/2} - y^n = \frac{\Delta t}{2} f(t_n, y^n) \tag{9.38}$$

$$y^{n+1} - y^n = \Delta t\, f(t_{n+1/2}, y^{n+1/2}) \tag{9.39}$$

Linear stability analysis:

$$y^{n+1/2} - y^n = \lambda \frac{\Delta t}{2} y^n$$

$$y^{n+1} - y^n = \lambda \Delta t \, y^{n+1/2}$$

Combining the predictor and the corrector steps:

$$y^{n+1} = y^n \, p(z)$$

with

$$p(z) = 1 + z + \frac{1}{2} z^2$$

The method is stable for all values of z such that $|p(z)| \leq 1$. The boundary of the stability region can be drawn by substituting $z := \rho\, e^{i\vartheta}$ in the polynomial

$$q(\rho;\, \vartheta) := p(z)\, p(z)^* - 1$$

and calculating the real roots of $q(z)$ for fixed value of ϑ. As for Heun's method, we get

$$q(\rho;\, \vartheta) = \frac{\rho^4}{4} + \rho^3 \cos(\vartheta) + 2\,\rho^2 \cos(\vartheta)^2 + 2\,\rho \cos(\vartheta)$$

Using MATLAB, the relevant commands to identify the real roots of $q(\rho;\, \vartheta)$ (after dividing through by ρ) are roots *and* isreal. *The boundary of the stability region for Heun's method is reported in figure 9.2.*

RK4 method (fourth-order)

Several different fourth-order Runge-Kutta methods (RK4) have been derived over the years. One very popular RK4 scheme consists of the following four steps:

1. *Explicit Euler predictor over* $(\Delta t)/2$*:*

$$y^* - y^n = \frac{\Delta t}{2} f(t_n, y^n)$$

2. *Implicit Euler corrector over* $(\Delta t)/2$*:*

$$y^{**} - y^n = \frac{\Delta t}{2} f(t_{n+1/2}, y^*)$$

3. Mid-point rule predictor over Δt:

$$y^{***} - y^n = \Delta t\, f(t_{n+1/2},\, y^{**})$$

4. Simpson quadrature rule corrector over Δt using the intermediate values of y already computed.

$$y^{n+1} - y^n = \frac{\Delta t}{6}\left[f(t_n,\, y^n) + 2\, f(t_{n+1/2},\, y^*) + 2\, f(t_{n+1/2},\, y^{**}) + f(t_{n+1},\, y^{***})\right]$$

The standard RK4 method can be recast in alternative form:

$$
\begin{aligned}
K_1 &= f(t_n, y_n) \\
K_2 &= f(t_n + h/2, y_n + K_1\, h/2) \\
K_3 &= f(t_n + h/2, y_n + K_2\, h/2) \\
K_4 &= f(t_n + h, y_n + K_3\, h) \\
y^{n+1} &= y^n + \frac{h}{6}\left[K_1 + 2\,K_2 + 2\,K_3 + K_4\right]
\end{aligned}
$$

The Butcher table for the standard RK4:

0	*0*	*0*	*0*	*0*
1/2	*1/2*	*0*	*0*	*0*
1/2	*0*	*1/2*	*0*	*0*
1	*0*	*0*	*1*	*0*
	1/6	*1/3*	*1/3*	*1/6*

There are several RK4 schemes widely used in practice, besides the standard *RK4 outlined above.*

The Butcher table for another RK4 candidate is as follows:

0	*0*	*0*	*0*	*0*
1/3	*1/3*	*0*	*0*	*0*
2/3	*-1/3*	*0*	*0*	*0*
1	*1*	*-1*	*1*	*0*
	1/8	*3/8*	*3/8*	*1/8*

9.2.8 Derivation of Runge-Kutta schemes

The Runge-Kutta methods can be derived following different approaches. Using the Taylor series expansion of both the numerical and the analytical solutions one can immediately derive conditions on the parameters characterizing a RK method. To this end, the coefficients in the two polynomial

expansions in h must be matched up to the desired order. Alternatively, specific conditions can be derived by requiring the satisfaction of specific constraints (e.g., energy conservation, limited dispersion or dissipation in the numerical solution of transport phenomena, etc.). Let's derive a family of RK methods with $s = 2$ steps. Where not explicitly mentioned, it is implicitly assumed that functions are evaluated at time t and for the value $\phi(t)$ of the dependent variable. E.g., f_t will mean

$$\frac{\partial f}{\partial t}\left(t,\, \phi(t)\right)$$

1. *Taylor expansion of the analytical solution:*

$$\phi(t+h) \,=\, \phi(t) \,+\, [f]\; h \,+\, [f_t \,+\, f_y\, f]\; \frac{h^2}{2} \,+\, O(h^3)$$

2. The Taylor expansion of the numerical solution y_{n+1} (approximating $\phi(t+h)$) returns:

$$y_{n+1} = \phi + [(c_1 + c_2)\, f]\, h + [c_2\,(a_2\, f_t + b_{2\,1}\, f\, f_y) + a_1\, c_1\, f_t]\, h^2 + O(h^3)$$

3. Matching like-terms in h yields the following non-linear system of equations:

$$p = 2 \implies c_1 + c_2 = 1$$

$$p = 3 \implies \begin{cases} c_1\, a_1 + c_2\, a_2 = \dfrac{1}{2} \\ c_2\, b_{2\,1} = \dfrac{1}{2} \end{cases}$$

4. Letting $a_1 = 0$ while using c_2 as free parameter, one gets the following one-parameter family

of two-steps, second-order Runge-Kutta methods (use $\beta \equiv 1/(2\,c_2)$*):*

$$\begin{aligned} K_1 &= f \\ K_2 &= f(t + \beta\, h, \phi + \beta\, h\, K_1) \\ y_{n+1} &= y_n + h\,[(1 - c_2)\, K_1 + c_2\, K_2] \end{aligned} \tag{9.40}$$

The *mid-point* RK2 method

Taking $c_2 = 1$ *in (9.40) yields the so-called* mid-point *method (2-step, second-order accurate):*

$$\begin{aligned} K_1 &= f \\ K_2 &= f(t + h/2, \phi + (h/2)\, K_1) \\ y_{n+1} &= y_n + h\, K_2 \end{aligned}$$

RK3 methods

$$y_{n+1} = y_n + h \sum_{j=1}^{3} K_j$$

$$K_1 = f(t_n, y_n)$$

$$K_2 = f(t_n + a_2\,h, y_n + b_{2\,1}\,h\,K_1)$$

$$K_3 = f(t_n + a_3\,h, y_n + b_{3\,1}\,h\,K_1 + b_{3\,2}\,h\,K_2)$$

Taylor expansion of the analytical solution:

$$\phi(t_n + 1) = y_n + f\,h + \frac{h^2}{2}\,(f_t + f_y\,f) + \frac{h^3}{6}\left(f_{tt} + 2f_{ty}f + f_{yy}f^2 + f_y\,f_t + f\,f_y^2\right) + \ldots$$

Taylor expansion of y_{n+1} w.r.t. h:

1. *Order 0:*

   $$y_n$$

   Perfect match.

2. *Order 1:*

$$h\, f\ (c_1 + c_2 + c_3)$$

Matching if

$$c_1\ +\ c_2\ +\ c_3\ =\ 1 \tag{9.41}$$

3. *Order 2:*

$$h^2\ [c_3\ (a_3\, f_t f + (b_{31} + b_{32})\ f\, f_y) + c_2\ (a_2\, f_t + b_{21}\, f\, f_y])$$

Matching if

$$a_3\, c_3\ +\ a_2\, c_2\ =\ \frac{1}{2} \tag{9.42}$$

and

$$c_3\,(b_{3\,1}\,+\,b_{3\,2})\,+\,c_2\,b_{2\,1}\,=\,\frac{1}{2} \tag{9.43}$$

4. *Order 3:*

$$h^3\,\left(c_2\,\left(\frac{a_2\,(a_2\,f_{tt}+b_{21}\,f\,f_{ty})}{2}+\frac{b_{21}\,f\,(a_2\,f_{ty}+b_{21}\,f\,f_{yy})}{2}\right)\right. \qquad +$$

$$\left.c_3\,\left(\frac{a_3\,(a_3\,f_{tt}+(b_{31}\,f+b_{32}\,f)\,f_{ty})}{2}+\frac{(a_3\,f_{ty}+(b_{31}\,f+b_{32}\,f)\,f_{yy})\,(b_{31}\,f+b_{32}\,f)}{2}+b_{32}\,\left(a_2\,f_t+b_{21}\,f\,f_y\right)\,f_y\right)\right)$$

Matching if:

$$a_2\,c_2\,+\,a_3^2\,c_3\,=\,\frac{1}{3} \tag{9.44}$$

$$c_2\,b_{2\,1}^2\,+\,c_3\,(b_{3\,1}\,+\,b_{3\,2})^2\,=\,\frac{1}{3} \tag{9.45}$$

$$c_2\, a_2\, b_{2\,1} + c_3\, a_3\, (b_{3\,1} + b_{3\,2}) = \frac{1}{3} \tag{9.46}$$

$$c_3\, b_{3\,2}\, a_2 = \frac{1}{6} \tag{9.47}$$

$$c_3\, b_{3\,2}\, b_{2\,1} = \frac{1}{6} \tag{9.48}$$

It can be shown that just 6 out of the 8 equations are independent. Indeed:

1. *Comparing (9.47) with (9.48) yields:*

$$a_2 = b_{2\,1} \tag{9.49}$$

2. *Subtracting (9.45) from (9.46) while using (9.49) yields:*

$$a_3 = b_{3\,1} + b_{3\,2} \tag{9.50}$$

3. *Replacing (9.49) and (9.50) in (9.41)–(9.48) leaves only (9.49) and (9.50) themselves, (9.41), (9.42), (9.47), then (9.44) in the form*

$$a_2^2\, c_2 \,+\, a_3^2\, c_3 \,=\, \frac{1}{3} \tag{9.51}$$

Therefore, we are left with 6 independent equations and 8 free parameters. Fixing e.g. $a_3 = 1$ *and* $b_{2\,1} = 1/2$ *results in:*

$$y_{n+1} \,=\, y_n \,+\, \frac{h}{6}\,[f(t_n, y_n) \,+\, 4\, f(t_n + h/2, y_n + (1/2)h\, K_1) \,+\, f(t_n + h, y_n - h\, K_1 \,+\, 2\, h\, K_2)]$$

Exercise 9.2.3 (Stability region of RK4) *Using MATLAB (or any programming language you like better), draw the boundary of the stability region for the RK4 method outlined above. Compare*

against Heun's method.

9.2.9 Error and step-size control with RK schemes

Runge-Kutta methods: error and step-size control

Runge-Kutta Fehlberg method:

- Combination of $s = 5, p = 4$ RK scheme
  with $s = 6, p = 5$ RK scheme

- The two RK schemes share 5 K_j terms are 6
  $\Rightarrow$ low computational overburden

0	0					
1/4	1/4	0	0	0	0	0
3/8	3/32	9/32	0	0	0	0
12/13	1932/2197	-7200/2197	7296/2197	0	0	0
1	439/216	-8	3680/513	-845/4104	0	0
1/2	-8/27	2	-3544/2565	1859/4104	-11/40	0
	16/135	0	6656/12825	28561/56430	-9/50	2/55
	25/216	0	1408/2565	2197/4104	-1/5	0

$s=6$ scheme

$s=5$ scheme

- At each time-step, the solution is advanced with both schemes.
- $LTE \approx y_n^{(s=6)} - y_n^{(s=5)}$

$$= \sum_{j=1}^{5} \left(c_j^{(s=6)} - c_j^{(s=5)} \right) K_j + c_6^{(s=6)} K_6$$

- Time-step update strategy:

$$LTE = O(h^5) = \alpha h^5$$

Fix tolerance TOL. Compute an updated value h_{new} for the time-step:

$$\left.\begin{aligned} \alpha h_{new}^5 &= TOL \\ \alpha h^5 &= LTE \end{aligned}\right\} \Rightarrow \left(\frac{h_{new}}{h}\right)^5 = \frac{TOL}{LTE} \Rightarrow h_{new} = h \left(\frac{TOL}{LTE}\right)^{1/5} \times 0.9$$

- If $LTE > TOL$, repeat the time-steps.

9.2.10 Other methods

Integrating both sides of (9.1) over the interval $[t_{n+1/2}, t_{n+3/2}]$ *while approximating the integrals via the mid-point rule, one gets:*

$$\dot{y}^{n+1} \, \Delta t \, = \, \Delta t \, f\left(t_{n+1}, \, y^{n+1}\right)$$

The time-derivative on left-hand side can be approximated using a polynomial interpolent through the points t_{n-1}, t_n, t_{n+1}, *yielding the second-order implicit scheme known as Gear's scheme:*

$$\frac{3\, y^{n+1} \, - \, 4\, y^n \, + \, y^{n-1}}{2} \, = \, \Delta t \, f\left(t_{n+1}, \, y^{n+1}\right) \tag{9.52}$$

9.3 Newmark's method

Newmark's method dates back to 1959, when it was proposed by the civil engineer N.M. Newmark for use in structural dynamics. The target system of ODEs is second-order, explicit in the acceleration term:

$$\boldsymbol{M}\ \ddot{\boldsymbol{x}} = \boldsymbol{f}\left(t, \boldsymbol{x}, \dot{\boldsymbol{x}}\right) \tag{9.53}$$

Newmark's method

- Numerical method for the solution of second-order ODEs

$$\underline{\underline{M}}\,\ddot{q} = g(t, q, \dot{q})$$

- In the following, $\underline{\underline{M}}$ will be combined with g to get:

$$\ddot{q} = M^{-1} g \equiv f(t, q, \dot{q})$$

- E.g.: $m\ddot{x} + c\dot{x} + q x = f(t)$

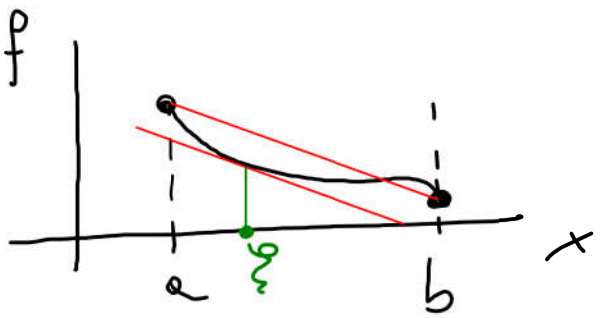
- Developed by Civil Eng Newmark (1959)

Derivation of Newmark's method invoking the Lagrange's theorem of differential calculus

- Recall Lagrange's theorem : (T1)

  Let $f : [a, b] \mapsto \mathbb{R}$ be continuous in $[a, b]$ and differentiable in (a, b). Then, $\exists\, \xi \in (a, b)$ s.t.

$$f'(\xi) = \frac{f(b) - f(a)}{b - a}$$



- Newmark's method provides a strategy to compute $\ddot{q}(t_n + h)$, $\dot{q}(t_n + h)$, $q(t_n + h)$ once $\ddot{q}(t_n)$, $\dot{q}(t_n)$, $q(t_n)$ are known.

- $\ddot{q}(t_n + h)$ is obtained directly from the ODE:

$$\ddot{q}(t_n + h) = f(t_n + h, q(t_n + h), \dot{q}(t_n + h))$$

- Invoking T1, $\exists\, \tilde{\gamma} \in (0, 1)$ s.t.

$$\ddot{q}(t_n + \tilde{\gamma} h) = \frac{\dot{q}(t_n + h) - \dot{q}(t_n)}{h}$$

Thus, $\dot{q}(t_n + h)$ is computed from:

$$\dot{q}(t_n + h) = \dot{q}(t_n) + h\, \ddot{q}(t_n + \tilde{\gamma} h)$$

So far, the method is both exact and useless, since T1 does not provide an algorithm to determine $\tilde{\gamma}$. Newmark's method invokes the following approximation: $\ddot{q}(t_n + \gamma h) = \ddot{q}(t_n) \cdot (1 - \gamma) + \ddot{q}(t_n + h)\, \gamma$ where γ is a <u>model parameter</u>, with $0 \le \gamma \le 1$.

Some choices for γ are noteworthy:

- $\gamma = 0$: $\dot{q}(t_n + h)$ is derived from an explicit stage:

$$\dot{q}(t_n + h) = \dot{q}(t_n) + h\,\ddot{q}(t_n)$$

- $\gamma = 1$: $\dot{q}(t_n + h)$ is derived from an entirely-implicit stage (i.e. no explicit contributions in the approximation for $\ddot{q}(t_n + \tilde{\gamma} h)$:

$$\dot{q}(t_n + h) = \dot{q}(t_n) + h\,\ddot{q}(t_n + h)$$

▶ Invoking again T1:

1a) $$q(t_n + h) = q(t_n) + h\,\dot{q}(t_n + \tilde{\beta} h)$$

Again, using T1:

$$\ddot{q}(t_n + \beta' h) = \frac{\dot{q}(t_n + \tilde{\beta} h) - \dot{q}(t_n)}{\tilde{\beta} h}, \qquad 0 < \beta' < \tilde{\beta}$$

$$\Rightarrow \quad 1b) \quad \dot{q}(t_n + \tilde{\beta} h) = \dot{q}(t_n) + \tilde{\beta} h \, \ddot{q}(t_n + \beta' h)$$

Combining 1a) with 1b):

$$q(t_n + h) = q(t_n) + h \left\{ \dot{q}(t_n) + \tilde{\beta} h \, \ddot{q}(t_n + \beta' h) \right\}$$

$$\Rightarrow 2) \quad q(t_n + h) = q(t_n) + h \dot{q}(t_n) + \tilde{\beta} h^2 \ddot{q}(t_n + \beta' h)$$

Newmark's method provides an approximation for the unknown terms in 2):

$$\tilde{\beta}\,\ddot{q}(t_n+\beta' h) \simeq \left(\frac{1}{2}-\beta\right)\ddot{q}(t_n) + \beta\,\ddot{q}(t_n+h)$$

In general, β is sought in the range $[0, 1/2]$, since $q(t)$ can be interpreted as an approximation to a truncated Taylor series:

$$q(t_n+h) = q(t_n) + h\dot{q}(t_n) + \frac{h^2}{2}\ddot{q}(t_n) + h.o.t.$$

Thus:

$$q(t_n+h) = q(t_n) + h\dot{q}(t_n) + \frac{h^2}{2}\left[(1-2\beta)\ddot{q}(t_n) + 2\beta\,\ddot{q}(t_n+h)\right]$$

At each time-step the vector $\underline{q} \equiv \left(q(t_n+h), \dot{q}(t_n+h), \ddot{q}(t_n+h)\right)^T$ must be calculated by solving an algebraic system of equations:

$$q(t_{n+1}) = q(t_n) + h\,\dot{q}(t_n) + \frac{h^2}{2}\left[(1-2\beta)\,\ddot{q}(t_n) + 2\beta\,\ddot{q}(t_{n+1})\right]$$

$$\dot{q}(t_{n+1}) = \dot{q}(t_n) + h\left[(1-\gamma)\,\ddot{q}(t_n) + \gamma\,\ddot{q}(t_{n+1})\right]$$

$$\ddot{q}(t_{n+1}) = f\left(t_{n+1}, q(t_{n+1}), \dot{q}(t_{n+1})\right)$$

$$\Rightarrow \begin{bmatrix} I & 0 & -h^2\beta I \\ 0 & I & -h\gamma I \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} q(t_{n+1}) \\ \dot{q}(t_{n+1}) \\ \ddot{q}(t_{n+1}) \end{bmatrix} = \begin{bmatrix} I & +hI & \frac{h^2}{2}(1-2\beta) \\ 0 & I & h(1-\gamma) \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} q(t_n) \\ \dot{q}(t_n) \\ \ddot{q}(t_n) \end{bmatrix} +$$

$$+ \begin{bmatrix} 0 \\ 0 \\ f\left(t_{n+1}, q(t_{n+1}), \dot{q}(t_{n+1})\right) \end{bmatrix}$$

- Derivation of Newmark's method from the integral form of the remainder of Taylor series.

$$y(b) - y(a) = \int_a^b y'(t)\,dt = \int_a^b y'(t)(t-b)^0\,dt$$

$$= y'(t)(t-b)^1\Big|_a^b - \int_a^b y''(t)(t-b)^1\,dt$$

$$= y'(a)(b-a) + \int_a^b y''(t)(b-t)\,dt$$

One more step:

$$\int_a^b y''(t)(b-t)\,dt = -y''(t)\frac{(b-t)^2}{2}\Big|_a^b + \int_a^b y'''(t)\frac{(b-t)^2}{2}\,dt$$

$$= y''(a)\frac{(b-a)^2}{2} + \int_a^b y'''(t)\frac{(b-t)^2}{2}\,dt$$

$$\Rightarrow y(b) - y(a) = y'(a)(b-a) + \frac{1}{2}y''(a)(b-a)^2 + \int_a^b y'''(t)\frac{(b-t)^2}{2}\,dt$$

Just another step:

$$\frac{1}{2}\int_a^b y'''(t)(b-t)^2\,dt = \frac{1}{2}\left\{-y''(t)\frac{(b-t)^3}{3}\Big|_a^b + \int_a^b y^{(iv)}(t)\frac{(b-t)^3}{3}\,dt\right\}$$

$$= \frac{1}{3!}\left\{y'''(a)(b-a)^3 + \int_a^b y^{(iv)}(t)(b-t)^3\,dt\right\}$$

Thus:

$$y(b) - y(a) = y'(a)(b-a) + \frac{1}{2}y''(a)(b-a)^2 + \frac{1}{3!}y'''(a)(b-a)^3 + \int_a^b \frac{y^{(iv)}(t)}{3!}(b-t)^3\,dt$$

The general form is:

$$\boxed{y(b) = \sum_0^k{}_j \frac{1}{j!}y^{(j)}(a)(b-a)^j + \int_a^b \frac{y^{(k+1)}(t)}{k!}(b-t)^k\,dt} \quad (1)$$

- Apply ① to $\dot{q}(t)$:

$$\dot{q}(t_n+h) = \dot{q}(t_n) + \int_{t_n}^{t_n+h} \ddot{q}(\tau)\, d\tau \qquad (2a)$$

- Apply ① to $q(t)$:

$$q(t_n+h) = q(t_n) + \dot{q}(t_n)\, h + \int_{t_n}^{t_n+h} \ddot{q}(\tau)\,(t_n+h-\tau)\, d\tau \qquad (2b)$$

- Provide a quadrature formula to approximate (2a) and (2b). Usually, 2-point formulas are used, though a multi-step or an RK approach can be conceived.
- One of the simplest approximations stems out from assuming $\ddot{q}(t)$ being linear in $[t_n, t_n+h]$:

$$\int_{t_n}^{t_n+h} \ddot{q}(\tau)\, d\tau = \frac{\ddot{q}_n + \ddot{q}_{n+1}}{2} h$$

$$\int_{t_n}^{t_n+h} \ddot{q}(\tau)\,(t_n + h - \tau)\, d\tau = \frac{h^2}{6}\left[2\ddot{q}_n + \ddot{q}_{n+1}\right]$$

- Substitution in (2a), (2b) yields:

(3a) $$\dot{q}_{n+1} = \dot{q}_n + (\ddot{q}_n + \ddot{q}_{n+1})\frac{h}{2} = \dot{q}_n + \left[(1-\gamma)\ddot{q}_n + \gamma \ddot{q}_{n+1}\right] h, \quad \gamma = \frac{1}{2}$$

(3b) $$q_{n+1} = q_n + \dot{q}_n h + \frac{h^2}{6}\left[2\ddot{q}_n + \ddot{q}_{n+1}\right] =$$

$$= q_n + \dot{q}_n h + \left[\left(\frac{1}{2} - \beta\right)\ddot{q}_n + \beta \ddot{q}_{n+1}\right] h^2, \quad \beta = \frac{1}{6}$$

The ODE adds physics to maths, providing an equation for $\ddot{q}_{n+1}$:

(3C) $$\ddot{q}_{n+1} = f(t_{n+1}, q_{n+1}, \dot{q}_{n+1})$$

We have just derived the linear acceleration method.

It is an implicit method.

$\beta = 1/6$

$\gamma = 1/2$

$$\begin{cases} q_{n+1} = q_n + \dot{q}_n h + \left[\left(\frac{1}{2} - \beta\right)\ddot{q}_n + \beta \ddot{q}_{n+1}\right] h^2 \\ \dot{q}_{n+1} = \dot{q}_n + \left[(1-\gamma)\ddot{q}_n + \gamma \ddot{q}_{n+1}\right] h \\ \ddot{q}_{n+1} = f(t_{n+1}, q_{n+1}, \dot{q}_{n+1}) \end{cases}$$

Applicazione : non-linear pendulum

$$\begin{cases} \ddot{\theta} + \sin\theta = 0 \\ \theta(0) = \theta_0 \\ \dot{\theta}(0) = 0 \end{cases}$$

$\beta = \frac{1}{6}$

$\gamma = \frac{1}{2}$

$$\begin{cases} \theta^{n+1} = \theta^n + \dot{\theta}^n h + \left[\left(\frac{1}{2} - \beta\right)\ddot{\theta}^n + \beta\,\ddot{\theta}^{n+1}\right] h^2 \\ \qquad = \theta^n + \dot{\theta}^n h - \left[\left(\frac{1}{2} - \beta\right)\sin\theta_n + \beta \sin\theta^{n+1}\right] h^2 \\ \dot{\theta}^{n+1} = \dot{\theta}^n + \left[(1-\gamma)\,\ddot{\theta}^n + \gamma\,\ddot{\theta}^{n+1}\right] h \\ \qquad = \dot{\theta}^n - \left[(1-\gamma)\sin\theta^n + \gamma \sin\theta^{n+1}\right] h \\ \ddot{\theta}^{n+1} = -\sin\theta^{n+1} \end{cases}$$

$\forall$ time-step we must solve a non-linear, algebraic system made of 3 equations ie 3 unknowns.

Using the Newton - Raphson method:

$$\begin{cases} \theta^{n+1} = \theta^n + \dot\theta^n h + \left[\left(\frac{1}{2}-\beta\right)\ddot\theta^n + \beta\,\ddot\theta^{n+1}\right]h^2 \\ \quad = \theta^n + \dot\theta^n h - \left[\left(\frac{1}{2}-\beta\right)\sin\theta_n + \beta\sin\theta^{n+1}\right]h^2 \\ \dot\theta^{n+1} = \dot\theta^n + \left[(1-\gamma)\ddot\theta^n + \gamma\,\ddot\theta^{n+1}\right]h \\ \quad = \dot\theta^n - \left[(1-\gamma)\sin\theta^n + \gamma\sin\theta^{n+1}\right]h \\ \ddot\theta^{n+1} = -\sin\theta^{n+1} \end{cases} \Rightarrow \underline{R} \equiv \begin{bmatrix} \theta^{n+1} + \beta h^2 \sin\theta^{n+1} - \left[\theta^n + \dot\theta^n h - \left(\frac{1}{2}-\beta\right)h^2\sin\theta^n\right] \\ \dot\theta^{n+1} + \gamma h \sin\theta^{n+1} - \left[\dot\theta^n - (1-\gamma)h\sin\theta^n\right] \end{bmatrix}$$

Residual vector

$$\underline{V} \equiv \begin{bmatrix} \theta^{n+1} \\ \dot\theta^{n+1} \end{bmatrix} \text{ vector of unknowns} \; ; \; \underline{\underline{J}} \equiv \frac{\partial \underline{R}}{\partial \underline{V}} = \begin{bmatrix} 1+\beta h^2\cos\theta & 0 \\ \gamma h\cos\theta & 1 \end{bmatrix}$$

$$\delta \underline{v} \equiv \begin{bmatrix} \delta\theta \\ \delta\dot{\theta} \end{bmatrix} \qquad \underline{R}(\underline{v}) = \underline{R}(\underline{v}^*) + \underline{\underline{J}}^* \delta \underline{v} + \cdots$$

$$\simeq \underline{R}(\underline{v}^*) + \underline{\underline{J}}^* \delta \underline{v}$$

Let $\underline{R}(\underline{v}) = \underline{0}$ to get: $\underline{\underline{J}}^* \delta \underline{v} = -\underline{R}(v^*)$ algebraic linear system

$$\begin{bmatrix} 1 + \beta h^2 \cos\theta^* & 0 \\ \gamma h \cos\theta^* & 1 \end{bmatrix} \begin{bmatrix} \delta\theta \\ \delta\dot{\theta} \end{bmatrix} = - \begin{bmatrix} R_1^* \\ R_2^* \end{bmatrix} \longrightarrow \begin{aligned} \delta\theta &= - \frac{R_1^*}{1 + \beta h^2 \cos\theta^*} \\ \delta\dot{\theta} &= - \left(R_2^* + \gamma h \cos\theta^* \delta\theta \right) \end{aligned}$$

lower-triangular matrix! System can readily solved by forward substitution.

Possible absolute stopping criteria:

$$\left.\begin{aligned} \delta\theta &< TOL_\theta \\ \delta\dot{\theta} &< TOL_{\dot{\theta}} \end{aligned}\right\} \text{ These can be based on physical ground}$$

$$\sqrt{(\delta\theta)^2 + (\delta\dot{\theta})^2} \leq \sqrt{(TOL_\theta)^2 + (TOL_{\dot{\theta}})^2} \qquad (L_2\text{-norm})$$

or

$$|\delta\theta| + |\delta\dot{\theta}| \leq TOL_\theta + TOL_{\dot{\theta}} \qquad (L_1\text{-norm})$$

Notice that both θ and $\dot{\theta}$ are non-dimensional !

Under-relaxation

Occasionally the iteration process may fail. Under such circumstances, try to "under-relax" the iteration process: $\delta \underline{v} = -\varepsilon \underline{\underline{J}}^{-1} \underline{R}^*$, $0 < \varepsilon \leq 1$.

Explicit Newmark method

Assume $\ddot{q} \simeq \ddot{q}_n$ in the integrals in equations (2a) and (2b).

The resulting explicit method is:

$$q^{n+1} = q^n + \dot{q}^n h + \ddot{q}^n \frac{h^2}{2}$$

$$\dot{q}^{n+1} = \dot{q}^n + \ddot{q}^n h$$

$$\ddot{q}^{n+1} = f(t_{n+1}, q^{n+1}, \dot{q}^{n+1})$$

} Can be derived also by simply truncating the Taylor series expansions for $q(t)$ and $\dot{q}(t)$.

Remark: this is equivalent to set $\gamma = \beta = 0$ in (3a), (3b).

How to derive a whole family of Newmark's methods

(4a) $\ddot{q}_n = \ddot{q}(t) + \dddot{q}(t)\,(t_n - t) + \cdots$

(4b) $\ddot{q}_{n+1} = \ddot{q}(t) + \dddot{q}(t)\,(t_{n+1} - t_n) + \cdots$

Consider two different convex linear combinations of (4a), (4b):

(5a) $\dot{q}(t) = (1-\gamma)\,\ddot{q}_n + \gamma\,\ddot{q}_{n+1} - \gamma\,\dddot{q}(t)\,(h - t - t_n) + \cdots$

(5b) $\ddot{q}(t) = (1-2\beta)\,\ddot{q}_n + 2\beta\,\ddot{q}_{n+1} - 2\beta\,\dddot{q}(t)\,(h - t - t_n) + \cdots$

Substitute in (2a) and (2b) and compute the integrals using an extended version of the mean-value theorem:

$$\int_a^b f(x)\, g(x)\, dx = f(\bar{x}) \int_a^b g(x)\, dx, \quad \bar{x} \in (a,b)$$

From (5a) and (7a):

$$\dot{q}_{n+1} = \dot{q}_n + \left[(1-\gamma)\ddot{q}_n + \gamma \ddot{q}_{n+1}\right] h - \gamma \frac{h^2}{2} \dddot{q}(\bar{t}) + \cdots$$

$$q_{n+1} = q_n + \dot{q}_n h + \left[\left(\tfrac{1}{2}-\beta\right)\ddot{q}_n + \beta \ddot{q}_{n+1}\right] h^2 - 2\beta \frac{h^3}{3} \dddot{q}(\bar{t}) + \cdots$$

Taking, e.g., $\gamma = \frac{1}{2}$ and $\beta = \frac{1}{4}$ yields

$$\dot{q}_{n+1} = \dot{q}_n + \ddot{q}_{ave}\, h$$

$$q_{n+1} = q_n + \dot{q}_n h + \ddot{q}_{ave}\, h^2/2 \qquad \text{where } \ddot{q}_{ave} \equiv \frac{\ddot{q}_{n+1} + \ddot{q}_n}{2}$$

Consistency

Newmark's method is said to be "consistent", since

$LTE \to 0$ as $h \to 0$.

Equivalently:

$$\frac{q_{n+1} - q_n}{h} = \dot{q}_n + \left[\left(\tfrac{1}{2} - \beta\right)\ddot{q}_n + \beta\,\ddot{q}_{n+1}\right] h \longrightarrow \dot{q}_n \quad \text{as } h \to 0$$

$$\frac{\dot{q}_{n+1} - \dot{q}_n}{h} = \left[(1-\gamma)\ddot{q}_n + \gamma\,\ddot{q}_{n+1}\right] = (1-\gamma) f\left(t, q(t), \dot{q}(t)\right)\Big|_{t=t_n} + \gamma f\left(t, q(t), \dot{q}(t)\right)\Big|_{t=t_n+h}$$

$$\longrightarrow f(t_n, q_n, \dot{q}_n) \equiv \ddot{q}_n \quad \text{as } h \to 0$$

9.3.1 Linear stability analysis of Newmark's method

Model system of equations:

$$\boldsymbol{M}\ \ddot{\boldsymbol{x}} + \boldsymbol{C}\ \dot{\boldsymbol{x}} + \boldsymbol{K}\,\boldsymbol{x} = \boldsymbol{p}(t) \tag{9.54}$$

Application of the Newmark method yields:

$$\boldsymbol{H}_1\ \overset{..n+1}{\boldsymbol{q}} = \boldsymbol{H}_0\ \overset{..n}{\boldsymbol{q}} + \boldsymbol{b}^{n+1} \tag{9.55}$$

$$\boldsymbol{H}_1 = \begin{bmatrix} \boldsymbol{M} + \gamma\,\Delta t\,\boldsymbol{C} & \gamma\,\Delta t\,\boldsymbol{K} \\ \\ \beta\,(\Delta t)^2\,\boldsymbol{C} & \boldsymbol{M} + \beta\,(\Delta t)^2\,\boldsymbol{K} \end{bmatrix} \tag{9.56}$$

$$\boldsymbol{H}_0 = -\begin{bmatrix} -\boldsymbol{M} + (1-\gamma)\,\Delta t\,\boldsymbol{C} & (1-\gamma)\,\Delta t\,\boldsymbol{K} \\ \left(\frac{1}{2}-\beta\right)(\Delta t)^2\,\boldsymbol{C} - \boldsymbol{M}\,\Delta t & -\boldsymbol{M} + \left(\frac{1}{2}-\beta\right)(\Delta t)^2\,\boldsymbol{K} \end{bmatrix} \tag{9.57}$$

$$\boldsymbol{b}^{n+1} = \begin{bmatrix} \gamma\,\Delta t\,\boldsymbol{p}^{n+1} + (1-\gamma)\,\Delta t\,\boldsymbol{p}^n \\ \beta\,(\Delta t)^2\,\boldsymbol{p}^{n+1} + \left(\frac{1}{2}-\beta\right)(\Delta t)^2\,\boldsymbol{p}^n \end{bmatrix} \tag{9.58}$$

$$\boldsymbol{q} = \begin{bmatrix} \dot{\boldsymbol{x}} \\ \\ \boldsymbol{x} \end{bmatrix} \tag{9.59}$$

The vector of unknowns $\boldsymbol{q}$ *can be updated by solving the system of algebraic equations (9.55):*

$$\ddot{\boldsymbol{q}}^{n+1} = \boldsymbol{H}_1^{-1} \left(\boldsymbol{H}_0 \ \ddot{\boldsymbol{q}}^{n} + \boldsymbol{b}^{n+1} \right) \equiv \boldsymbol{A}\, \boldsymbol{q}^n + \boldsymbol{g}^{n+1} \tag{9.60}$$

$$\boldsymbol{A} \equiv \boldsymbol{H}_1^{-1}\, \boldsymbol{H}_0; \quad \boldsymbol{g}^{n+1} \equiv \boldsymbol{H}_1^{-1}\, \boldsymbol{b}^{n+1} \tag{9.61}$$

In order to assess the stability of the method we investigate the amplification of an initial perturbation:

$$\delta \boldsymbol{q}_0 \equiv \boldsymbol{q}_0' - \boldsymbol{q}_0 \tag{9.62}$$

$$\delta \boldsymbol{q}^{n+1} \;=\; \boldsymbol{A}\,\delta \boldsymbol{q}^{n} \tag{9.63}$$

Assume $\boldsymbol{A}$ *admits a basis of eigenvectors:*

$$\boldsymbol{A} \;=\; \boldsymbol{P}^{-1}\,\boldsymbol{\Lambda}\,\boldsymbol{P} \tag{9.64}$$

being $\boldsymbol{\Lambda}$ *the diagonal matrix of eigenvalues.*

$$\delta \boldsymbol{u} \;\equiv\; \boldsymbol{P}\,\delta \boldsymbol{q} \tag{9.65}$$

$$\delta \boldsymbol{u}^{n+1} \;=\; \boldsymbol{\Lambda}\,\delta \boldsymbol{u}^{n} \tag{9.66}$$

Thus, the method is stable (for the specific considered problem) if the spectral radius of $\boldsymbol{A}$ *is not larger than* 1*. Conversely, the method is unstable.*

Undamped case

In this case $\boldsymbol{C} = \boldsymbol{0}$*. Invoking the modal-basis decomposition:*

$$\boldsymbol{M}^{-1}\,\boldsymbol{K} = \boldsymbol{Q}^T\,\boldsymbol{\Omega}\,\boldsymbol{Q} \tag{9.67}$$

where $\boldsymbol{\Omega}$ *is the diagonal matrix of the eigenvalues of* $\boldsymbol{M}^{-1}\,\boldsymbol{K}$*. Since both* $\boldsymbol{M}$ *and* $\boldsymbol{K}$ *are symmetric positive-definite matrices, the eigenvalues of* $\boldsymbol{M}^{-1}\,\boldsymbol{K}$ *are all real and positive. Hence they can be*

denoted as ω_i^2, $i = 1, \ldots, N$. *Letting* $\boldsymbol{y} := \boldsymbol{Q}\,\boldsymbol{x}$ *leads to:*

$$\ddot{\boldsymbol{y}} + \boldsymbol{\Omega}\,\boldsymbol{y} = \boldsymbol{Q}\,\boldsymbol{M}^{-1}\,\boldsymbol{p} \equiv \boldsymbol{p}' \tag{9.68}$$

Since $\boldsymbol{\Omega}$ *is a diagonal matrix, the equations in (9.68) are mutually independent of each other and can be solved on a one-by-one basis:*

$$\ddot{y}_i + \omega_i^2\, y_i = p_i' \tag{9.69}$$

The matrices resulting from the application of the Newmark method are, in this case:

$$\boldsymbol{H}_{1,i}(\Delta t) = \left[\begin{array}{c:c} 1 & \gamma\,\Delta t\,\omega_i^2 \\ \hdashline 0 & 1 + \beta\,(\omega_i\,\Delta t)^2 \end{array}\right] \tag{9.70}$$

$$\boldsymbol{H}_{0,i}(\Delta t) \;=\; \left[\begin{array}{c|c} 1 & -\,(1\,-\,\gamma)\;\Delta t\,\omega_i^2 \\ \hline \Delta t & 1\,-\,\left(\dfrac{1}{2}\,-\,\beta\right)\,(\omega_i\,\Delta t)^2 \end{array}\right] \tag{9.71}$$

$$\boldsymbol{A}_i \;=\; \boldsymbol{H}_{1,i}^{-1}\,\boldsymbol{H}_{0,i} \;=\; \left[\begin{array}{c|c} \dfrac{1-\gamma\widetilde{\omega}_i^2}{s_i^2} & -\dfrac{-\omega_i^2\,\Delta t\;(2\,s_i\,-\,\gamma\,\widetilde{\omega}_i^2)}{2\,s_i} \\ \hline \dfrac{\Delta t}{s_i} & \dfrac{2\,s_i\,-\,\widetilde{\omega}_i^2}{2\,s_i} \end{array}\right] \tag{9.72}$$

where

$$s_i \;\equiv\; 1\,+\,\beta\;(\omega_i\,\Delta t)^2\,;\quad \widetilde{\omega}_i \;\equiv\; \omega_i\,\Delta t \tag{9.73}$$

The characteristic polynomial associated to matrix $\boldsymbol{A}$ *is:*

$$p(\lambda) = \lambda^2 - \lambda \left[2 - \left(\gamma + \frac{1}{2} \right) \eta_i^2 \right] + 1 - \left(\gamma - \frac{1}{2} \right) \eta_i^2 \tag{9.74}$$

where

$$\eta_i^2 \equiv \frac{\widetilde{\omega}_i^2}{1 + \beta \widetilde{\omega}_i^2} \tag{9.75}$$

The characteristic polynomial depends on the parameters γ *and* η*:* γ *depends on the specific Newmark method used while* η *depends on the method (through* β*), on the specific physical problem (through* ω_i*) and on the time-step* Δt*.*

It can be shown (though it is a tough work) that the limiting case occurs when the discriminant of

the characteristic polynomial is negative:

$$(\gamma + 1/2)\,\eta^2 - 4\,\eta^2 < 0$$

In this case the characteristic polynomial has two complex-conjugate roots, whose magnitude is

$$\rho = \sqrt{1 - \eta^2\,(\gamma - 1/2)}$$

The scheme is stable if $\rho \leq 1$*, which yields the following sufficient condition for stability:*

$$\gamma \geq \frac{1}{2}; \quad \beta > \frac{(\gamma + 1/2)^2}{4} \tag{9.76}$$

The choice $\gamma = 1/2$, $\beta = 1/4$ *leads to an unconditionally-stable scheme of maximum accuracy.*

Application/example

1) Show how the following IVP can be solved by an implicit Newmark's method. You choose the specific method. Establish the stability limit the resulting algorithm.

$$m\ddot{x} + c\dot{x} + kx = p(t)$$

$$x(0) = x_0 \qquad \dot{x}(0) = v_0$$

Solution

$$\begin{cases} x_{n+1} = x_n + h\dot{x}_n + \frac{h^2}{2}\left[\ddot{x}_n(1-2\beta) + \ddot{x}_{n+1}2\beta\right] \\ \dot{x}_{n+1} = \dot{x}_n + h\left[\ddot{x}_n(1-\gamma) + \ddot{x}_{n+1}\gamma\right] \\ \ddot{x}_{n+1} = \frac{1}{m}\left[-c\dot{x}_{n+1} - kx_{n+1} + p(t_{n+1})\right] \end{cases}$$

Linear-acceleration method: $\gamma = 1/2$, $\beta = 1/6$.

$$x_{n+1} = x_n + h\dot{x}_n + h^2\beta\cdot\left\{\frac{1}{m}\left[-c\dot{x}_{n+1} - kx_{n+1} + p_{n+1}\right]\right\} + \frac{h^2}{2}(1-2\beta)\ddot{x}_n$$

$$x_{n+1}\left(1+\frac{k}{m}h^2\beta\right)+\dot{x}_{n+1}\left(\frac{ch^2\beta}{m}\right)=x_n+\dot{x}_n h+\frac{h^2}{2}(1-2\beta)\ddot{x}_n+\beta\frac{h^2}{m}P_{n+1}$$

$$\dot{x}_{n+1}=\dot{x}_n+h\left[(1-\gamma)\ddot{x}_n+\gamma\cdot\frac{1}{m}\left(-c\dot{x}_{n+1}-kx_{n+1}+P_{n+1}\right)\right]$$

$$\Rightarrow\quad x_{n+1}\left(\frac{\gamma hk}{m}\right)+\dot{x}_{n+1}\left(1+\frac{\gamma ch}{m}\right)=\dot{x}_n+h(1-\gamma)\ddot{x}_n+\frac{\gamma h}{m}P_{n+1}$$

$$\underline{\underline{A}}\equiv\begin{bmatrix}1+\frac{kh^2\beta}{m} & \frac{ch^2\beta}{m}\\ \frac{\gamma hk}{m} & 1+\frac{\gamma ch}{m}\end{bmatrix};\quad \underline{x}_{n+1}\equiv\begin{bmatrix}x_{n+1}\\ \dot{x}_{n+1}\end{bmatrix}$$

$$\underline{b}\equiv\begin{bmatrix}x_n+\dot{x}_n h+\frac{h^2}{2}(1-2\beta)\ddot{x}_n+\beta\frac{h^2}{m}P_{n+1}\\ \dot{x}_n+h(1-\gamma)\ddot{x}_n+\frac{\gamma h}{m}P_{n+1}\end{bmatrix}$$

At each time-step, solve:

$$\underline{\underline{A}}\,\underline{x}_{n+1} = \underline{b}$$

and compute

$$\ddot{x}_{n+1} = \frac{1}{m}\left[-c\,\dot{x}_{n+1} - k\,x_{n+1} + p_{n+1}\right]$$

Linear stability:

1) Consider the simpler case where $p \equiv 0$
2) Expand $\underline{b}$ to make explicit the contribution $\underline{x}_{n+1}$ from the previous time-step:

$$\underline{b} = \begin{bmatrix} x_n + \dot{x}_n h + \frac{h^2}{2}(1-2\beta)\frac{1}{m}\left(-c\,\dot{x}_n - k\,x_n\right) \\ \dot{x}_n + h(1-\gamma)\frac{1}{m}\left(-c\,\dot{x}_n - k\,x_n\right) \end{bmatrix} = \begin{bmatrix} 1 - \frac{k}{m}\frac{h^2}{2}(1-2\beta) & h - \frac{c}{m}\frac{h^2}{2}(1-2\beta) \\ \frac{k}{m}h(1-\gamma) & 1 - \frac{c}{m}h(1-\gamma) \end{bmatrix} \begin{bmatrix} x_n \\ \dot{x}_n \end{bmatrix}$$

$$\equiv \underline{\underline{B}}\,\underline{x}_n$$

$$\Rightarrow \underline{\underline{A}}\, \underline{x}_{n+1} = \underline{\underline{B}}\, \underline{x}_n$$

Define:

$$\omega_0^2 \equiv \frac{k}{m} \qquad 2\xi\omega_0 \equiv \frac{c}{m} \qquad \tilde{h} \equiv \omega_0 h$$

$$\underline{\tilde{x}} := \begin{bmatrix} x \\ \dot{x}/\omega_0 \end{bmatrix}$$

At each time-step, the following system of equations must be solved:

$$\underline{\underline{\tilde{A}}}\, \underline{\tilde{x}}_{n+1} = \underline{\underline{\tilde{B}}}\, \underline{\tilde{x}}_n$$

where:

$$\underline{\underline{\tilde{A}}} \equiv \begin{bmatrix} 1+\beta\tilde{h}^2 & 2\beta\xi\tilde{h}^2 \\ \gamma\tilde{h} & 1+2\xi\gamma\tilde{h} \end{bmatrix} \qquad \underline{\underline{\tilde{B}}} \equiv \begin{bmatrix} 1-(1-2\beta)\frac{\tilde{h}^2}{2} & \tilde{h}\left[1-(1-2\beta)\tilde{h}\xi\right] \\ \tilde{h}(\gamma-1) & 1-2\xi\tilde{h}(1-\beta) \end{bmatrix}$$

$\tilde{\underline{\underline{A}}}^{-1}\tilde{\underline{\underline{B}}}$ = ... (use Matlab Symbolic Toolbox A\B)

Let $\tilde{\underline{\underline{A}}}^{-1}\tilde{\underline{\underline{B}}} = \underline{\underline{P}}\,\underline{\underline{\Lambda}}\,\underline{\underline{P}}^{-1}$

Then:

$$\tilde{\underline{X}}_{u+1} = \underline{\underline{P}}\,\underline{\underline{\Lambda}}\,\underline{\underline{P}}^{-1}\tilde{\underline{X}}_u \Rightarrow \left(\underline{\underline{P}}^{-1}\tilde{\underline{X}}_{u+1}\right) = \underline{\underline{\Lambda}}\left(\underline{\underline{P}}^{-1}\tilde{\underline{X}}_u\right)$$

Thus, the linear stability of the method depends on the magnitude of the eigenvalues of $\tilde{\underline{\underline{A}}}^{-1}\tilde{\underline{\underline{B}}}$. → use Matlab Symbolic Toolbox eig(A\B)

As for the undamped case, $\xi = 0$,

```
syms gam bet xi h real positive
syms xn dxn ddxn

A = [1+h*h*bet   2*xi*bet*h*h;gam*h   1+2*xi*gam*h];
B = [1-0.5*(1-2*bet)*h*h   h*(1-xi*(1-2*bet)*h);-h*(1-gam)   1-2*xi*(1-gam)*h];

[P,D]=eig(A\B);

d = simplify(diag(D));

d_undamped = simplify(subs(d,'xi',0));

eq1 = d_undamped(1)^2-1;
eq2 = d_undamped(2)^2-1;
solve(eq1,h)
solve(eq2,h)
```

The magnitude of both eigenvalues is 1 if $h = \frac{\sqrt{2}}{\sqrt{\gamma - 2\beta}}$

$\Rightarrow \gamma \geq 2\beta$. For e.g. $\gamma = 1/2$ and $\beta = 1/4$ the scheme is uncond. stab

9.3.2 How to deal with non-linearities

*The system (**??**) can be recast in the following, more general form:*

$$\boldsymbol{F}\left(t_{n+1},\, \boldsymbol{z}^{n+1}\right) \,=\, 0 \tag{9.77}$$

If an explicit Newmark's method is used, the system of equations (9.77) is invariably linear with a diagonal coefficient matrix, so that an actual solution *process does not have to be undertaken. Conversely, implicit Newmark's methods require the solution of a system of algebraic equations at each time-step and the system is non-linear if the original ODE is. Assuming we have a reasonably good guess $\boldsymbol{z}^{n+1,k}$ for $\boldsymbol{z}^{n+1}$ at hand, as for instance $\boldsymbol{z}^{n+1,k} \equiv \boldsymbol{z}^{n}$, we may linearize the function $\boldsymbol{F}$*

about $\boldsymbol{z}^{n+1,k}$*:*

$$0 \;=\; \boldsymbol{F}\left(t_{n+1},\, \boldsymbol{z}^{n+1}\right) \;\approx\; \boldsymbol{F}\left(t_{n+1},\, \boldsymbol{z}^{n+1,k}\right) \;+\; \left.\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{z}}\right|_{t_{n+1},\, \boldsymbol{z}^{n+1,k}} \left(\boldsymbol{z}^{n+1} \;-\; \boldsymbol{z}^{n+1,k}\right) \tag{9.78}$$

Requiring that the rightmost term in (9.78) be zero, we end-up with a new approximation $\boldsymbol{z}^{n+1,k+1}$ *for* $\boldsymbol{z}^{n+1}$*:*

$$\begin{aligned} 0 \;&=\; \boldsymbol{F}\left(t_{n+1},\, \boldsymbol{z}^{n+1,k}\right) \;+\; \left.\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{z}}\right|_{t_{n+1},\, \boldsymbol{z}^{n+1,k}} \left(\boldsymbol{z}^{n+1} \;-\; \boldsymbol{z}^{n+1,k}\right) \\ \Longrightarrow \quad & \left.\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{z}}\right|_{t_{n+1},\, \boldsymbol{z}^{n+1,k}} \left(\boldsymbol{z}^{n+1,k+1} \;-\; \boldsymbol{z}^{n+1,k}\right) \;=\; -\boldsymbol{F}\left(t_{n+1},\, \boldsymbol{z}^{n+1,k}\right) \end{aligned} \tag{9.79}$$

The system of equations (9.79) is linear and can be solved with one of the many pertinent numerical methods. What we have been describing is, in its essence, the Newton-Rapson *method.*

*Application of the Newton-Rapson method to the Newmark's model (***??***) requires the differentiation*

of the mass-matrix $\boldsymbol{M}$ *(if depending on* $\boldsymbol{x}$ *or* $\dot{\boldsymbol{x}}$*) and of the - possibly non-linear - forcing term* $\boldsymbol{f}$*.*

Let's consider the simple case of a one-dimensional problem:

$$m\ \ddot{x}\ + c\ \dot{x}\ + k\, x\ =\ f \tag{9.80}$$

where m*,* c*,* k *and* f *may all be sources of non-linearities. When applying the Newmark's method, the only non-linear equation is the third one, i.e., the only one containing physics. Resolving the non-linearity of this equation with the Newton-Rapson approach yields:*

$$\boldsymbol{J}^*\, \delta \boldsymbol{z}\ =\ -R^* \tag{9.81}$$

where the Jacobian matrix $\boldsymbol{J}^*$ *at the previous iteration* $\boldsymbol{z}^*$ *reads*

$$
\begin{aligned}
&= \begin{bmatrix} k^* & c^* & m^* \end{bmatrix} \\
\boldsymbol{J}^* = &+ \begin{bmatrix}
m^*_{,x}\ \ddot{x}^* + & m^*_{\dot{,}x}\ \ddot{x}^* + & m^*_{\ddot{,}x}\ \ddot{x}^* + \\
c^*_{,x}\ \dot{x}^* + & c^*_{\dot{,}x}\ \dot{x}^* + & c^*_{\ddot{,}x}\ \dot{x}^* + \\
k^*_{,x}\, x^* - & k^*_{\dot{,}x}\, x^* - & k^*_{\ddot{,}x}\, x^* - \\
f^*_{,x} & f^*_{\dot{,}x} & f^*_{\ddot{,}x}
\end{bmatrix}
\end{aligned}
\tag{9.82}
$$

while the residual R^* *at the previous iteration is*

$$
R^* = m^*\ \ddot{x}^* + c^*\ \dot{x}^* + k^*\, x^* - f^* \tag{9.83}
$$

Combining the Newmark's method with the Newton-Rapson treatment of the physical equation yields:

$$\boldsymbol{A}^* \, \delta \boldsymbol{z} \; = \; \boldsymbol{b}^* \tag{9.84}$$

where the variation of $\boldsymbol{z}$ *is*

$$\delta \boldsymbol{z} \; \equiv \; \boldsymbol{z} \; - \; \boldsymbol{z}^* \tag{9.85}$$

the matrix $\boldsymbol{A}^*$ *is*

$$\boldsymbol{A}^* \; \equiv \; \begin{bmatrix} 1 & 0 & -\beta \, (\Delta t)^2 \\ 0 & 1 & -\gamma \, \Delta t \\ J_1^* & J_2^* & J_3^* \end{bmatrix} \tag{9.86}$$

and the right-hand side term $\boldsymbol{b}^*$ *is*

$$\boldsymbol{b}^* \equiv \begin{bmatrix} x^n - x^* + \dot{x}^n \ \Delta t + \dfrac{(\Delta t)^2}{2}\left[(1-2\beta)\,\ddot{x}^n + 2\beta\,\ddot{x}^*\right] \\ \dot{x}^n - \dot{x}^* + \Delta t\left[(1-\gamma)\,\ddot{x}^n + \gamma\,\ddot{x}^*\right] \\ -R^* \end{bmatrix} \tag{9.87}$$

9.3.3 Homework: undamped 1-DOF oscillator

$$\ddot{x} + \omega_0^2\, x = 0 \tag{9.88a}$$

$$\omega_0 = \pi \tag{9.88b}$$

$$x(0) = 1; \quad \dot{x}\,(0) = 0 \tag{9.88c}$$

Analytical solution:

$$x(t) \;=\; \cos\left(\omega_0\, t\right) \tag{9.88d}$$

It is requested to:

1. *Transform the second-order ODE into an equivalent system of first-order ODEs.*

2. *Solve the system of first-order ODEs using the forward-Euler, backward-Euler and the trapezoidal methods.*

3. *Solve the second-order ODE using Newmark method with different combinations of the* γ *and* β *parameters.*

4. *Compare all numerical results against the analytical solution on the time interval* $[0, T]$ *with* $T = 3s$*, using a time-step* $\Delta t = T/32$*.*

5. *Comment on the results.*

9.3.4 Homework: 1-DOF model for roll motion in beam waves

Theory: review lecture material of *Statica della Nave*

$$J_{xx}\ \ddot{\Phi} = M_{ai} + M_d + M_r + M_f \tag{9.89}$$

Added inertia:

$$M_{ai} = -J_a\ \ddot{\Phi} \tag{9.90}$$

Damping moment:

$$M_d = -B_{roll}\ \dot{\Phi} \tag{9.91}$$

Restoring moment:

$$M_r \;=\; -\,\Delta\,\overline{GM}\,\Phi \tag{9.92}$$

We end up with an equation analogue to that of the mass-spring-damper model: Damping moment:

$$(J_{xx} \;+\; J_a)\;\ddot{\Phi} \;+B_{roll}\;\dot{\Phi} \;+\Delta\,\overline{GM}\,\Phi \;=\; M_f \tag{9.93}$$

The forcing moment, due to the beam sea, is represented by the following model:

$$M_f \;=\; \Delta\,\overline{GM}\,r(\omega)\,\alpha_m\,\cos\left(\omega\,t \;+\; \Psi\right) \tag{9.94}$$

where:

- $r(\omega)$ *is the* effective wave slope coefficient, *depending on the wave angular frequency* ω.

- α_m *is the* maximum wave slope, *related to the wavenumber* K *and to the wave amplitude* $a \equiv H/2$ *by*

$$\alpha_m = K\,a = \frac{2\,\pi}{\lambda}\,\frac{H}{2} = \pi\,s \tag{9.95}$$

  with $s \equiv H/\lambda$.

- Ψ *is the phase, which can be set as* 0 *in the present application.*

Equation (9.93) can be recast in an equivalent form:

$$\ddot{\Phi} + 2\,\nu\,\omega_0\,\dot{\Phi} + \omega_0^2\,\Phi = \omega_0^2\,\pi\,r\,s\,\cos\left(\omega\,t + \Psi\right) \tag{9.96}$$

with

- ω_0 *natural angular frequency of the system:*

$$\omega_0^2 \equiv \frac{\Delta \overline{GM}}{J_{xx} + J_a} \tag{9.97}$$

- ν *damping coefficient:*

$$2\,\nu\,\omega_0 \equiv \frac{B_{roll}}{J_{xx} + J_a} \tag{9.98}$$

Common ranges of values for the aforementioned model parameters are as follows:

- $r = 0.5 \div 0.9$
- $s = 0.01 \div 0.05$

- $\nu = 0.01 \div 0.15$
- $\omega_0 = \dfrac{2\,\pi}{T_{roll}}$ *with* $T_{roll} = 15\,s \div 25\,s.$
- $\omega = 0.5\,\omega_0 \div 1.5\,\omega_0$

9.3.5 Homework

Code your favourite Newmark's method and solve the 1-DOF model for roll motion in beam sea, considering the following values for the model's parameters:

- $r = 0.7$
- $s = 0.02$
- $\nu = 0.01, 0.05, 0.1, 0.15$

- $T_{roll} = 20\,s.$
- $\omega = 0.9\,\omega_0$
- $\Phi(0) = 5^\circ$
- $\dot{\Phi}\,(0) = 0^\circ\,s^{-1}$

Compare the time-trace of roll angle, angular velocity and angular acceleration against their analytical counterparts.

9.4 MATLAB ODE solvers

Chapter 15

Ordinary Differential Equations

Mathematical models in many different fields.

Systems of differential equations form the basis of mathematical models in a wide range of fields – from engineering and physical sciences to finance and biological sciences. Differential equations are relations between unknown functions and their derivatives. Computing numerical solutions to differential equations is one of the most important tasks in technical computing, and one of the strengths of MATLAB.

If you have studied calculus, you have learned a kind of mechanical process for differentiating functions represented by formulas involving powers, trig functions, and the like. You know that the derivative of x^3 is $3x^2$ and you may remember that the derivative of $\tan x$ is $1 + \tan^2 x$. That kind of differentiation is important and useful, but not our primary focus here. We are interested in situations where the functions are not known and cannot be represented by simple formulas. We will compute numerical approximations to the values of a function at enough points to print a table or plot a graph.

Imagine you are traveling on a mountain road. Your altitude varies as you travel. The altitude can be regarded as a function of time, or as a function of longitude and latitude, or as a function of the distance you have traveled. Let's consider the latter. Let x denote the distance traveled and $y = y(x)$ denote the altitude. If you happen to be carrying an altimeter with you, or you have a deluxe GPS system, you can collect enough values to plot a graph of altitude versus distance, like the first plot in figure 15.1.

Suppose you see a sign saying that you are on a 6% uphill grade. For some

October 2, 2011

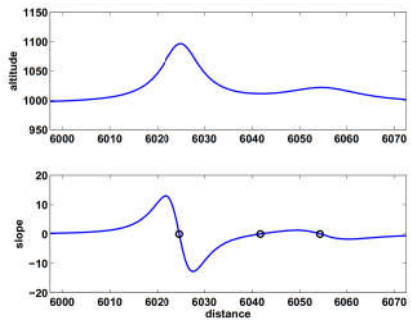


Figure 15.1. *Altitude along a mountain road, and derivative of that altitude. The derivative is zero at the local maxima and minima of the altitude.*

value of x near the sign, and for $h = 100$, you will have

$$\frac{y(x+h) - y(x)}{h} = .06$$

The quotient on the left is the *slope* of the road between x and $x + h$.

Now imagine that you had signs every few meters telling you the grade at those points. These signs would provide approximate values of the rate of change of altitude with respect to distance traveled, This is the derivative dy/dx. You could plot a graph of dy/dx, like the second plot in figure 15.1, even though you do not have closed-form formulas for either the altitude or its derivative. This is how MATLAB solves differential equations. Note that the derivative is positive where the altitude is increasing, negative where it is decreasing, zero at the local maxima and minima, and near zero on the flat stretches.

Here is a simple example illustrating the numerical solution of a system of differential equations. Figure 15.2 is a screen shot from Spacewar, the world's first video game. Spacewar was written by Steve "Slug" Russell and some of his buddies at MIT in 1962. It ran on the PDP-1, Digital Equipment Corporation's first computer. Two space ships, controlled by players using switches on the PDP-1 console, shoot space torpedoes at each other.

The space ships and the torpedoes orbit around a central star. Russell's program needed to compute circular and elliptical orbits, like the path of the torpedo in the screen shot. At the time, there was no MATLAB. Programs were written in terms of individual machine instructions. Floating-point arithmetic was so slow that it was desirable to avoid evaluation of trig functions in the orbit calculations. The orbit-generating program looked something like this.

```
x = 0
y = 32768
```

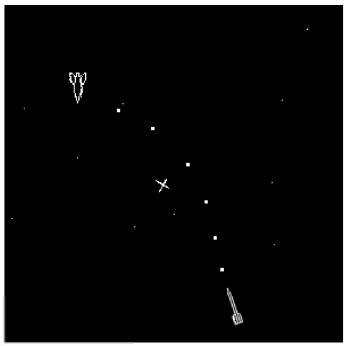


Figure 15.2. *Spacewar, the world's first video game. The gravitational pull of the central star causes the torpedo to move in an elliptical orbit.*

```
L: plot x y
   load y
   shift right 2
   add x
   store in x
   change sign
   shift right 2
   add y
   store in y
   go to L
```

What does this program do? There are no trig functions, no square roots, no multiplications or divisions. Everything is done with shifts and additions. The initial value of y, which is 2^{15}, serves as an overall scale factor. All the arithmetic involves a single integer register. The "shift right 2" command takes the contents of this register, divides it by $2^2 = 4$, and discards any remainder.

If Spacewar orbit generator were written today in MATLAB, it would look something the following. We are no longer limited to integer values, so we have changed the scale factor from 2^{15} to 1.

```
x = 0;
y = 1;
h = 1/4;
n = 2*pi/h;
plot(x,y,'.')
for k = 1:n
```

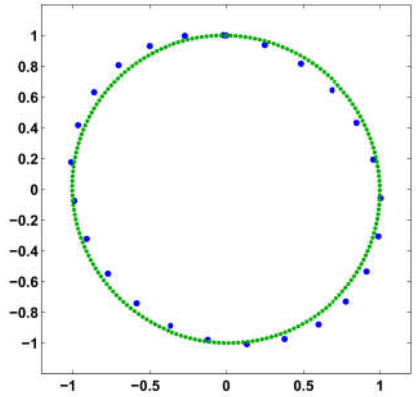


Figure 15.3. *The 25 blue points are generated by the Spacewar orbit generator with a step size of 1/4. The 201 green points are generated with a step size of 1/32.*

```
    x = x + h*y;
    y = y - h*x;
    plot(x,y,'.')
end
```

The output produced by this program with `h = 1/4` and `n = 25` is shown by the blue dots in figure 15.3. The blue orbit is actually an ellipse that deviates from an exact circle by about 7%. The output produced with `h = 1/32` and `n = 201` is shown by the green dots. The green orbit is another ellipse that deviates from an exact circle by less than 1%.

Think of `x` and `y` as functions of time, t. We are computing $x(t)$ and $y(t)$ at discrete values of t, incremented by the step size h. The values of x and y at time $t+h$ are computed from the values at time t by

$$\begin{aligned} x(t+h) &= x(t) + hy(t) \\ y(t+h) &= y(t) - hx(t+h) \end{aligned}$$

This can be rewritten as

$$\begin{aligned} \frac{x(t+h) - x(t)}{h} &= y(t) \\ \frac{y(t+h) - y(t)}{h} &= -x(t+h) \end{aligned}$$

You have probably noticed that the right hand side of this pair of equations involves

two different values of the time variable, t and $t + h$. That fact turns out to be important, but let's ignore it for now.

Look at the left hand sides of the last pair of equations. The quotients are approximations to the derivatives of $x(t)$ and $y(t)$. We are looking for two functions with the property that the derivative of the first function is equal to the second and the derivative of the second function is equal to the negative of the first.

In effect, the Spacewar orbit generator is using a simple numerical method involving a step size h to compute an approximate solution to the system of differential equations

$$\begin{aligned} \dot{x} &= y \\ \dot{y} &= -x \end{aligned}$$

The dot over x and y denotes differentiation with respect to t.

$$\dot{x} = \frac{dx}{dt}$$

The initial values of x and y provide the initial conditions

$$\begin{aligned} x(0) &= 0 \\ y(0) &= 1 \end{aligned}$$

The exact solution to the system is

$$\begin{aligned} x(t) &= \sin t \\ y(t) &= \cos t \end{aligned}$$

To see why, recall the trig identities

$$\begin{aligned} \sin(t+h) &= \sin t \cos h + \cos t \sin h \\ \cos(t+h) &= \cos t \cos h - \sin t \sin h \end{aligned}$$

For small values of h,

$$\begin{aligned} \sin h &\approx h, \\ \cos h &\approx 1 \end{aligned}$$

Consequently

$$\begin{aligned} \frac{\sin(t+h) - \sin t}{h} &\approx \cos t, \\ \frac{\cos(t+h) - \cos t}{h} &\approx -\sin t, \end{aligned}$$

If you plot $x(t)$ and $y(t)$ as functions of t, you get the familiar plots of sine and cosine. But if you make a *phase plane* plot, that is $y(t)$ versus $x(t)$, you get a circle of radius 1.

It turns out that the solution computed by the Spacewar orbit generator with a fixed step size h is an ellipse, not an exact circle. Decreasing h and taking more

steps generates a better approximation to a circle. Actually, the fact that $x(t+h)$ is used instead of $x(t)$ in the second half of the step means that the method is not quite as simple as it might seem. This subtle change is responsible for the fact that the method generates ellipses instead of spirals. One of the exercises asks you to verify this fact experimentally.

Mathematical models involving systems of ordinary differential equations have one *independent* variable and one or more *dependent* variables. The independent variable is usually time and is denoted by t. In this book, we will assemble all the dependent variables into a single vector y. This is sometimes referred to as the *state* of the system. The state can include quantities like position, velocity, temperature, concentration, and price.

In MATLAB a system of odes takes the form

$$\dot{y} = F(t, y)$$

The function F always takes two arguments, the scalar independent variable, t, and the vector of dependent variables, y. A program that evaluates $F(t, y)$ hould compute the derivatives of all the state variables and return them in another vector.

In our circle generating example, the state is simply the coordinates of the point. This requires a change of notation. We have been using $x(t)$ and $y(t)$ to denote position, now we are going to use $y_1(t)$ and $y_2(t)$. The function F defines the velocity.

$$\begin{aligned} \dot{y}(t) &= \begin{pmatrix} \dot{y}_1(t) \\ \dot{y}_2(t) \end{pmatrix} \\ &= \begin{pmatrix} y_2(t) \\ -y_1(t) \end{pmatrix} \end{aligned}$$

MATLAB has several functions that compute numerical approximations to solutions of systems of ordinary differential equations. The suite of ode solvers includes `ode23`, `ode45`, `ode113`, `ode23s`, `ode15s`, `ode23t`, and `ode23tb`. The digits in the names refer to the *order* of the underlying algorithms. The order is related to the complexity and accuracy of the method. All of the functions automatically determine the step size required to obtain a prescribed accuracy. Higher order methods require more work per step, but can take larger steps. For example `ode23` compares a second order method with a third order method to estimate the step size. while `ode45` compares a fourth order method with a fifth order method.

The letter "`s`" in the name of some of the ode functions indicates a *stiff* solver. These methods solve a matrix equation at each step, so they do more work per step than the nonstiff methods. But they can take much larger steps for problems where numerical stability limits the step size, so they can be more efficient overall.

You can use `ode23` for most of the exercises in this book, but if you are interested in the seeing how the other methods behave, please experiment.

All of the functions in the ode suite take at least three input arguments.

- `F`, the function defining the differential equations,
- `tspan`, the vector specifying the integration interval,

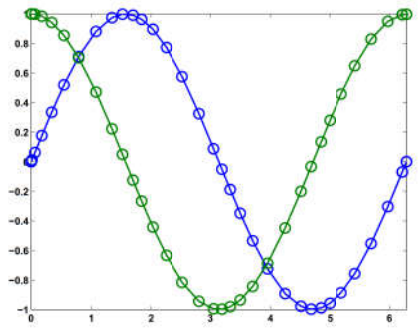


Figure 15.4. *Graphs of sine and cosine generated by ode23.*

- `y0`, the vector of initial conditions.

There are several ways to write the function describing the differential equation. Anticipating more complicated functions, we can create a MATLAB program for our circle generator that extracts the two dependent variables from the state vector. Save this in a file named `mycircle.m`.

```
function ydot = mycircle(t,y)
ydot = [y(2); -y(1)];
```

Notice that this function has two input arguments, `t` and `y`, even though the output in this example does not depend upon `t`.

With this function definition stored in `mycircle.m`, the following code calls `ode23` to compute the solution over the interval $0 \le t \le 2\pi$, starting with $x(0) = 0$ and $y(0) = 1$.

```
tspan = [0 2*pi];
y0 = [0; 1];
ode23(@mycircle,tspan,y0)
```

With no output arguments, the ode solvers automatically plot the solutions. Figure 15.4 is the result for our example. The small circles in the plot are not equally spaced. They show the points chosen by the step size algorithm.

To produce the phase plot plot shown in figure 15.5, capture the output and plot it yourself.

```
tspan = [0 2*pi];
y0 = [0; 1];
[t,y] = ode23(@mycircle,tspan,y0)
plot(y(:,1),y(:,2)'-o')
```

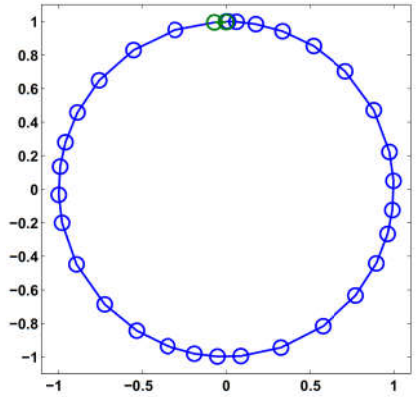


Figure 15.5. *Graph of a circle generated by ode23.*

```
axis([-1.1 1.1 -1.1 1.1])
axis square
```

The circle generator example is so simple that we can bypass the creation of the function file `mycircle.m` and write the function in one line.

```
acircle = @(t,y) [y(2); -y(1)]
```

The expression created on the right by the "`@`" symbol is known as an *anonymous function* because it does not have a name until it is assigned to `acircle`. Since the "`@`" sign is included in the definition of `acircle`, you don't need it when you call an ode solver.

Once `acircle` has been defined, the statement

```
ode23(acircle,tspan,y0)
```

automatically produces figure 15.4. And, the statement

```
[t,y] = ode23(acircle,tspan,y0)
```

captures the output so you can process it yourself.

Many additional options for the ode solvers can be set via the function `odeset`. For example

```
opts = odeset('outputfcn',@odephas2)
ode23(acircle,tspan,y0,opts)
axis square
axis([-1.1 1.1 -1.1 1.1])
```

will also produce figure 15.5.

Use the command

```
doc ode23
```

to see more details about the MATLAB suite of ode solvers. Consult the ODE chapter in our companion book, *Numerical Computing with MATLAB*, for more of the mathematical background of the ode algorithms, and for `ode23tx`, a textbook version of `ode23`.

Here is a very simple example that illustrates how the functions in the ode suite work. We call it "`ode1`" because it uses only one elementary first order algorithm, known as Euler's method. The function does not employ two different algorithms to estimate the error and determine the step size. The step size `h` is obtained by dividing the integration interval into 200 equal sized pieces. This would appear to be appropriate if we just want to plot the solution on a computer screen with a typical resolution, but we have no idea of the actual accuracy of the result.

```
function [t,y] = ode1(F,tspan,y0)
% ODE1  World's simplest ODE solver.
%   ODE1(F,[t0,tfinal],y0) uses Euler's method to solve
%      dy/dt = F(t,y)
%   with y(t0) = y0 on the interval t0 <= t <= tfinal.

t0 = tspan(1);
tfinal = tspan(end);
h = (tfinal - t0)/200;
y = y0;
for t = t0:h:tfinal
   ydot = F(t,y);
   y = y + h*ydot;
end
```

However, even with 200 steps this elementary first order method does not have satisfactory accuracy. The output from

```
[t,y]] = ode1(acircle,tspan,y0)
```

is

```
t =
   6.283185307179587
y =
   0.032392920185564
   1.103746317465277
```

We can see that the final value of `t` is `2*pi`, but the final value of `y` has missed returning to its starting value by more than 10 percent. Many more smaller steps would be required to get graphical accuracy.

Recap

```
%% Ordinary Differential Equations Chapter Recap
% This is an executable program that illustrates the statements
% introduced in the Ordinary Differential Equations Chapter
% of "Experiments in MATLAB".
% You can access it with
%
%    odes_recap
%    edit odes_recap
%    publish odes_recap
%
% Related EXM programs
%
%    ode1

%% Spacewar Orbit Generator.
    x = 0;
    y = 1;
    h = 1/4;
    n = 2*pi/h;
    plot(x,y,'.')
    hold on
    for k = 1:n
       x = x + h*y;
       y = y - h*x;
       plot(x,y,'.')
    end
    hold off
    axis square
    axis([-1.1 1.1 -1.1 1.1])

%% An Anonymous Function.
    acircle = @(t,y) [y(2); -y(1)];

%% ODE23 Automatic Plotting.
    figure
    tspan = [0 2*pi];
    y0 = [0; 1];
    ode23(acircle,tspan,y0)

%% Phase Plot.
    figure
    tspan = [0 2*pi];
    y0 = [0; 1];
    [t,y] = ode23(acircle,tspan,y0)
```

```
    plot(y(:,1),y(:,2),'-o')
    axis square
    axis([-1.1 1.1 -1.1 1.1])

%% ODE23 Automatic Phase Plot.
    opts = odeset('outputfcn',@odephas2)
    ode23(acircle,tspan,y0,opts)
    axis square
    axis([-1.1 1.1 -1.1 1.1])

%% ODE1 implements Euler's method.
% ODE1 illustrates the structure of the MATLAB ODE solvers,
% but it is low order and employs a coarse step size.
% So, even though the exact solution is periodic, the final value
% returned by ODE1 misses the initial value by a substantial amount.

    type ode1
    [t,y] = ode1(acircle,tspan,y0)
    err = y - y0
```

Exercises

15.1 *Walking to class.* You leave home (or your dorm room) at the usual time in the morning and walk toward your first class. About half way to class, you realize that you have forgotten your homework. You run back home, get your homework, run to class, and arrive at your usual time. Sketch a rough graph by hand showing your distance from home as a function of time. Make a second sketch of your velocity as a function of time. You do not have to assume that your walking and running velocities are constant, or that your reversals of direction are instantaneous.

15.2 *Divided differences.* Create your own graphic like our figure 15.1. Make up your own data, `x` and `y`, for distance and altitude. You can use

```
    subplot(2,1,1)
```

and

```
    subplot(2,1,2)
```

to place two plots in one figure window. The statement

```
    d = diff(y)./diff(x)
```

computes the *divided difference* approximation to the derivative for use in the second subplot. The length of the vector `d` is one less than the length of `x` and `y`, so you can add one more value at the end with

```
d(end+1) = d(end)
```

For more information about `diff` and `subplot`, use

```
help diff
help subplot
```

15.3 *Orbit generator.* Here is a complete MATLAB program for the orbit generator, including appropriate setting of the graphics parameters. Investigate the behavior of this program for various values of the step size `h`.

```
axis(1.2*[-1 1 -1 1])
axis square
box on
hold on
x = 0;
y = 1;
h = ...
n = 2*pi/h;
plot(x,y,'.')
for k = 1:n
   x = x + h*y;
   y = y - h*x;
   plot(x,y,'.')
end
```

15.4 *Modified orbit generator.* Here is a MATLAB program that makes a simpler approximation for the orbit generator. What does it do? Investigate the behavior for various values of the step size `h`.

```
axis(1.5*[-1 1 -1 1])
axis square
box on
hold on
x = 0;
y = 1;
h = 1/32;
n = 6*pi/h;
plot(x,y,'.')
for k = 1:n
   savex = x;
   x = x + h*y
   y = y - h*savex;
   plot(x,y,'.')
end
```

15.5 *Linear system* Write the system of differential equations

$$\begin{aligned} \dot{y}_1 &= y_2 \\ \dot{y}_2 &= -y_1 \end{aligned}$$

in matrix-vector form,

$$\dot{y} = Ay$$

where y is a vector-valued function of time,

$$y(t) = \begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix}$$

and A is a constant 2-by-2 matrix. Use our `ode1` as well as `ode23` to experiment with the numerical solution of the system in this form.

15.6 *Example from ode23.* The first example in the documentation for `ode23` is

$$\begin{aligned} \dot{y}_1 &= y_2\, y_3 \\ \dot{y}_2 &= -y_1\, y_3 \\ \dot{y}_3 &= -0.51\, y_1\, y_2 \end{aligned}$$

with initial conditions

$$\begin{aligned} y_1(0) &= 0 \\ y_2(0) &= 1 \\ y_3(0) &= 1 \end{aligned}$$

Compute the solution to this system on the interval $0 \le t \le 12$. Reproduce the graph included in the documentation provided by the command

```
doc ode23
```

15.7 *A cubic system.* Make a phase plane plot of the solution to the ode system

$$\begin{aligned} \dot{y}_1 &= y_2^3 \\ \dot{y}_2 &= -y_1^3 \end{aligned}$$

with initial conditions

$$\begin{aligned} y_1(0) &= 0 \\ y_2(0) &= 1 \end{aligned}$$

on the interval

$$0 \le t \le 7.4163$$

What is special about the final value, $t = 7.4163$?

15.8 *A quintic system.* Make a phase plane plot of the solution to the ode system

$$\begin{aligned} \dot{y_1} &= y_2^5 \\ \dot{y_2} &= -y_1^5 \end{aligned}$$

with initial conditions

$$\begin{aligned} y_1(0) &= 0 \\ y_2(0) &= 1 \end{aligned}$$

on an interval

$$0 \le t \le T$$

where T is the value between 7 and 8 determined by the periodicity condition

$$\begin{aligned} y_1(T) &= 0 \\ y_2(T) &= 1 \end{aligned}$$

15.9 *A quadratic system.* What happens to solutions of

$$\begin{aligned} \dot{y_1} &= y_2^2 \\ \dot{y_2} &= -y_1^2 \end{aligned}$$

Why do solutions of

$$\begin{aligned} \dot{y_1} &= y_2^p \\ \dot{y_2} &= -y_1^p \end{aligned}$$

have such different behavior if p is odd or even?

9.4.1 Application: kinematics of a point in circular orbit

Let $z \in \mathbb{C}$ represent the position of a point mass on a (complex) plane. If the point describes a circular trajectory with constant angular velocity ω, the time rate of change of its position is given by:

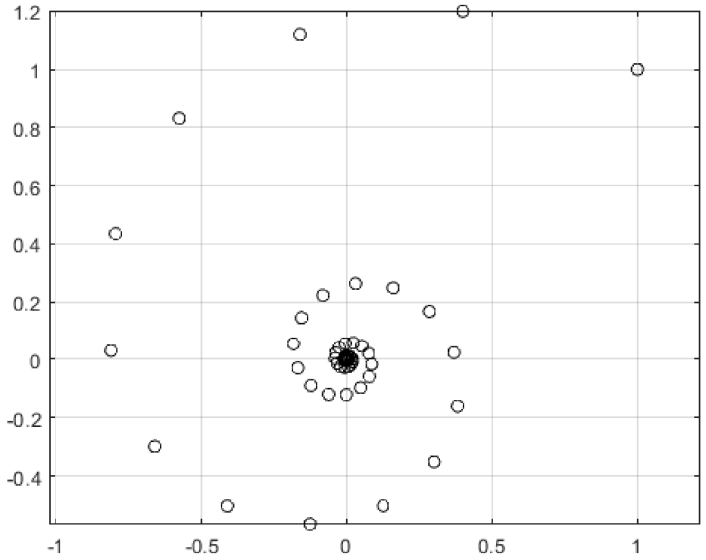
$$\dot{z} = i\omega\, z$$

where $z \equiv \rho\, e^{i\omega t}$ with $\rho \in \mathbb{R}^+$ is a constant. The particle departs from an initial position $z(0) = z_0$. A naive MATLAB implementation follows....

```
%{
  Kinematics: rotation of a point-mass with constant
  angular velocity.

Let $z\in\mathbb{C}$ represent the position of a poin
 mass on a (complex) plane. If the point describes
a circular trajectory with constant angular velocity
$\omega$, the time rate of change of its position is
given by:
\[
\dotd{z}\,=\,\textrm{i}\,\omega\,z
\]
where $z\,\equiv\,\rho\,\textrm{e}^{\textrm{i}\,\omega\,t}$
with $\rho\,\in\,\mathbb{R}^+$ is a constant.
The particle departs from an initial position
$z(0)\,=\,z_0$.

When using the ode15s solver, the complex-values scalar problem is converted
into a real-valued system of two first-order odes.
%}
function circular_path
close all
clc

z0     = 1+1i;
omega  = 2*pi;
T      = 20*2*pi/omega;
method = 'IE';
sample = 0.12;

t            = 0;
z            = z0;
h            = 0.5/omega;
samples      = complex(nan(ceil(T/sample),2));
samples(1,:) = [t,z0];
nsamples     = 1;
nstep        = 0;
while 1

    nstep = nstep + 1;

    if (t>(T-h))
        h = T-t;
    end

    t = t+h;

    if nstep>1
        znm1 = zn;
    end
    if nstep>2
        znm2 = znm1;
```

```
    end

    zn = z;

    switch method
        case{'Heun'}
            z = heun(@velocity,t,zn,h,omega);
        case{'AB3'}
            if nstep>3
                z = AB3(@velocity,t,zn,znm1,znm2,h,omega);
            else
                z = IE(@velocity,t,zn,h,omega);
            end

        case{'CN'}
            z = CN(@velocity,t,zn,h,omega);
        case{'IE'}
            z = IE(@velocity,t,zn,h,omega);
        otherwise
            break
    end

    if (samples(nsamples,1)>(t-sample))
        nsamples = nsamples+1;
        samples(nsamples,:) = [t,z];
    end

    if t>=T
        break;
    end
end

switch method
    case{'ode15s'}
        options = odeset('RelTol',1e-6);
      [t,z] = ode15s(@(t,z)rvelocity(t,z,{omega}),unique([0:sample:T,T]),
[real(z0);imag(z0)],options);
      samples = [t(:),z(:,1)+1i*z(:,2)];
end

figure
plot(real(samples(:,2)),imag(samples(:,2)),'ko');
axis equal
grid on

end

function f = velocity(t,z,parameters)

omega = parameters{1};

f = 1i*omega*z;

end
```

```
function f = rvelocity(t,z,parameters)

omega = parameters{1};

f = omega*[-z(2);z(1)];

end

function z=heun(funct,t,zn,h,omega)

fn = funct(t,zn,{omega});

z = zn + h*fn;

f = funct(t,z,{omega});

z = zn + 0.5*h*(f+fn);

end

function z=AB3(funct,t,zn,znm1,znm2,h,omega)

% Very inefficient implementation! Recalculation of
% fnm2 and fnm1 is not strictly needed as they could be stored
% and reused.

fnm2 = funct(t,znm2,{omega});

fnm1 = funct(t,znm1,{omega});

fn   = funct(t,zn  ,{omega});

z = zn + (h/12)*(23*fn-16*fnm1+5*fnm2);

end

function z=CN(funct,t,zn,h,omega)

z = zn * ((1+0.5*1i*omega*h)/(1-0.5*1i*omega*h));

end

function z=IE(funct,t,zn,h,omega)

z = zn / (1-1i*omega*h);

end
```



Published with MATLAB® R2021b

9.5 Selected applications of numerical methods for the solution of IVPs

9.5.1 The generic transport equation

$$\frac{\partial \varphi}{\partial t} = -u \frac{\partial \varphi}{\partial x} + \alpha \frac{\partial^2 \varphi}{\partial x^2}; \quad u \geq 0,\ \alpha \geq 0 \text{ constants} \tag{9.99}$$

Equation (9.99) is a widely-used model for the Navier-Stokes equations. As in general it is recommended to have at least a qualitative understanding of the main features of the considered problem and of its solutions, we first investigate in some detail a simpler form of equation (9.99), where the time derivative is dropped off.

The associated steady-state problem

$$0 = -u\frac{\partial\varphi}{\partial x} + \alpha\frac{\partial^2\varphi}{\partial x^2}; \quad u \geq 0,\ \alpha \geq 0 \text{ constants} \tag{9.100a}$$

Equation (9.100a) holds within the domain $(0, L)$*. Dirichlet boundary conditions apply on the boundaries:*

$$\varphi(x = 0) = \varphi_0 \tag{9.100b}$$

$$\varphi(x = L) = \varphi_L\ (\geq \varphi_0) \tag{9.100c}$$

The exact solution of (9.100a) – (9.100c) is:

$$\varphi(x) = \varphi_0 + \frac{e^{Pe_x} - 1}{e^{Pe_L} - 1}\ (\varphi_L - \varphi_0) \tag{9.101}$$

where the global (Pe_L) and cell (Pe_x) Péclet numbers are, respectively:

$$Pe_L = \frac{u\,L}{\alpha} \tag{9.102}$$

$$Pe_x = \frac{u\,x}{\alpha} \tag{9.103}$$

Physical interpretation of the Péclet number: is the ratio of the diffusion time-scale τ_d to the advection time-scale τ_c:

$$\frac{u\,L}{\alpha} = \frac{L^2\,/\,\alpha}{L\,/\,u}$$

When advection dominates ($\tau_c \ll \tau_d$) the Péclet number is large ($Pe_L \gg 1$). Conversely, when diffusion dominates, ($\tau_c \gg \tau_d$) the Péclet number is small ($Pe_L \ll 1$).

Let's consider what happens in the two aforementioned limit conditions:

- *When diffusion dominates:*

$$\lim_{Pe_L \longrightarrow 0} \varphi(x) = \varphi_0 + \frac{x}{L}\left(\varphi_L - \varphi_0\right),\ x \in [0,\, L]$$

- *When advection dominates:*

$$\lim_{Pe_L \longrightarrow +\infty} \varphi(x) = \varphi_0,\ x \in [0,\, L)$$

Notice that, when advection dominates, φ grows slowly with x and then rises suddenly to φ_L over a short distance close to $x = L$. Resolving such steep gradients is challenging for spatial-discretization schemes, as can be appreciated in figure 9.3.

The finite-difference representation of (9.100a), where a three-point stencil is used for both the advective and the diffusive terms, is:

$$A_P^j \, \varphi_j \;=\; A_E^j \, \varphi_{j+1} \;+\; A_W^j \, \varphi_{j-1} \tag{9.104}$$

As any numerical scheme should be able to represent a uniform solution, the coefficients A_P^j, A_E^j, A_W^j *must satisfy the following requirement:*

$$A_P^j \;=\; A_E^j \;+ A_W^j \tag{9.105}$$

Furthermore, the discretized transport equation should satisfy the physically-reasonable requirement that the solution at any grid point must increase if the solution at both the neighbouring points

increase. This property hold for the continuous equation (9.101), as can be easily verified by checking the sensitivity of the analytical solution to the boundary conditions:

$$\frac{\partial \varphi}{\partial \varphi_0} = 1 - \frac{e^{\frac{Pe\, x}{L}} - 1}{e^{Pe} - 1} \in [0,1] \text{ for } (x/L) \in [0,1]$$

$$\frac{\partial \varphi}{\partial \varphi_L} = \frac{e^{\frac{Pe\, x}{L}} - 1}{e^{Pe} - 1} \in [0,1] \text{ for } (x/L) \in [0,1]$$

Thus, a good *numerical approximation for the equation (9.100a) should satisfy*

$$A_P^j > 0; \quad A_E^j > 0; \quad A_W^j > 0$$

Let's now consider two widely used discretization schemes:

1. Central-difference scheme for both the diffusion and the advection terms (CDS-CDS scheme):

$$\begin{aligned} A_E^d &= \frac{\alpha}{h^2} \\ A_W^d &= \frac{\alpha}{h^2} \\ A_P^d &= A_E^d + A_W^d = 2\,\frac{\alpha}{h^2} \\ A_E^c &= -\,\frac{u}{2\,h} \\ A_W^c &= \frac{u}{2\,h} \\ A_P^c &= A_E^c + A_W^c = 0 \end{aligned} \tag{9.106}$$

The coefficient $A_E \equiv A_E^d + A_E^c$ *becomes negative when* $Pe_h > 2.0$*, suggesting that this nu-*

merical scheme might suffer of unphysical oscillations on excessively coarse meshes.

2. *Forward-difference scheme for the advection term and central difference scheme for the diffusion term (UDS-CDS scheme):*

$$\begin{aligned} A_E^d &= \frac{\alpha}{h^2} \\ A_W^d &= \frac{\alpha}{h^2} \\ A_P^d &= A_E^d + A_W^d = 2\,\frac{\alpha}{h^2} \\ A_E^c &= 0 \\ A_W^c &= \frac{u}{h} \\ A_P^c &= A_E^c + A_W^c = \frac{u}{h} \end{aligned} \tag{9.107}$$

The numerical solutions obtained with the two schemes are compared against the analytical solution in figure 9.4, for the case with $Pe_L = 50$ *and with either 11 or 41 grid points (corresponding to a cell Péclet numbers of* 5 *and* 1.25*, respectively). Some qualitative information can be drawn from a close examination of figure 9.4:*

- *the upwind-difference scheme for the advection term is over-diffusive: the numerical solution on the used coarse grid corresponds roughly to the analytical solution with* $Pe_L = 18$*.*

- *The central-difference scheme for the advection term suffers of severe unphysical oscillations: in the reported numerical simulation* $Pe_h = 5.0 > 2.0$*.*

- *When the mesh is refined, encompassing* 41 *nodes, the CDS-CDS numerical solution is oscillation-free and very accurate, while the UDS-CDS solution is still substantially in error*

for $x > 0.8$.

Analytical solution of the difference equation arising from the CDS-CDS discretization

$$A_E\,\varphi_{j+1} - (A_E + A_W)\,\varphi_j + A_W\,\varphi_{j-1} = 0$$

The characteristic polynomial associated to the difference equation is:

$$p(\vartheta) = A_E\,\vartheta^2 - (A_E + A_W)\,\vartheta + A_W$$

The roots of $p(\vartheta)$ *are:*

$$\begin{aligned} \vartheta_{1,2} &= \frac{(A_E + A_W) \pm |A_E - A_W|}{2\,A_E} \\ &= \frac{-2 \pm Pe_h}{-2 + Pe_h} \end{aligned}$$

The generic solution is thus:

$$\vartheta_j = A + B \left(-\frac{2 + Pe_h}{-2 + Pe_h} \right)^j$$

which, upon substitution of the boundary conditions, yields:

$$\frac{\vartheta_j - \varphi_0}{\varphi_L - \varphi_0} = \left(\frac{2 - Pe_h}{2 + Pe_h} \right)^{N-j}$$

The sign of the solution will change from one grid point to the next if $Pe_h > 2$*, confirming the heuristic result obtained above.*

Application of the numerical methods for IVPs to the unsteady transport equation

Both the advective and the diffusive terms will be discretized using the CDS scheme, unless otherwise noted. The following dimensionless groups are defined:

Definition 9.5.1 (CourantFriedrichsLewy number)

$$c := \frac{u\,\Delta t}{h} \tag{9.108}$$

Definition 9.5.2 (Diffusion stability number)

$$d := \frac{\alpha\,\Delta t}{h^2} \tag{9.109}$$

Explicit Euler time-stepping scheme

$$\varphi_j^{n+1} = (1 - 2\,d)\,\varphi_j^n + \left(d - \frac{c}{2}\right)\varphi_{j+1}^n + \left(d + \frac{c}{2}\right)\varphi_{j-1}^n \tag{9.110}$$

The heuristic criterion prescribing the positivity of the coefficients returns the following constraints:

$$d \leq \frac{1}{2} \tag{9.111}$$

$$Pe_h \leq 2 \tag{9.112}$$

Equation (9.111) is a stability constraint related to diffusion: notice that it is a rather severe requirement, in particular under conditions where the computational mesh is significantly refined in a

limited region of the computational domain (e.g., boundary layers). Indeed:

$$d \leq \frac{1}{2} \Longrightarrow \Delta t \leq \frac{h^2}{2\,\alpha}$$

When doubling the mesh refinement, the time-step must be reduced by a factor of four.

A more formal stability analysis can be carried out using the Von Neumann Analysis, which neglects the actual boundary conditions applied to the problem at hand while enforcing periodic boundary conditions. Let L denote the size of the computational domain. A uniform mesh is used, where h denotes the distance between subsequent nodes. Then, the Von Neumann Analysis looks for solutions in the form of complex exponentials (a more formal derivation of the method stems from the identification of the eigen-space of circulant matrices or, alternatively, from the related concept of Discrete Fourier

Transform*):*

$$\varphi_j^n \;=\; \sigma^n \, e^{i\,\widehat{k}\,x_j} \tag{9.113}$$

The wave-number k *is defined as:*

$$\widehat{k} \;:=\; \frac{2\,\pi\,k}{L},\; k \;\in\; \mathbb{N}$$

while the modified wavenumber *is given by*

$$\widetilde{k} \;:=\; \widehat{k}\,h \tag{9.114}$$

Furthermore,

$$x_j \;:=\; j\,h$$

The Nyquist sampling theorem requires that

$$\widetilde{k} \leq \pi \tag{9.115}$$

Substituting (9.113) in (9.110) yields a difference equation for σ, which can be rearranged as follows:

$$|\sigma|^2 = \left[1 + 2\,d\,\left(\cos\left(\widetilde{k}\right) - 1\right)\right]^2 + c^2\,\sin^2\left(\widetilde{k}\right) \tag{9.116}$$

The requirement for stability is

$$|\sigma|^2 \leq 1 \qquad \forall\,\widehat{k} \in [0,\,\pi] \tag{9.117}$$

For purely-diffusive problems, $c = 0$, the following diffusive stability constrain results:

$$d \leq \frac{1}{2} \tag{9.118}$$

The highest-order mode, $\widetilde{k} = \pi$, is also the most unstable.

For purely-convective problems, $d = 0$, the method turns out to be unconditionally unstable.

In the 1920's Courant and Friedrichs suggested a cure for the unconditional instabilty of the Explicit Euler time-stepping scheme, when applied to purely-convective problems: they recognized that the coefficient of φ^n_{j+1} is invariably negative if the advective term is approximated by the CDS finite-difference scheme. Therefore, they suggested to use the forward difference scheme for the discretization of the

advective term. The resulting difference equation reads:

$$\varphi_j^{n+1} = (1 - 2\,d - c)\,\varphi_j^n + d\,\varphi_{j+1}^n + (d + c)\,\varphi_{j-1}^n \tag{9.119}$$

The positivity of the coefficient of φ_j^n results in the following heuristic stability requirement:

$$1 - 2\,d - c > 0 \Longrightarrow \Delta t < \frac{1}{2\,\dfrac{\alpha}{h^2} + \dfrac{u}{h}} \tag{9.120}$$

For purely-convective problems the stability constraint simplifies as

$$c < 1 \tag{9.121}$$

Exercise 9.5.1 (Von Neumann analysis) *Apply the Von Neumann Analysis to derive the stability requirements for equation (9.119). In particular, confirm the stability requirement (9.121).*

Leapfrog and DuFort-Frankel time-stepping scheme *The Leapfrog method is an explicit scheme, consisting essentially in the application of the mid-point quadrature rule over an interval of size* $2\,\Delta t$*:*

$$\varphi_j^{n+1} \;=\; \varphi_j^{n-1} \;+\; (-4\,d)\;\varphi_j^n \;+\; (2\,d \;-\; c)\;\varphi_{j+1}^n \;+\; (2\,d \;+\; c)\;\varphi_{j-1}^n \tag{9.122}$$

The method is moderately unstable for diffusive problems (notice that the coefficient of φ_j^n *is negative whenever* $d > 0$*) while for purely-advective problems the stability requirement is* $c \leq 1$*.*

Exercise 9.5.2 (Von Neumann Analysis) *Apply the Von Neumann Analysis to derive the stability requirements for the Leapfrog method. In particular, confirm the stability requirements for*

purely advective and purely diffusive conditions.

The DuFort-Frankel method stems from the observation that the coefficient of φ_j^n *arising from the Leapfrog scheme is invariably negative. Thus, the unknown* φ_j^n*, arising from the CDS discretization of the second derivative, is replaced by the following approximation:*

$$\varphi_j^n \approx \frac{\varphi_j^{n-1} + \varphi_j^{n+1}}{2} \tag{9.123}$$

The aforementioned procedure yields the following difference equation for the DuFort-Frankel method:

$$(1 + 2\,d)\,\varphi_j^{n+1} = (1 - 2\,d)\,\varphi_j^{n-1} + (2\,d - c)\,\varphi_{j+1}^n + (2\,d + c)\,\varphi_{j-1}^n \tag{9.124}$$

The heuristic criterion about the positivity of the coefficients requires

$$d \leq \frac{1}{2}; \qquad Pe_h \leq 2 \tag{9.125}$$

A Von Neumann Analysis results in the following equation for the Fourier coefficient σ*:*

$$\sigma^2 \, (1 + 2\,d) + 2 \left[i c \sin(\widetilde{k}) - 2\,d \cos(\widetilde{k}) \right] \sigma - 1 + 1\,d = 0 \tag{9.126}$$

It can be verified that the method is stable for purely-advective problems as long as $c \leq 1$*, while it is unconditionally stable for purely diffusive problems.*

The truncation error of the DuFort-Frankel method is proportional to $(\Delta t \,/\, h)^2$*: this is an undesirable feature of the method, as it is consistent only if* Δt *tends faster to zero than* Δx*.*

Implicit Euler time-stepping scheme

$$(1 + 2\,d)\,\varphi_j^{n+1} + \left(\frac{c}{2} - d\right)\,\varphi_{j+1}^{n+1} + \left(-\frac{c}{2} - d\right)\,\varphi_{j-1}^{n+1} = \varphi_j^n \tag{9.127}$$

The heuristic criterion for the positivity of the coefficients states:

$$Pe_h \leq 2$$

The Von Neumann Analysis shows that the method is unconditionally stable.

Crank-Nicolson time-stepping scheme

$$\begin{aligned}\frac{\varphi_j^{n+1} - \varphi_j^n}{\Delta t} &= -\frac{1}{2}\left[u\,\frac{\varphi_{j+1}^{n+1} - \varphi_{j-1}^{n+1}}{2\,h} + u\,\frac{\varphi_{j+1}^{n} - \varphi_{j-1}^{n}}{2\,h}\right] \\ &+ \frac{1}{2}\left[\frac{\alpha}{h^2}\left(\varphi_{j+1}^{n+1} - 2\,\varphi_j^{n+1} + \varphi_{j-1}^{n+1}\right) + \frac{\alpha}{h^2}\left(\varphi_{j+1}^{n} - 2\,\varphi_j^{n} + \varphi_{j-1}^{n}\right)\right]\end{aligned} \tag{9.128}$$

The method is unconditionally stable. The coefficient for φ_j^n *becomes negative for* $d \geq 1$*: thus, the scheme can give rise to the onset of wiggles.*

Implicit three-time level time-stepping scheme *The time-discretization is based on the Gear time-stepping scheme. The resulting method is unconditionally unstable and less prone to develop wiggles than the Crank-Nicolson method, though the coefficient of* φ_j^{n-1} *is always negative. The truncation error is four times larger than for the Crank-Nicolson scheme. The method is not self-*

starting and is often blended with the Implicit Euler scheme.

$$(3 + 4\,d)\,\varphi_j^{n+1} + (-\,c - 2\,d)\,\varphi_{j-1}^{n+1} + (c - 2\,d)\,\varphi_{j+1}^{n+1} = 4\,\varphi_j^n - \varphi_j^{n-1} \tag{9.129}$$

9.6 Proficiency test

1. *Consider the following IVP:*

$$\begin{aligned} y' &= -\,iy \\ y(0) &= y_0 \end{aligned}$$

Can you solve it numerically using the explicit Euler method? Motivate your answer.

2. *Consider the following IVP:*

$$\begin{aligned} y' &= t^3 + 3\,t^2 - 4\,t - 1 \\ y(0) &= 1 \end{aligned}$$

Assume you aim to solve numerically this IVP using the explicit Euler method. Compute ***exactly*** *the LTE arising during the first time-step, using* $h = 0.1$.

3. *Consider the following IVP:*

$$\begin{aligned} y' &= -3\,y \\ y(0) &= 1 \end{aligned}$$

Consider solving this problem with the explicit Euler method, using $h = 0.1$. *Fill table* **??** *for*

the first 4 iterates and compute the LTE and the GTE for these time-steps. y_n denotes the numerical solution at the n-th step. ϕ_n denotes the analytical solution at the n-th step. $\widetilde{\phi}_n$ denotes the analytical solution at the n-th step starting from (t_{n-1}, y_{n-1}).

n	t	y_n	ϕ_n	$\widetilde{\phi}_n$
0				
1				
2				
3				
4				

4. *Consider solving an IVP with the AB2 method, using two different values of the time-step, h and $2\,h$. The numerical solutions at fixed time T are denoted by $y(h)$ and $y(2\,h)$, respectively. Conceive a strategy to provide an estimate for the GTE on $y(h)$ and to provide a better estimate for the actual solution of the IVP at time T. [Suggestion: Richardson extrapolation.]*

5. *Devise a single-step, second-order accurate method for the following IVP, using Taylor series expansion up to second-order.*

$$\begin{aligned} y' &= e^{-y} \\ y(0) &= 1 \end{aligned}$$

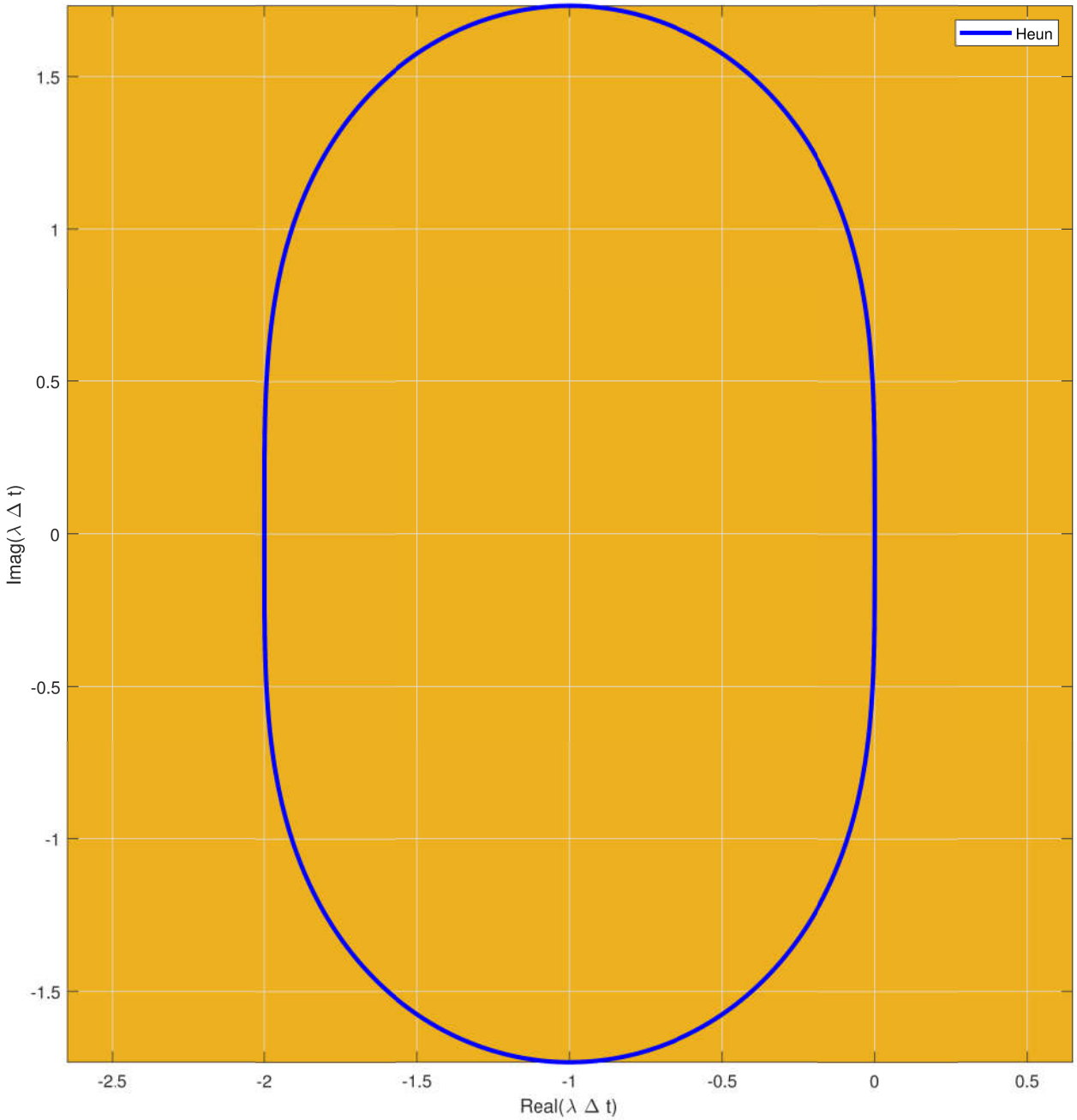


Figure 9.2: Boundary of the stability region of Heun's method.

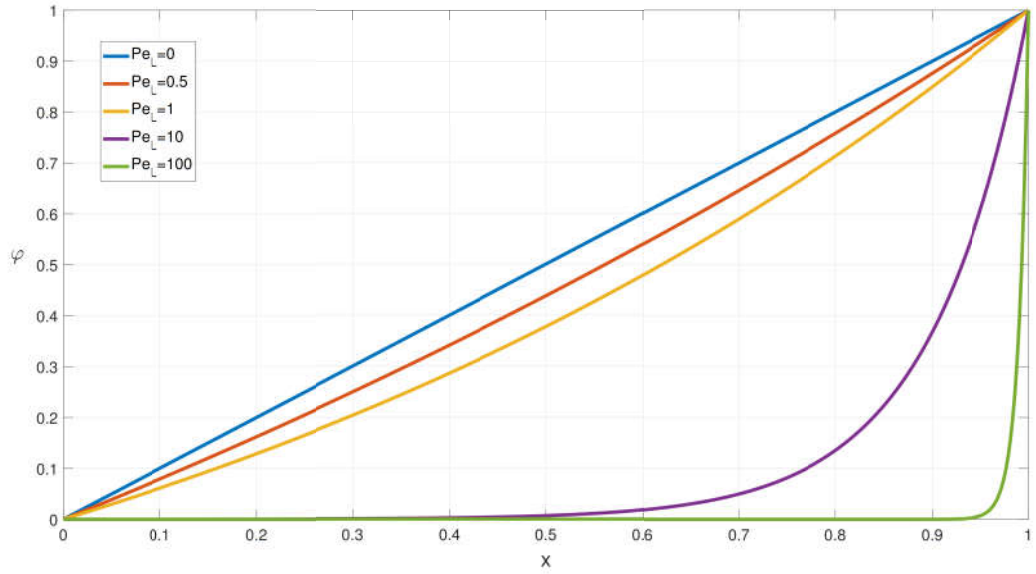
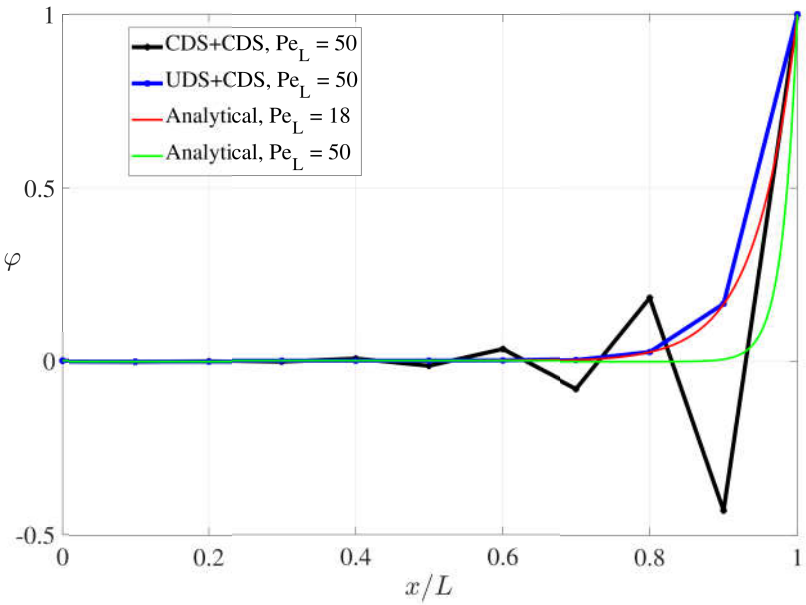
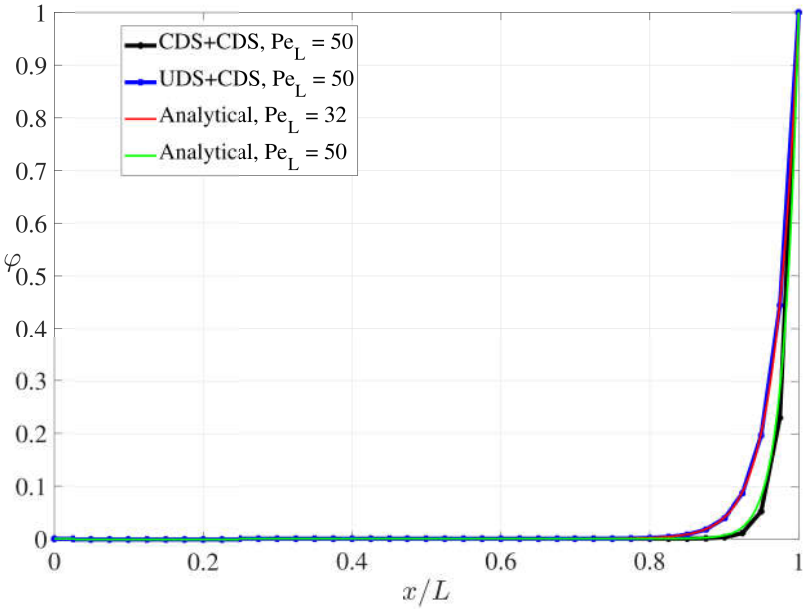


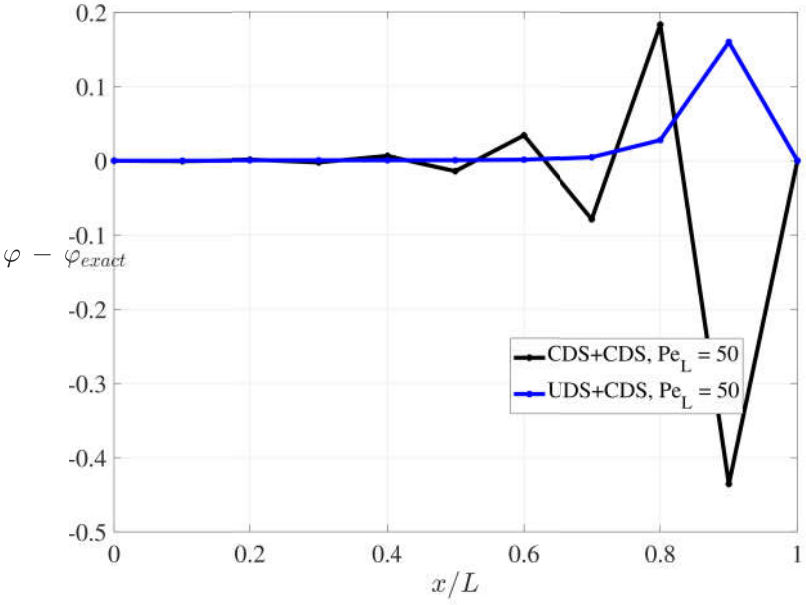
Figure 9.3: Solution of the steady convection-diffusion equation for different values of the Péclet number.

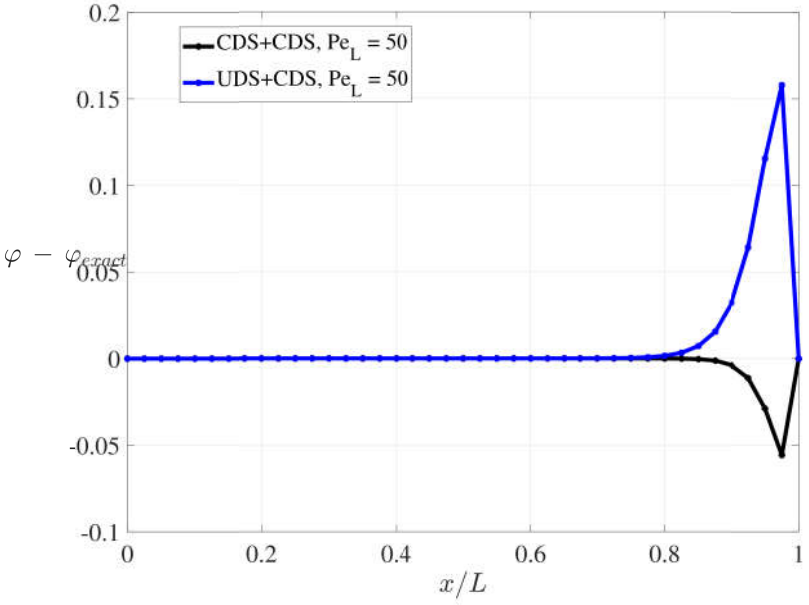


(a) Numerical solution obtained with either CDS-CDS or UDS-CDS scheme. Mesh with 11 nodes.



(b) Numerical solution obtained with either CDS-CDS or UDS-CDS scheme. Mesh with 41 nodes.





Chapter 10

Boundary Value Problems

10.1 Shooting method

10.2 Direct methods

10.2.1 Finite Difference Schemes

Direct methods for Boundary-Value Problems

- Boundary conditions incorporated into the discretized equations
- Several methods: FDM, FVM, FEM, BEM, Meshless methods, SPH, ...
- BVP methods form the basis for the numerical solution of PDEs.
- We are going to focus on second-order BVPs (most common in Mechanics).

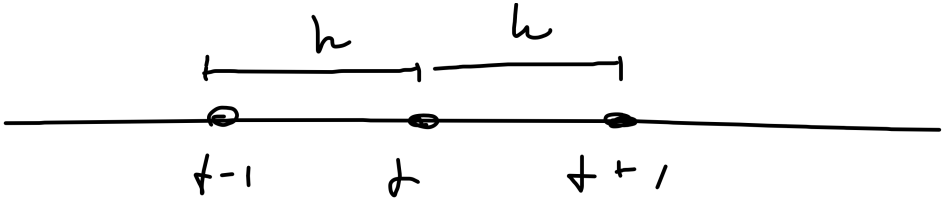
Finite-difference methods

$$y'' = f(x, y, y') \quad , x \in (a, b)$$

- Substitute y' and y'' with "finite-difference" approximations, i.e. approximations relying only on nodal values of y.
- Residual must vanish at each node
- End up with one algebraic equation per node
- Finite-differences can be derived:
  a) by differentiating interpolating polynomials
  b) by matching Taylor series expansions
  c) by divided differences

Example

FD approximation to y' on uniform grid



a) $\tilde{y}(x) = L_{j-1}\, y_{j-1} + L_j\, y_j + L_{j+1}\, y_{j+1}$

$$L_{j-1}(x) = \frac{\prod_{i \in \{j,j+1\}} (x - x_i)}{\prod_{i \in \{j,j+1\}} (x_{j-1} - x_i)} \qquad L_j(x) = \frac{\prod_{i \in \{j-1,j+1\}} (x - x_i)}{\prod_{i \in \{j-1,j+1\}} (x_j - x_i)} \qquad L_{j+1}(x) = \frac{\prod_{i \in \{j-1,j\}} (x - x_i)}{\prod_{i \in \{j-1,j\}} (x_j - x_i)}$$

$$y'(x_j) \simeq \tilde{y}'(x_j) = L'_{j-1}(x_j)\, y_{j-1} + L'_j(x_j)\, y_j + L'_{j+1}(x_j)\, y_{j+1} = \frac{y_{j+1} - y_{j-1}}{2h}$$

$$f''(x_j) \approx \tilde{y}''(x_j) = \frac{y_{j+1} - 2y_j + y_{j-1}}{h^2}$$

No direct outcome of approximation error (but can be derived using approximation results for interpolating polynomials).

b) $$y'(x_j) = \frac{1}{h}\left(a y_{j-1} + b y_j + c y_{j+1}\right) + \varepsilon(h)$$

Taylor expansion:

$$y_j' = \frac{1}{h}\left[a\left(y_j - y_j' h + \frac{1}{2}y_j'' h^2 - \frac{1}{6}y_j''' h^3 + \dots\right) + b y_j + c\left(y_j + y_j' h + \frac{1}{2}y_j'' h^2 + \frac{1}{6}y_j''' h^3 + \dots\right)\right]$$

$$+ \varepsilon(h)$$

$$\Rightarrow y_j' = \frac{1}{h}y_j\underbrace{(a+b+c)}_{A_0} + y_j'\underbrace{(-a+c)}_{A_1} + \left[y_j''\underbrace{\left(\frac{a}{2}+\frac{c}{2}\right)}_{A_2}\right]h + \left[y_j'''\left(-\frac{a}{6}+\frac{c}{6}\right)\right]h^2 + \dots + \varepsilon(h)$$

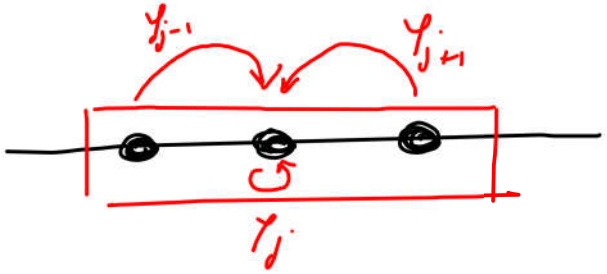
Letting $A_0 = 0$, $A_1 = 1$, $A_2 = 0$ yields $a = -\frac{1}{2}$, $c = \frac{1}{2}$, $b = 0$:

$$\Rightarrow \cancel{y'_j} = \frac{1}{h} y_j \underbrace{(a+b+c)}_{A_0} + y'_j \underbrace{(-a+c)}_{A_1} + \left[y''_j \underbrace{\left(\frac{a}{2}+\frac{c}{2}\right)}_{A_2} \right] h + \left[y'''_j \left(-\frac{a}{6}+\frac{c}{6}\right) \right] h^2 + \dots + \varepsilon(h)$$

symmetry of computational molecule "kills" odd terms

$$\Rightarrow \quad \varepsilon(h) = -\frac{1}{6} y'''_j h^2 + O(h^4)$$

$$\Rightarrow \quad y'_j = \frac{y_{j+1} - y_{j-1}}{2h} - \frac{1}{6} y'''_j h^2 + O(h^4)$$



II nd - order method.

Central (i.e., symmetric) 3-point stencil

Explicit (i.e, y'_j depends only on $y_{j-K} \dots y_{j+K}$)

CD2

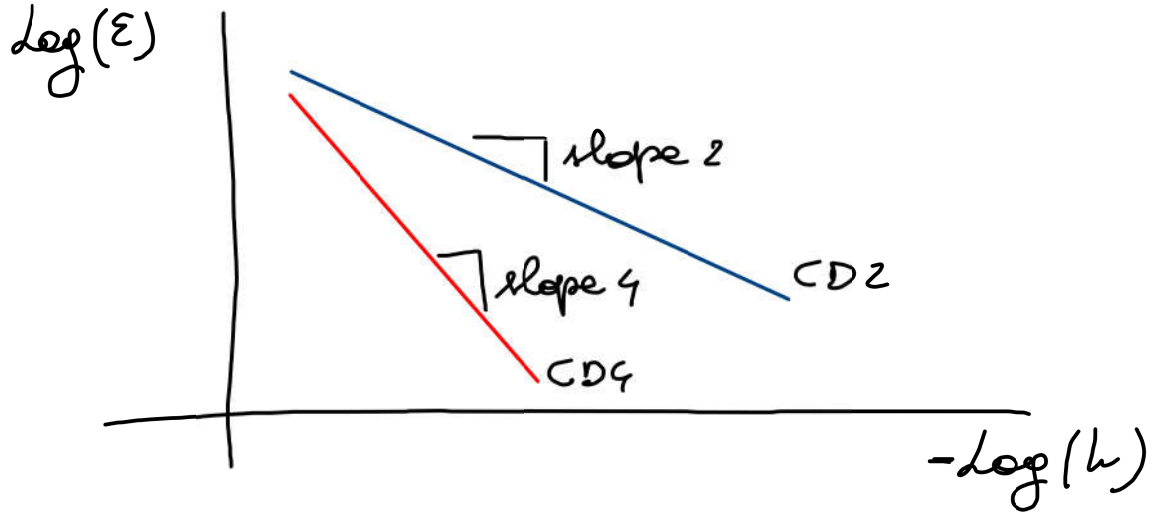
Exercise

- Derive $E(h)$ for the 3-point, symmetric approximation to y''_j
- Derive the FD formula and $E(h)$ for y'_j based on $y_{j-2}, y_{j-1}, y_j, y_{j+1}, y_{j+2}$. ($4^{th}$-order approximation) CD4
- Compare approximation error for CD2 and CD4 for the following functions $y(x)$:

$$y(x) = \cos(x) \quad \text{at } x = 0$$

$$y(x) = \sin(x) \quad \text{at } x = 0$$

Report $|\mathcal{E}(h)|$ in Log - Log plot:



$\mathcal{E}(h) \approx \alpha h^p \Rightarrow \text{Log}(\mathcal{E}(h)) \cong \text{Log}(\alpha) - p\,\text{Log}(1/h) \Rightarrow$ linear in Log - Log plot.

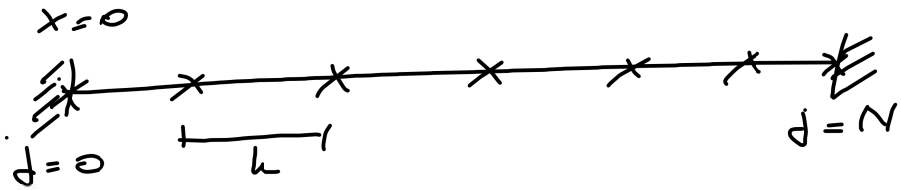
$$\mathcal{E}(2h)/\mathcal{E}(h) \simeq \alpha \cdot 2^p h^p / \alpha h^p = 2^p$$

$$\Rightarrow p = \frac{\ln\left(\mathcal{E}(2h)/\mathcal{E}(h)\right)}{\ln(2)}$$

Example : FD solution of BVP

$$y'' = f(x, y, y') \qquad x \in (0,1)$$

$$y(0) = A \qquad y(1) = B$$



Using CD2:

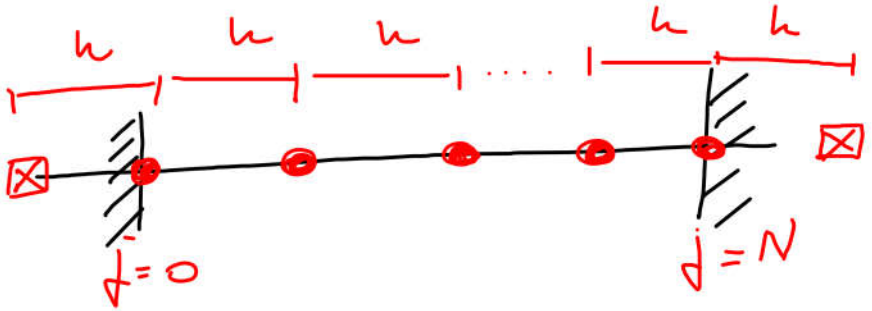
$$\frac{y_{j+1} - 2y_j + y_{j-1}}{h^2} = f\left(x_j, y_j, \frac{y_{j+1} - y_{j-1}}{2h}\right) \qquad j = 1 \text{ ---- } N-1$$

$$y_0 = A \qquad y_N = B$$

System of algebraic equations

Example, with more elaborated b.c.'s and ghost nodes

$$\begin{cases} y'' = f(x, y, y') & x \in (0,1) \\ a_1 y(0) + b_1 y'(0) = c_1 \\ a_2 y(1) + b_2 y'(1) = c_2 \end{cases}$$



⊠ ghost nodes

Discretize ODE for $j = 0 \dots N$:

a) $j = 0$: $\dfrac{y_1 - 2y_0 + y_{-1}}{h^2} = f\left(x_0, y_0, \dfrac{y_1 - y_{-1}}{2h}\right)$

b) $j = N$: $\dfrac{y_{N+1} - 2y_N + y_{N-1}}{h^2} = f\left(x_N, y_N, \dfrac{y_{N+1} - y_{N-1}}{2h}\right)$

$N+1$ equations but $(N+1)+2$ unknowns.

Get 2 more equations from approximation of boundary conditions:

$$a_1 y_0 + b_1 \frac{y_1 - y_{-1}}{2h} = c_1$$
$$a_2 y_N + b_2 \frac{y_{N+1} - y_{N-1}}{2h} = c_2$$

} (02 approximations to b. conds.

$$\Rightarrow \quad y_{-1} = 2h \frac{a_1}{b_1} y_0 + y_1 - 2h \frac{c_1}{b_1}$$

$$y_{N+1} = -2h \frac{a_2}{b_2} + y_{N-1} + 2h \frac{c_2}{b_2}$$

} substitute into equations a) and b) above to leave just $y_0 - - - - y_N$ as unknowns.

Example: approximation of b.c.'s via one-sided FD schemes:

c) $y_0' = \frac{y_1 - y_0}{h} + O(h)$

d) $y_N' = \frac{y_N - y_{N-1}}{h} + O(h)$

Approximate ODE with CD2 for $j = 1 ----- N-1$.

Then, use c), d) to approximate b. conds:

$$a_1 y_0 + b_1 \frac{y_1 - y_0}{h} = c_1$$

$$a_2 y_N + b_2 \frac{y_N - y_{N-1}}{h} = c_2$$

$\Rightarrow$ These approximations do not introduce additional unknowns other than $y_0 \cdots y_N$.

#

Example : FD solution of stiff problem

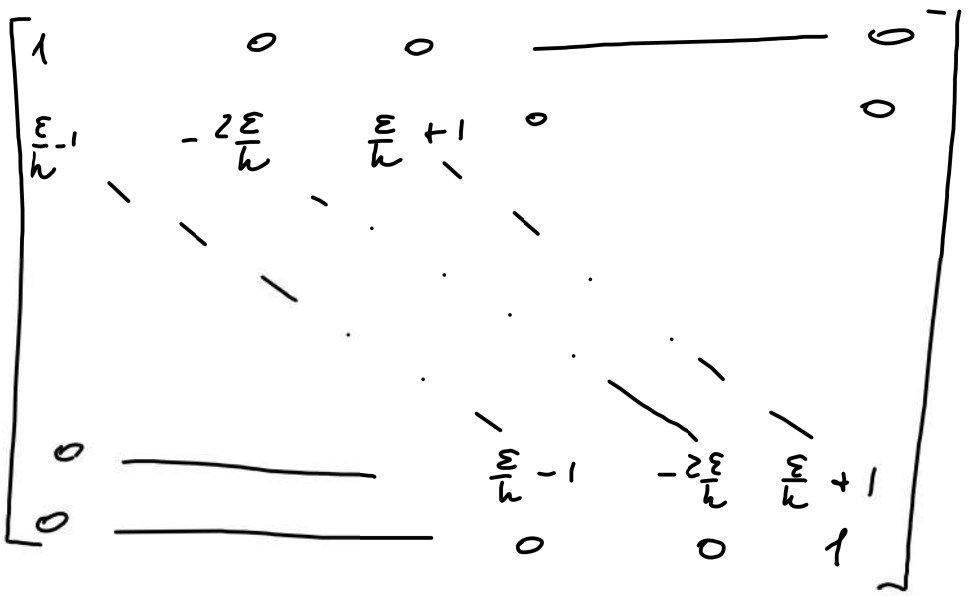
$$\begin{cases} \varepsilon y'' + y' = 0 & x \in (0, +\infty) \\ y(0) = 0 \\ y(x \to +\infty) = 1 \end{cases}$$

- Replace condition at $x \to +\infty$ with $y(b) = 1$, $b >> \varepsilon$.
  For $\varepsilon \to 0$ a steep b. layer develops at $x = 0$, with thickness $O(\varepsilon)$.
- CD2 approximation :

$$\varepsilon \frac{y_{j+1} - 2y_j + y_{j-1}}{h^2} + \frac{y_{j+1} - y_{j-1}}{2h} = 0 \qquad j = 1 \dots N-1$$

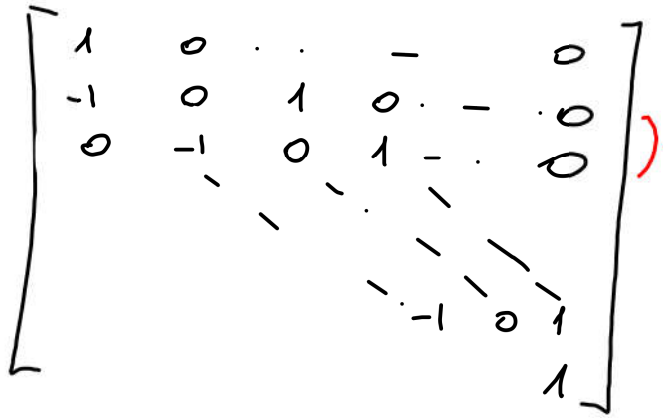
$$\Rightarrow y_{j-1}\left(\frac{\varepsilon}{h} - 1\right) - 2y_j\left(\frac{\varepsilon}{h}\right) + y_{j+1}\left(1 + \frac{\varepsilon}{h}\right) = 0 \qquad j = 1 \dots N-1$$

$$y_0 = 0 \qquad y_N = 1$$



coeff. matrix.

For $\frac{\varepsilon}{h} \to 0$:



→ odd- even equations un-related !

$$-y_{j-1} + y_{j+1} = 0 \qquad y_j = \sigma^j \qquad -\sigma^{j-1} + \sigma^{j+1} = 0$$

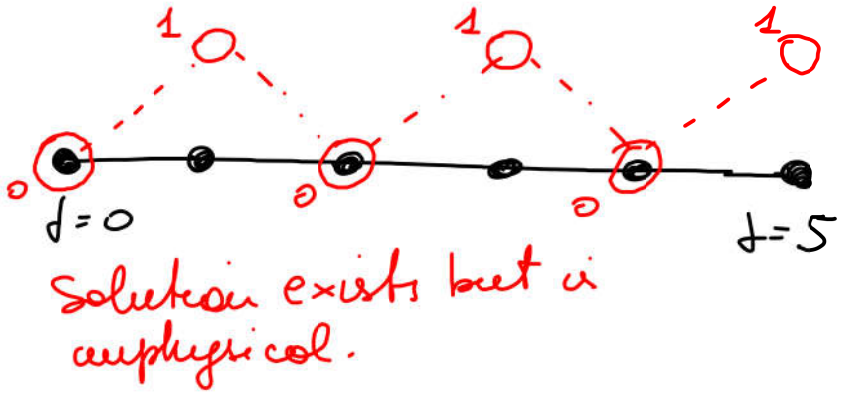
$$-1 + \sigma^2 = 0 \qquad \sigma = \pm 1 \Rightarrow \quad y_j = c_1 + c_2 (-1)^j$$

$$y_0 = 0 \Rightarrow \quad 0 = c_1 + c_2 \Rightarrow \quad y_j = c\left[1 + (-1)^{j+1}\right]$$

$$y_N = 1 \Rightarrow \quad 1 = c\left[1 + (-1)^{N+1}\right]$$

N odd: $\quad 1 = c\left[1 + \underbrace{(-1)^{N+1}}_{\text{even}}\right] = c \cdot 2 \Rightarrow c = 1/2$

$$y_j = \frac{1}{2}\left[1 + (-1)^{j+1}\right]$$



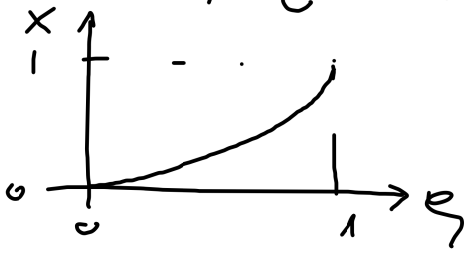
Solution exists but is unphysical.

Now: $1 = C \cdot [1 + (-1)^{N+1}] = 0$ ⚠ Impossible ⚠

$\Rightarrow$ solution does not exist (matrix is singular)

Thus, $\frac{\varepsilon}{h}$ must be significantly larger than 0, i.e., mesh must be fine enough to resolve b. layer ⚠ Best approach: use a non-uniform mesh clustered near $x = 0$.

$$\xi = \frac{1 - e^{-x/(a\varepsilon)}}{1 - e^{-1/(a\varepsilon)}}, \quad a > 1$$



$$\xi = f(x) \qquad x = f^{-1}(\xi) \qquad \frac{d\xi}{dx} = \frac{1}{\frac{dx}{d\xi}}$$

$$y_{,x} = y_{,\xi}\, \xi_{,x} = y_{,\xi} \cdot \left(\frac{1}{x_{,\xi}}\right)$$

$$y_{,xx} = \left(y_{,\xi}\, \xi_{,x}\right)_{,\xi} \xi_{,x} = y_{,\xi\xi} \left(\xi_{,x}\right)^2 + y_{,\xi} \left(\xi_{,x}\right)_{,\xi} \xi_{,x}$$

$$= y_{,\xi\xi} \frac{1}{(x_{,\xi})^2} + y_{,\xi} \cdot \left(-\frac{1}{x_{,\xi}}\right)^2 \cdot x_{,\xi\xi} \cdot \frac{1}{x_{,\xi}}$$

$$= \frac{1}{(x_{,\xi})^2} \left[y_{,\xi\xi} - \left(\frac{x_{,\xi\xi}}{x_{,\xi}}\right) y_{,\xi} \right]$$

$x_{,\xi}$, $x_{,\xi\xi}$: "metric terms"

$y_{,\xi}$, $y_{,\xi\xi}$, $x_{,\xi}$, $x_{,\xi\xi}$ are computed on uniform mesh.
This is both accurate and computationally efficient.

Boundary Layer is "resolved" : GOOD !

But ..... equations become more complex.

Remark : metric terms are usually approximated by FD on uniform ξ - mesh.

Application: finite-difference solution of BVP

Exercise: Finite-difference solution of a BVP

$$\begin{cases} y'' + k^2 y = 0 \quad , k \geq 0 \\ y(0) = 0 \quad y'(1) = 1 \end{cases} \quad , \quad x \in (0,1)$$

Analytical solution:

$\lambda^2 + k^2 = 0 \Rightarrow \lambda = \pm ik$

$y = A e^{ikx} + B e^{-ikx} = A_1 \cos(kx) + A_2 \sin(kx)$

$y(0) = 0 \Rightarrow A_1 = 0$

$y'(1) = 1 \Rightarrow A_2 k \cos k = 1 \Rightarrow A_2 = 1/(k \cos k)$

$$\boxed{y(x) = \frac{\sin(kx)}{k \cos(k)}}$$

Finite-difference solution:

- Uniform grid: $x_j = jh$, $j = 0 \dots N$

  $h = \frac{1}{N}$

  x_0, x_N: boundary nodes

- F.D. approach: residual of ODE has to be zero at each interior node.

  Given y, $R(y) \equiv y''(x) + k^2 y(x)$. $R(y) \equiv 0$ on $(0,1)$ iff y is the exact (strong-form) solution of the ODE.

  F.D. constraint: $R(x_j) = y''(x_j) + k^2 y(x_j) = 0$ for $j = 1 \dots N-1$ (1)

  The next step in the FD solution process consists in choosing the trial space for y, i.e. the function space where we look for the

solution of ①.

We define the trial space as follows:

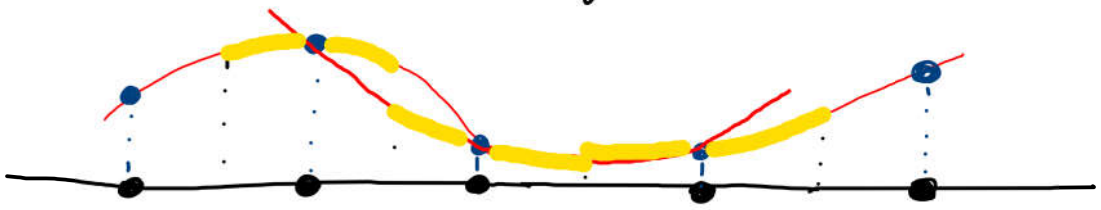
1) Given $N+1$ real numbers $y_0 \dots y_j \dots y_N$

2) $\forall$ interval $[x_{j-1/2}, x_{j+1/2}]$ consider the quadratic interpolant $\tilde{y}_j(x)$ through $(x_{j-1}, y_{j-1}), (x_j, y_j), (x_{j+1}, y_{j+1})$:

$$\tilde{y}_j(x) = \begin{cases} 0, & x \notin (x_{j-1/2}, x_{j+1/2}) \\ L_0(x)\, y_{j-1} + L_1(x)\, y_j + L_2(x)\, y_{j+1} \end{cases}$$

$\tilde{y} = \sum_{j=0}^{N} \tilde{y}_j$ trial function.

L_0, L_1, L_2: Lagrange coefficients for points x_{j-1}, x_j, x_{j+1}



Yellow curve: example of trial function

Remark: the definition of the trial functions near the boundaries usually differs from the interior, but the general idea does not.

Then, look for the trial function $\tilde{y}$ that satisfies the FD requirements:

$$R(\tilde{y})\Big|_{x_j} = \tilde{y}''(x_j) + k^2 y(x_j) = 0 \quad \forall j = 1 \dots N-1 \qquad (2)$$

and $\tilde{y}$ satisfies some discrete version of the boundary conditions.

Given that $\tilde{y}''(x_j) = \dfrac{y_{j+1} - 2y_j + y_{j-1}}{h^2}$, the conditions (2) yield:

$$\boxed{\frac{y_{j+1} - 2y_j + y_{j-1}}{h^2} + k^2 y_j = 0 \qquad j = 1 \dots N-1}$$

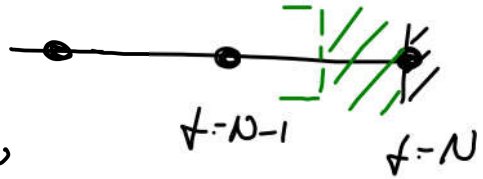
Furthermore $\tilde{y}$ must satisfy the boundary conditions:

$$\tilde{y}(0) = 0 \Rightarrow y_0 = 0 \quad \text{very easy!}$$

$\tilde{y}'(1) = 1$ .... How do we enforce this b.c.?

- "Special" definition of trial function near right boundary:

  Within the green hatched area:



$$\tilde{y}(x) = \frac{1}{2}\frac{\delta^2 y}{\delta x^2}(x - x_N)^2 + \frac{\delta y}{\delta x}(x - x_N) + y_N$$

$$\frac{\delta y}{\delta x} \equiv \frac{3y_N - 4y_{N-1} + y_{N-2}}{2h} \quad ; \quad \frac{\delta^2 y}{\delta x^2} \equiv \frac{y_N - 2y_{N-1} + y_{N-2}}{h^2}$$

$$\Rightarrow \tilde{y}'(x_N) = \frac{\delta y}{\delta x} = \frac{3y_N - 4y_{N-1} + y_{N-2}}{2h} = 1 \quad \text{b.c.}$$

Let's gather all equations arising from the FD procedure:

$$\begin{cases} y_0 = 0 \\ \dfrac{y_{j+1} - 2y_j + y_{j-1}}{h^2} + k^2 y_j = 0 \qquad j = 1 \dots N-1 \\ \dfrac{3y_N - 4y_{N-1} + y_{N-2}}{2h} = 1 \end{cases}$$

$$\Rightarrow \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & -2+\tilde{k}^2 & 1 & & 0 \\ & & & & \\ 0 & 0 & \cdots & 0 \boxed{1} & -4 \quad 3 \end{bmatrix} \begin{bmatrix} y_0 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 2h \end{bmatrix}$$

! This term makes the matrix no more tridiagonal

To maintain a tridiagonal coefficient matrix:

Eq. $j=1$:
$$\boxed{(-2+\tilde{k}^2)\,y_1 + 1\cdot y_2 = 0 - 1\cdot y_0 = 0}$$

Eq. $j=n$:
$$y_n = \frac{2}{3}h^2 y'(x=1) + \frac{4}{3}y_{n-1} - \frac{1}{3}y_{n-2}$$
$$= \frac{2}{3}h + \frac{4}{3}y_{n-1} - \frac{1}{3}y_{n-2}$$

Eq $j=n-1$:
$$1\cdot y_{n-2} + (-2+\tilde{k}^2)\,y_{n-1} + 1\cdot\left[\frac{2}{3}h + \frac{4}{3}y_{n-1} - \frac{1}{3}y_{n-2}\right] = 0$$

$$\boxed{\left(-\frac{1}{3}+1\right)\cdot y_{n-2} + \left(-2+\tilde{k}^2+\frac{4}{3}\right)y_{n-1} = -\frac{2}{3}h}$$

Let's gather all equations for $j = 1 \dots N-1$

$$\begin{bmatrix} -2+\tilde{k}^2 & 1 & 0 & \cdots & 0 \\ 1 & -2+\tilde{k}^2 & 1 & 0 & \cdots & 0 \\ 0 & 1 & -2+\tilde{k}^2 & 1 & 0 & \cdots & 0 \\ & & & & & & \\ & & & 1 & -2+\tilde{k}^2 & 1 & \\ 0 & \cdots & & 0 & \frac{2}{3} & -\frac{2}{3}+\tilde{k}^2 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ y_{N-1} \end{bmatrix} = \begin{bmatrix} 0 \\ \\ \\ \\ \\ \\ -\frac{2}{3}h \end{bmatrix}$$

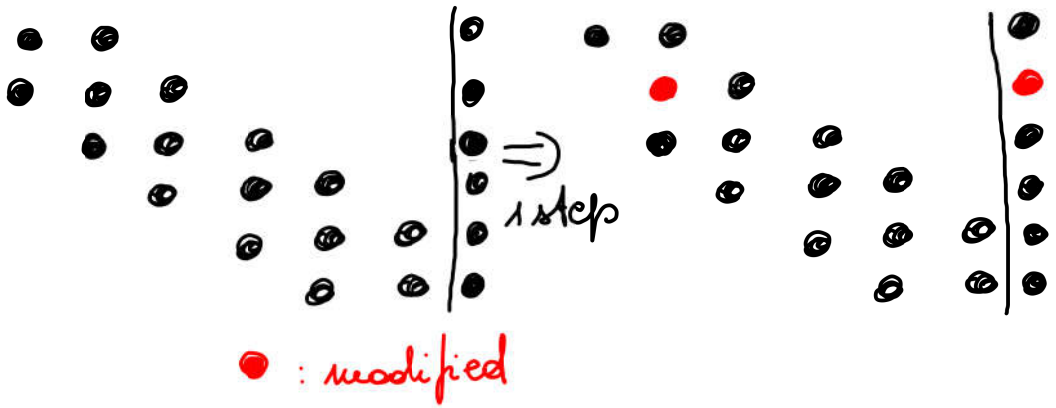
The coeff. matrix is indeed tridiagonal.

Gaussian elimination (without pivoting) for tridiagonal matrices: Thomas' algorithm

$$l_j y_{j-1} + d_j y_j + u_j y_{j+1} = b_j \qquad j = 1 \dots n$$

$$l_1 = 0; \quad u_n = 0.$$

Forward elimination:



● : modified

for $j = 1 : n-1$

$m \leftarrow \frac{l_{j+1}}{d_j}$

$d_{j+1} \leftarrow d_{j+1} - m\, u_j$

$b_{j+1} \leftarrow b_{j+1} - m\, b_j$

Need

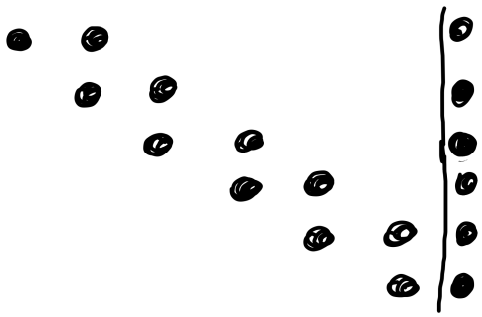
$n-1$ divisions

$2(n-1)$ multiplications

$2(n-1)$ additions/subtractions

$3(n-1)$ assignements

At the end of the forward elimination:



Back substitution:

$y_N = b_N / d_N$

for $j = N-1 : -1 : 1$

$y_j = (b_j - u_j y_{j+1}) / d_j$

end

N divisions
$N-1$ multiplications
$N-1$ additions/substractions
N assignements.

Let "a", "d", "m", "aa" denote the CPU time required for add/sub/div/mult/assig

Total CPU time is:

$$(2N-1)d + 3(N-1)m + 3(N-1)a + [4(N-1)+1]aa =$$

$$= (N-1)[2d + 3m + 3a + 4aa] + aa + d$$

$\Rightarrow$ CPU time scales like N ! Thomas algorithm has linear complexity.

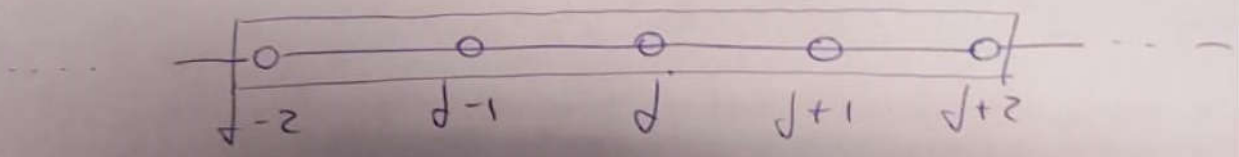
- LU prefactorization reduces the total operation count when many systems with same coeff. matrix must be solved. But... still linear complexity.

Explicit Finite Difference Schemes

Automatic computation of the coefficients of explicit finite-difference formulas of arbitrary order

S: stencil

E.g.: $S = -2 : 2$



$f_j^{(d)} \cong \frac{1}{h^d} \sum_{k \in S} a_k f_{j+k}$ finite-difference formula

Matching orders of Taylor expansion:

$$f_j^{(d)} \cong \sum_{k \in S} a_k \sum_{m=0}^{+\infty} \frac{1}{m!} f_j^{(m)} k^m h^{m-d}$$

$$= \sum_{m=0}^{\infty} \frac{1}{m!} f_j^{(m)} h^{m-d} \sum_{k \in S} k^m a_k$$

For $m \neq d$: $\sum_{k \in S} k^m a_k = 0$

For $k = d$: $\frac{1}{d!} f_j^{(d)} \cdot h^0 \sum_{k \in S} k^d a_k = f_j^{(d)}$

$$\Rightarrow \sum_{k \in S} k^d a_k = d!$$

If $\#S = S$, we have S unknown coefficients a_k so that $d < S$.

(max "d" with a trivial decomposing S points is $S-1$).

In matrix form

$$\underline{\underline{A}} \equiv (a_{ij}) \qquad a_{ij} = (s(j))^i$$

$i = 0 \ldots S-1$

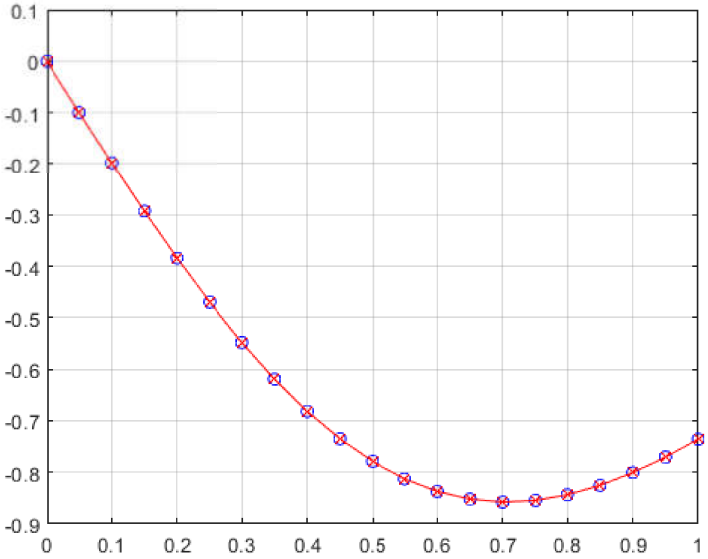
$j = 1 \ldots S$

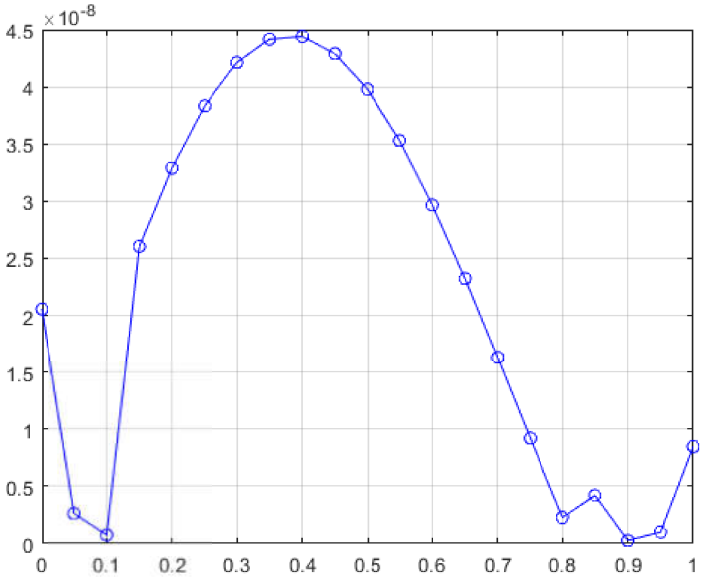
$$\underline{b} \equiv (b_i) \qquad b_i = d!\, \delta_{id}$$

```
%{
  Compare explicit finite-difference schemes for the first derivative,
  whose coefficients are computed automatically (uniform grid only).

  clear all;
  close all;
  clc;

  func    = @(x) exp(-x.^2);
  dfunc   = @(x) -2*x.*exp(-x.^2);
  h       = 0.05;
  x       = 0:h:1;
  stencil = -3:3;

  [df,err,Linf,L2]=FDschemes(func,dfunc,x,stencil);


%}
function FDschemesDriver

clear all;
close all;
clc;

func    = @(x) exp(-x.^2);
dfunc   = @(x) -2*x.*exp(-x.^2);
h       = 0.05;
x       = 0:h:1;
stencil = -3:3;

[df,err,Linf,L2]=FDschemes(func,dfunc,x,stencil);

end

function [df,err,Linf,L2]=FDschemes(func,dfunc,x,stencil)

x   = x(:);
N   = length(x);
f   = func(x);
dfa = dfunc(x);
h   = x(2)-x(1);


df          = nan(N,1);

sl   = length(stencil);
smin = min(stencil); smax = max(stencil);

for j=1:-smin
    s = -j+1:(-j+1)+sl+1;
    a = computeFDcoeffs(s,1);
    df(j) = sum(f(j+s).*a)/h;
```

```
end

a = computeFDcoeffs(stencil,1);
for j=1-smin:N-smax
    df(j) = sum(f(j+stencil).*a)/h;
end

for j=N-smax+1:N
    s = N-j-sl-1:N-j;
    a = computeFDcoeffs(s,1);
    df(j) = sum(f(j+s).*a)/h;
end




% Error
err  = abs(dfa-df);
Linf = max(err);
L2   = sqrt(sum(err.^2));

figure
plot(x,df,'bo');
hold on
plot(x,dfa,'rx-');
grid on
box on

figure
plot(x,err,'bo-');
grid on
box on

end

% Compute FD coefficients of arbitrary order on arbitrary stencil
function a = computeFDcoeffs(stencil,d)

N = length(stencil);

A = nan(N,N);
b = nan(N,1);
for i=1:N
    for j=1:N
        A(i,j) = stencil(j)^(i-1);
    end
    b(i) = dKron(i-1,d);
end
b = b.*factorial(d);
a = A\b;


end
```

```
function del=dKron(i,j)

del = double(i==j);

end
```





Published with MATLAB® R2021b

Compact Schemes

Compact Finite Difference Schemes

Per-Olof Persson
persson@berkeley.edu

Department of Mathematics
University of California, Berkeley

Math 228B Numerical Solutions of Differential Equations

Approximation of First Derivative

- Consider a uniformly spaced mesh $x_i = hi$ with given function values $f_i = f(x_i)$. Seek approximations to the first derivative at the nodes $f'_i = f'(x_i)$ of the form

$$\begin{aligned}\beta f'_{i-2} + \alpha f'_{i-1} + f'_i + \alpha f'_{i+1} + \beta f'_{i+2} \\ = c\frac{f_{i+3} - f_{i-3}}{6h} + b\frac{f_{i+2} - f_{i-2}}{4h} + a\frac{f_{i+1} - f_{i-1}}{2h}\end{aligned}$$

- Match Taylor series coefficients for order conditions:

$$a + b + c = 1 + 2\alpha + 2\beta \qquad \text{(2nd order)}$$

$$a + 2^2 b + 3^2 c = 2\frac{3!}{2!}(\alpha + 2^2\beta) \qquad \text{(4th order)}$$

$$a + 2^4 b + 3^4 c = 2\frac{5!}{4!}(\alpha + 2^4\beta) \qquad \text{(6th order)}$$

$$a + 2^6 b + 3^6 c = 2\frac{7!}{6!}(\alpha + 2^6\beta) \qquad \text{(8th order)}$$

$$a + 2^8 b + 3^8 c = 2\frac{9!}{8!}(\alpha + 2^8\beta) \qquad \text{(10th order)}$$

Tridiagonal Schemes

- If $\beta = 0$ and α nonzero, tridiagonal systems need to be solved to obtain the derivative approximations
- If in addition $c = 0$, a one-parameter (α) family of 4th order tridiagonal schemes is obtained:

$$\beta = 0, \quad a = \frac{2}{3}(\alpha + 2), \quad b = \frac{1}{3}(4\alpha - 1), \quad c = 0$$

- Special cases:
  - $\alpha = 0$ gives the standard 4th order central difference scheme,
  - $\alpha = 1/4$ gives the classical Padé scheme.
  - $\alpha = 1/3$ gives a 6th order accurate scheme:

$$\alpha = \frac{1}{3}, \quad \beta = 0, \quad a = \frac{14}{9}, \quad b = \frac{1}{9} \quad c = 0$$

Tridiagonal Schemes

- If $\beta = 0$ and $c \neq 0$, a one-parameter (α) family of 6th order tridiagonal schemes is obtained:

$$\beta = 0, \quad a = \frac{1}{6}(\alpha + 9), \quad b = \frac{1}{15}(32\alpha - 9), \quad c = \frac{1}{10}(-3\alpha + 1)$$

- Special cases:
  - $\alpha = 3/8$ gives an 8th order accurate scheme:

$$\beta = 0, \quad a = \frac{25}{16}, \quad b = \frac{1}{5}, \quad c = -\frac{1}{80}$$

Pendadiagonal Schemes

- If $\beta \neq 0$ and $c = 0$, a one-parameter (α) family of 6th order pentadiagonal schemes is obtained:

$$\beta = \frac{1}{12}(-1 + 3\alpha), \quad a = \frac{2}{9}(8 - 3\alpha), \quad b = \frac{1}{18}(-17 + 57\alpha), \quad c = 0$$

which becomes an 8th order scheme if $\alpha = 4/9$:

$$\alpha = \frac{4}{9}, \quad \beta = \frac{1}{36}, \quad a = \frac{40}{27}, \quad b = \frac{25}{54}, \quad c = 0$$

- If $\beta \neq 0$ and $c \neq 0$, a one-parameter (α) family of 8th order pentadiagonal schemes is obtained:

$$\beta = \frac{1}{20}(-3 + 8\alpha), \; a = \frac{1}{6}(12 - 7\alpha), \; b = \frac{1}{150}(568\alpha - 183), \; c = \frac{1}{50}(9\alpha - 4)$$

which becomes a 10th order scheme if $\alpha = 1/2$:

$$\alpha = \frac{1}{2}, \quad \beta = \frac{1}{20}, \quad a = \frac{17}{12}, \quad b = \frac{101}{150}, \quad c = \frac{1}{100}$$

Approximation of Second Derivative

- Seek approximations to the second derivative at the nodes $f_i'' = f''(x_i)$ of the form

$$\beta f''_{i-2} + \alpha f''_{i-1} + f''_i + \alpha f''_{i+1} + \beta f''_{i+2}$$
$$= c\frac{f_{i+3} - 2f_i + f_{i-3}}{9h^2} + b\frac{f_{i+2} - 2f_i + f_{i-2}}{4h^2} + a\frac{f_{i+1} - 2f_i + f_{i-1}}{h^2}$$

- Match Taylor series coefficients for order conditions:

$$a + b + c = 1 + 2\alpha + 2\beta \qquad \text{(2nd order)}$$
$$a + 2^2 b + 3^2 c = 2\frac{4!}{2!}(\alpha + 2^2\beta) \qquad \text{(4th order)}$$
$$a + 2^4 b + 3^4 c = 2\frac{6!}{4!}(\alpha + 2^4\beta) \qquad \text{(6th order)}$$
$$a + 2^6 b + 3^6 c = 2\frac{8!}{6!}(\alpha + 2^6\beta) \qquad \text{(8th order)}$$
$$a + 2^8 b + 3^8 c = 2\frac{10!}{8!}(\alpha + 2^8\beta) \qquad \text{(10th order)}$$

Tridiagonal Schemes

- If $\beta = 0$ and $c = 0$, a one-parameter (α) family of 4th order tridiagonal schemes is obtained:

$$\beta = 0, \quad c = 0, \quad a = \frac{4}{3}(1-\alpha), \quad b = \frac{1}{3}(-1+10\alpha)$$

- Special cases:
  - $\alpha = 0$ gives the standard 4th order central difference scheme,
  - $\alpha = 1/10$ gives the classical Padé scheme.
  - $\alpha = 2/11$ gives a 6th order accurate scheme:

$$\alpha = \frac{2}{11}, \quad \beta = 0, \quad a = \frac{12}{11}, \quad b = \frac{3}{11}, \quad c = 0$$

- Similarly to before, higher order and pentadiagonal schemes can be derived

Fourier Analysis

- Assume periodic domain $[0, L]$ with $f_1 = f_{N+1}$ and $h = L/N$
- Decompose variables into fourier coefficients

$$f(x) = \sum_{k=-N/2}^{n=N/2} \hat{f}_k \exp\left(\frac{2\pi i k x}{L}\right)$$

- Introduced scaled wavenumber $w = 2\pi kh/L = 2\pi k/N$ in the domain $[0, \pi]$, and the scaled coordinate $s = x/h$, giving Fourier modes $\exp(iws)$
- The exact first derivative generates Fourier coefficients $\hat{f}'_k = iw\hat{f}_k$
- Compare with the coefficients $(\hat{f}'_k)_{fd} = iw'\hat{f}_k$ obtained from the differencing scheme

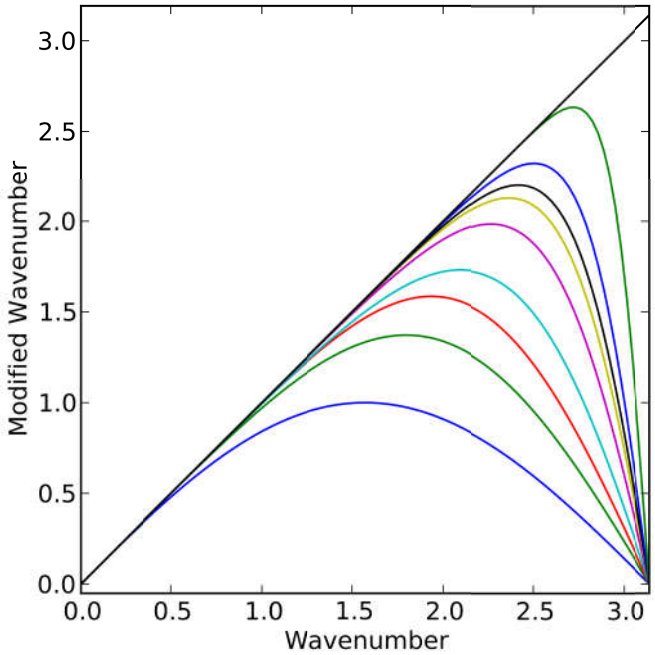
Modified wavenumber, first derivatives

- The first derivative approximations:

$$\beta f'_{i-2} + \alpha f'_{i-1} + f'_i + \alpha f'_{i+1} + \beta f'_{i+2}$$
$$= c\frac{f_{i+3} - f_{i-3}}{6h} + b\frac{f_{i+2} - f_{i-2}}{4h} + a\frac{f_{i+1} - f_{i-1}}{2h}$$

corresponds to

$$w'(w) = \frac{a\sin(w) + (b/2)\sin(2w) + (c/3)\sin(3w)}{1 + 2\alpha\cos(w) + 2\beta\cos(2w)}$$



From bottom to top: 2nd order central, 4th order central, 4th order central, standard Padé, 6th order tridiagonal, 8th order tridiagonal, 8th order pentadiagonal, 10th order pentadiagonal, spectral-like, exact differentiation

Modified wavenumber, second derivatives

- The exact second derivative generates Fourier coefficients $\hat{f}''_k = -w^2 \hat{f}_k$
- Compare with the coefficients $(\hat{f}''_k)_{fd} = -w'' \hat{f}_k$ obtained from the differencing scheme
- The second derivative approximations:

$$\beta f''_{i-2} + \alpha f''_{i-1} + f''_i + \alpha f''_{i+1} + \beta f''_{i+2}$$
$$= c\frac{f_{i+3} - 2f_i + f_{i-3}}{9h^2} + b\frac{f_{i+2} - 2f_i + f_{i-2}}{4h^2} + a\frac{f_{i+1} - 2f_i + f_{i-1}}{h^2}$$

corresponds to

$$w''(w) = \frac{2a(1-\cos(w)) + (b/2)(1-\cos(2w)) + (2c/9)(1-\cos(3w))}{1 + 2\alpha\cos(w) + 2\beta\cos(2w)}$$

Non-Periodic Boundaries

- Find first derivative at the boundary $i = 1$ by one-sided approximation:

$$f_1' + \alpha f_2' = \frac{1}{h}(af_1 + bf_2 + cf_3 + df_4)$$

- 2nd order accuracy requires

$$a = -\frac{3 + \alpha + 2d}{2}, \quad b = 2 + 3d, \quad c = -\frac{1 - \alpha + 6d}{2}$$

- Third and fourth order schemes can also be derived
- Harder to study using Fourier analysis

Filtering

- Find filtered values $\hat{f}_i$ from an approximation

$$\beta\hat{f}_{i-2} + \alpha\hat{f}_{i-1} + \hat{f}_i + \alpha\hat{f}_{i+1} + \beta\hat{f}_{i+2}$$
$$= af_i + \frac{d}{2}(f_{i+3} + f_{i-3}) + \frac{c}{2}(f_{i+2} + f_{i-2}) + \frac{b}{2}(f_{i+1} + f_{i-1})$$

  with transfer function

$$T(w) = \frac{a + b\cos(w) + c\cos(2w) + d\cos(3w)}{1 + 2\alpha\cos(w) + 2\beta\cos(2w)}$$

- Impose $T(\pi) = 0$, and possibly higher order derivatives
- Match Taylor series coefficients for high formal accuracy

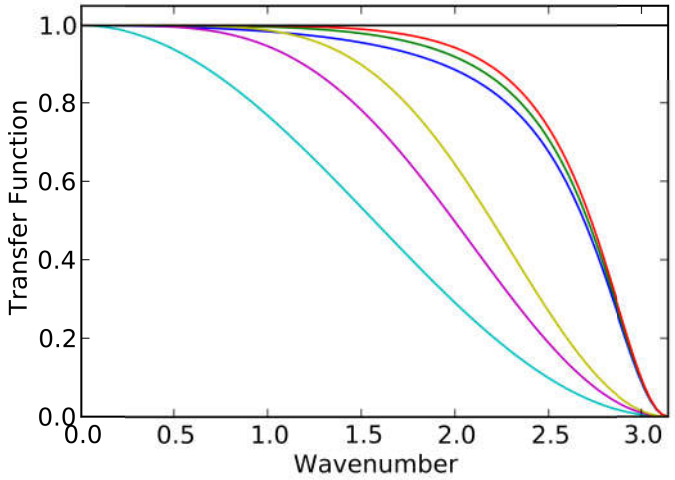
Filtering

- Set $\beta = 0$ for tridiagonal system, use one free parameter $-0.5 < \alpha \leq 0.5$:

$$(F2) \quad a = \frac{1}{2} + \alpha, \quad b = \frac{1}{2} + \alpha$$

$$(F4) \quad a = \frac{5}{8} + \frac{3\alpha}{4}, \quad b = \frac{1}{2} + \alpha \quad c = -\frac{1}{8} + \frac{\alpha}{4}$$

$$(F6) \quad a = \frac{11}{16} + \frac{5\alpha}{8}, \quad b = \frac{15}{32} + \frac{17\alpha}{16}, \quad c = -\frac{3}{16} + \frac{3\alpha}{8}, \quad d = \frac{1}{32} - \frac{\alpha}{16}$$



From left to right:
$(F2)$ $\alpha = 0$, $(F4)$ $\alpha = 0$, $(F6)$ $\alpha = 0$,
$(F2)$ $\alpha = 0.45$, $(F4)$ $\alpha = 0.45$,
$(F6)$ $\alpha = 0.45$,

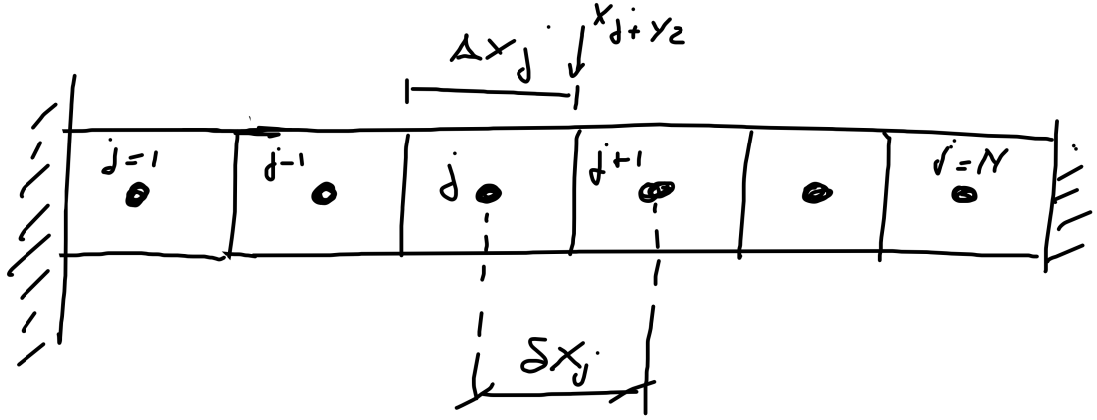
10.2.2 Finite Volume Schemes

Finite-volume method

- Integral form of conservation equations
- Inherently respects conservation of quantities at integral level:
  If there are no source terms nor net flux of the transported through the boundaries, the amount of that quantity within the computational domain does not change.

This is a physical link to reality and sometimes guarantees the boundedness of the solution. This is the case for the positive-definite quantities (e.g., kinetic energy).

- FVM on 1D, co-located, Cartesian mesh:



- Starting from the differential form of the BVP:

$$\begin{cases} \lambda \dfrac{d^2y}{dx^2} + g = 0 & \text{in } (0,L) \\ y(0) = y(L) = 0 \end{cases}$$

Integrate over C.V.s:

$$\int_{\Delta x_j} [\lambda y'' + g]\, dx = 0$$

So, condition of vanishing residual is satisfied on each C.V.

Integrate by parts $\Longrightarrow$ <u>fluxes</u> show up!

$$\lambda y' \Big|_{x_{j-1/2}}^{x_{j+1/2}} + \int_{\Delta x_j} g\, dx = 0$$

Fluxes — source

Sum over all c.v.s to get:

$$\sum_{1}^{N} \lambda y' \Big|_{x_{j-1/2}}^{x_{j+1/2}} + \sum_{1}^{N} \int_{\Delta x_j} g(x)\,dx = 0$$

$$\Downarrow$$

$$\lambda y'(L) - \lambda y'(0) + \int_{0}^{L} g(x)\,dx = 0$$

This is exactly the global conservation law!

$$y'_{j+1/2} \simeq \frac{y_{j+1} - y_j}{\delta x_j} \qquad y'_{j-1/2} \simeq \frac{y_j - y_{j-1}}{\delta x_{j-1}}$$

$$\int_{\Delta x_j} g(x)\, dx \simeq g(x_j)\, \Delta x_j$$

$$\Rightarrow \boxed{\left(\frac{\lambda}{\delta x_{j-1}}\right) y_{j-1} - \left(\frac{\alpha}{\delta x_j} + \frac{\lambda}{\delta x_{j-1}}\right) y_j + \left(\frac{\lambda}{\delta x_j}\right) y_{j+1} + g_j \Delta x_j = 0}$$

$$a W_j\, y_{j-1} - a P_j y_j + a E_j\, y_{j+1} - b_j = 0$$

Boundary conditions:

- $j=1$: unknown y_{j-1} undefined. Use boundary condition with ghost cell: $\frac{y_0 + y_1}{2} = 0 \Rightarrow y_0 = -y_1$

$$a W_1 (-y_1) + a P_1 y_1 + a E_1 y_2 = b_1$$

$$\rightarrow \left(a W_1 + a P_1\right) y_1 + a E_1 y_2 = b_1$$

Similarly for $j = N$.