Truncation Error Analysis

Hans Petter Langtangen[1,2]

[1]Center for Biomedical Computing, Simula Research Laboratory
[2]Department of Informatics, University of Oslo

Sep 29, 2013

WARNING: Preliminary version (expect typos!)

Contents

List of Exercises

Purpose.
Truncation error analysis provides a widely applicable framework for analyzing the accuracy of finite difference schemes. This type of analysis can also be used for finite element and finite volume methods if the discrete equations are written in finite difference form. The result of the analysis is an asymptotic estimate of the error in the scheme on the form Ch^r, where h is a discretization parameter (Δt, Δx, etc.), r is a number, known as the convergence rate, and C is a constant, typically dependent on the derivatives of the exact solution.

Knowing r gives understanding of the accuracy of the scheme. But maybe even more important, a powerful verification method for computer codes is to check that the empirically observed convergence rates in experiments coincide with the theoretical value of r found from truncation error analysis.

The analysis can be carried out by hand, by symbolic software, and also numerically. All three methods will be illustrated. From examining the symbolic expressions of the truncation error we can add correction terms to the differential equations in order to increase the numerical accuracy.

In general, the term truncation error refers to the discrepancy that arises from performing a finite number of steps to approximate a process with infinitely many steps. The term is used in a number of contexts, including truncation of infinite series, finite precision arithmetic, finite differences, and differential equations. We shall be concerned with computing truncation errors arising in finite difference formulas and in finite difference discretizations of differential equations.

1 Overview of truncation error analysis

1.1 Abstract problem setting

Consider an abstract differential equation

$$\mathcal{L}(u) = 0,$$

where $\mathcal{L}(u)$ is some formula involving the unknown u and its derivatives. One example is $\mathcal{L}(u) = u'(t) + a(t)u(t) - b(t)$, where a and b are contants or functions of time. We can discretize the differential equation and obtain a corresponding discrete model, here written as

$$\mathcal{L}_\Delta(u) = 0\,.$$

The solution u of this equation is the *numerical solution*. To distinguish the numerical solution from the exact solution of the differential equation problem, we denote the latter by u_{e} and write the differential equation and its discrete counterpart as

$$\mathcal{L}(u_{\mathrm{e}}) = 0,$$
$$\mathcal{L}_\Delta(u) = 0\,.$$

Initial and/or boundary conditions can usually be left out of the truncation error analysis and are omitted in the following.

The numerical solution u is in a finite difference method computed at a collection of mesh points. The discrete equations represented by the abstract equation $\mathcal{L}_\Delta(u) = 0$ are usually algebraic equations involving u at some neighboring mesh points.

1.2 Error measures

A key issue is how accurate the numerical solution is. The ultimate way of addressing this issue would be to compute the error $u_{\mathrm{e}} - u$ at the mesh points. This is usually extremely demanding. In very simplified problem settings we may, however, manage to derive formulas for the numerical solution u, and therefore closed form expressions for the error $u_{\mathrm{e}} - u$. Such special cases can provide considerable insight regarding accuracy and stability, but the results are established for special problems.

The error $u_{\mathrm{e}} - u$ can be computed empirically in special cases where we know u_{e}. Such cases can be constructed by the method of manufactured solutions, where we choose some exact solution $u_{\mathrm{e}} = v$ and fit a source term f in the governing differential equation $\mathcal{L}(u_{\mathrm{e}}) = f$ such that $u_{\mathrm{e}} = v$ is a solution (i.e., $f = \mathcal{L}(v)$). Assuming an error model of the form Ch^r, where h is the discretization parameter, such as Δt or Δx, one can estimate the convergence rate r. This is a widely applicable procedure, but the valididity of the results is, strictly speaking, tied to the chosen test problems.

Another error measure is to ask to what extent the exact solution u_{e} fits the discrete equations. Clearly, u_{e} is in general not a solution of $\mathcal{L}_\Delta(u) = 0$, but we can define the residual

$$R = \mathcal{L}_\Delta(u_{\mathrm{e}}),$$

and investigate how close R is to zero. A small R means intuitively that the discrete equations are close to the differential equation, and then we are tempted to think that u^n must also be close to $u_{\mathrm{e}}(t_n)$.

The residual R is known as the truncation error of the finite difference scheme $\mathcal{L}_\Delta(u) = 0$. It appears that the truncation error is relatively straightforward to compute by hand or symbolic software *without specializing the differential equation and the discrete model to a special case*. The resulting R is found as a power series in the discretization parameters. The leading-order terms in the series provide an asymptotic measure of the accuracy of the numerical solution method (as the discretization parameters tend to zero). An advantage of truncation error analysis compared empricial estimation of convergence rates or detailed analysis of a special problem with a mathematical expression for the numerical solution, is that the truncation error analysis reveals the accuracy

of the various building blocks in the numerical method and how each building block impacts the overall accuracy. The analysis can therefore be used to detect building blocks with lower accuracy than the others.

Knowing the truncation error or other error measures is important for verification of programs by empirically establishing convergence rates. The forthcoming text will provide many examples on how to compute truncation errors for finite difference discretizations of ODEs and PDEs.

2 Truncation errors in finite difference formulas

The accuracy of a finite difference formula is a fundamental issue when discretizing differential equations. We shall first go through a particular example in detail and thereafter list the truncation error in the most common finite difference approximation formulas.

2.1 Example: The backward difference for $u'(t)$

Consider a backward finite difference approximation of the first-order derivative u':

$$[D_t^- u]^n = \frac{u^n - u^{n-1}}{\Delta t} \approx u'(t_n)\,. \tag{1}$$

Here, u^n means the value of some function $u(t)$ at a point t_n, and $[D_t^- u]^n$ is the *discrete derivative* of $u(t)$ at $t = t_n$. The discrete derivative computed by a finite difference is not exactly equal to the derivative $u'(t_n)$. The error in the approximation is

$$R^n = [D_t^- u]^n - u'(t_n)\,. \tag{2}$$

The common way of calculating R^n is to

1. expand $u(t)$ in a Taylor series around the point where the derivative is evaluated, here t_n,
2. insert this Taylor series in (2), and
3. collect terms that cancel and simplify the expression.

The result is an expression for R^n in terms of a power series in Δt. The error R^n is commonly referred to as the *truncation error* of the finite difference formula.

The Taylor series formula often found in calculus books takes the form

$$f(x+h) = \sum_{i=0}^{\infty} \frac{1}{i!}\frac{d^i f}{dx^i}(x)h^i\,.$$

In our application, we expand the Taylor series around the point where the finite difference formula approximates the derivative. The Taylor series of u^n at t_n

is simply $u(t_n)$, while the Taylor sereis of u^{n-1} at t_n must employ the general formula,

$$u(t_{n-1}) = u(t-\Delta t) = \sum_{i=0}^{\infty} \frac{1}{i!}\frac{d^i u}{dt^i}(t_n)(-\Delta t)^i$$
$$= u(t_n) - u'(t_n)\Delta t + \frac{1}{2}u''(t_n)\Delta t^2 + \mathcal{O}(\Delta t^3),$$

where $\mathcal{O}(\Delta t^3)$ means a power-series in Δt where the lowest power is Δt^3. We assume that Δt is small such that $\Delta t^p \gg \Delta t^q$ if p is smaller than q. The details of higher-order terms in Δt are therefore not of much interest. Inserting the Taylor series above in the left-hand side of1 (2) gives rise to some algebra:

$$\begin{aligned}
[D_t^- u]^n - u'(t_n) &= \frac{u(t_n) - u(t_{n-1})}{\Delta t} - u'(t_n) \\
&= \frac{u(t_n) - (u(t_n) - u'(t_n)\Delta t + \frac{1}{2}u''(t_n)\Delta t^2 + \mathcal{O}(\Delta t^3))}{\Delta t} - u'(t_n) \\
&= -\frac{1}{2}u''(t_n)\Delta t + \mathcal{O}(\Delta t^2)),
\end{aligned}$$

which is, according to (2), the truncation error:

$$R^n = -\frac{1}{2}u''(t_n)\Delta t + \mathcal{O}(\Delta t^2))\,. \tag{3}$$

The dominating term for small Δt is $-\frac{1}{2}u''(t_n)\Delta t$, which is proportional to Δt, and we say that the truncation error is of *first order* in Δt.

2.2 Example: The forward difference for $u'(t)$

We can analyze the approximation error in the forward difference

$$u'(t_n) \approx [D_t^+ u]^n = \frac{u^{n+1} - u^n}{\Delta t},$$

by writing

$$R^n = [D_t^+ u]^n - u'(t_n),$$

and expanding u^{n+1} in a Taylor series around t_n,

$$u(t_{n+1}) = u(t_n) + u'(t_n)\Delta t + \frac{1}{2}u''(t_n)\Delta t^2 + \mathcal{O}(\Delta t^3)\,.$$

The result becomes

$$R = \frac{1}{2}u''(t_n)\Delta t + \mathcal{O}(\Delta t^2),$$

showing that also the forward difference is of first order.

2.3 Example: The central difference for $u'(t)$

For the central difference approximation,

$$u'(t_n) \approx [D_t u]^n, \quad [D_t u]^n = \frac{u^{n+\frac{1}{2}} - u^{n-\frac{1}{2}}}{\Delta t},$$

we write

$$R^n = [D_t u]^n - u'(t_n),$$

and expand $u(t_{n+\frac{1}{2}})$ and $u(t_{n-1/2})$ in Taylor series around the point t_n where the derivative is evaluated. We have

$$\begin{aligned}
u(t_{n+\frac{1}{2}}) =& u(t_n) + u'(t_n)\frac{1}{2}\Delta t + \frac{1}{2}u''(t_n)(\frac{1}{2}\Delta t)^2 + \\
& \frac{1}{6}u'''(t_n)(\frac{1}{2}\Delta t)^3 + \frac{1}{24}u''''(t_n)(\frac{1}{2}\Delta t)^4 + \\
& \frac{1}{120}u'''''(t_n)(\frac{1}{2}\Delta t)^5 + \mathcal{O}(\Delta t^6), \\
u(t_{n-1/2}) =& u(t_n) - u'(t_n)\frac{1}{2}\Delta t + \frac{1}{2}u''(t_n)(\frac{1}{2}\Delta t)^2 - \\
& \frac{1}{6}u'''(t_n)(\frac{1}{2}\Delta t)^3 + \frac{1}{24}u''''(t_n)(\frac{1}{2}\Delta t)^4 - \\
& \frac{1}{120}u'''''(t_n)(\frac{1}{2}\Delta t)^5 + \mathcal{O}(\Delta t^6)\,.
\end{aligned}$$

Now,

$$u(t_{n+\frac{1}{2}}) - u(t_{n-1/2}) = u'(t_n)\Delta t + \frac{1}{24}u'''(t_n)\Delta t^3 + \frac{1}{960}u'''''(t_n)\Delta t^5 + \mathcal{O}(\Delta t^7)\,.$$

By collecting terms in $[D_t u]^n - u(t_n)$ we find the truncation error to be

$$R^n = \frac{1}{24}u'''(t_n)\Delta t^2 + \mathcal{O}(\Delta t^4), \tag{4}$$

with only even powers of Δt. Since $R \sim \Delta t^2$ we say the centered difference is of *second order* in Δt.

2.4 Overview of leading-order error terms in finite difference formulas

Here we list the leading-order terms of the truncation errors associated with several common finite difference formulas for the first and second derivatives.

$$[D_t u]^n = \frac{u^{n+\frac{1}{2}} - u^{n-\frac{1}{2}}}{\Delta t} = u'(t_n) + R^n,$$
$$R^n = \frac{1}{24}u'''(t_n)\Delta t^2 + \mathcal{O}(\Delta t^4) \tag{5}$$
$$[D_{2t} u]^n = \frac{u^{n+1} - u^{n-1}}{2\Delta t} = u'(t_n) + R^n,$$
$$R^n = \frac{1}{6}u'''(t_n)\Delta t^2 + \mathcal{O}(\Delta t^4) \tag{6}$$
$$[D_t^- u]^n = \frac{u^n - u^{n-1}}{\Delta t} = u'(t_n) + R^n,$$
$$R^n = -\frac{1}{2}u''(t_n)\Delta t + \mathcal{O}(\Delta t^2) \tag{7}$$
$$[D_t^+ u]^n = \frac{u^{n+1} - u^n}{\Delta t} = u'(t_n) + R^n,$$
$$R^n = \frac{1}{2}u''(t_n)\Delta t + \mathcal{O}(\Delta t^2) \tag{8}$$
$$[\bar{D}_t u]^{n+\theta} = \frac{u^{n+1} - u^n}{\Delta t} = u'(t_{n+\theta}) + R^{n+\theta},$$
$$R^{n+\theta} = \frac{1}{2}(1-2\theta)u''(t_{n+\theta})\Delta t - \frac{1}{6}((1-\theta)^3 - \theta^3)u'''(t_{n+\theta})\Delta t^2 + \mathcal{O}(\Delta t^3) \tag{9}$$
$$[D_t^{2-} u]^n = \frac{3u^n - 4u^{n-1} + u^{n-2}}{2\Delta t} = u'(t_n) + R^n,$$
$$R^n = -\frac{1}{3}u'''(t_n)\Delta t^2 + \mathcal{O}(\Delta t^3) \tag{10}$$
$$[D_t D_t u]^n = \frac{u^{n+1} - 2u^n + u^{n-1}}{\Delta t^2} = u''(t_n) + R^n,$$
$$R^n = \frac{1}{12}u''''(t_n)\Delta t^2 + \mathcal{O}(\Delta t^4) \tag{11}$$

It will also be convenient to have the truncation errors for various means or averages. The weighted arithmetic mean leads to

$$[\overline{u}^{t,\theta}]^{n+\theta} = \theta u^{n+1} + (1-\theta)u^n = u(t_{n+\theta}) + R^{n+\theta},$$
$$R^{n+\theta} = \frac{1}{2}u''(t_{n+\theta})\Delta t^2\theta(1-\theta) + \mathcal{O}(\Delta t^3)\,. \tag{12}$$

The standard arithmetic mean follows from this formula when $\theta = 1/2$. Expressed at point t_n we get

$$[\overline{u}^t]^n = \frac{1}{2}(u^{n-\frac{1}{2}} + u^{n+\frac{1}{2}}) = u(t_n) + R^n,$$
$$R^n = \frac{1}{8}u''(t_n)\Delta t^2 + \frac{1}{384}u''''(t_n)\Delta t^4 + \mathcal{O}(\Delta t^6)\,. \tag{13}$$

The geometric mean also has an error $\mathcal{O}(\Delta t^2)$:

$$u^{n-\frac{1}{2}}u^{n+\frac{1}{2}} = (u^n)^2 + R^n,$$
$$R^n = -\frac{1}{4}u'(t_n)^2\Delta t^2 + \frac{1}{4}u(t_n)u''(t_n)\Delta t^2 + \mathcal{O}(\Delta t^4)\,. \tag{14}$$

The harmonic mean is also second-order accurate:

$$u^n = \frac{2}{\frac{1}{u^{n-\frac{1}{2}}} + \frac{1}{u^{n+\frac{1}{2}}}} + R^{n+\frac{1}{2}},$$
$$R^n = -\frac{u'(t_n)^2}{4u(t_n)}\Delta t^2 + \frac{1}{8}u''(t_n)\Delta t^2\,. \tag{15}$$

2.5 Software for computing truncation errors

We can use `sympy` to aid calculations with Taylor series. The derivatives can be defined as symbols, say `D3f` for the 3rd derivative of some function f. A truncated Taylor series can then be written as `f + D1f*h + D2f*h**2/2`. The following class takes some symbol `f` for the function in question and makes a list of symbols for the derivatives. The `__call__` method computes the symbolic form of the series truncated at `num_terms` terms.

```
import sympy as sm

class TaylorSeries:
    """Class for symbolic Taylor series."""
    def __init__(self, f, num_terms=4):
        self.f = f
        self.N = num_terms
        # Introduce symbols for the derivatives
        self.df = [f]
        for i in range(1, self.N+1):
            self.df.append(sm.Symbol('D%d%s' % (i, f.name)))

    def __call__(self, h):
        """Return the truncated Taylor series at x+h."""
        terms = self.f
        for i in range(1, self.N+1):
            terms += sm.Rational(1, sm.factorial(i))*self.df[i]*h**i
        return terms
```

We may, for example, use this class to compute the truncation error of the Forward Euler finite difference formula:

```
>>> from truncation_errors import TaylorSeries
>>> from sympy import *
>>> u, dt = symbols('u dt')
>>> u_Taylor = TaylorSeries(u, 4)
>>> u_Taylor(dt)
D1u*dt + D2u*dt**2/2 + D3u*dt**3/6 + D4u*dt**4/24 + u
>>> FE = (u_Taylor(dt) - u)/dt
```

```
>>> FE
(D1u*dt + D2u*dt**2/2 + D3u*dt**3/6 + D4u*dt**4/24)/dt
>>> simplify(FE)
D1u + D2u*dt/2 + D3u*dt**2/6 + D4u*dt**3/24
```

The truncation error consists of the terms after the first one (u').

The module file `trunc/truncation_errors.py`[1] contains another class `DiffOp` with symbolic expressions for most of the truncation errors listed in the previous section. For example:

```
>>> from truncation_errors import DiffOp
>>> from sympy import *
>>> u = Symbol('u')
>>> diffop = DiffOp(u, independent_variable='t')
>>> diffop['geometric_mean']
-D1u**2*dt**2/4 - D1u*D3u*dt**4/48 + D2u**2*dt**4/64 + ...
>>> diffop['Dtm']
D1u + D2u*dt/2 + D3u*dt**2/6 + D4u*dt**3/24
>>> >>> diffop.operator_names()
['geometric_mean', 'harmonic_mean', 'Dtm', 'D2t', 'DtDt',
 'weighted_arithmetic_mean', 'Dtp', 'Dt']
```

The indexing of `diffop` applies names that correspond to the operators: `Dtp` for D_t^+, `Dtm` for D_t^-, `Dt` for D_t, `D2t` for D_{2t}, `DtDt` for D_tD_t.

3 Truncation errors in exponential decay ODE

We shall now compute the truncation error of a finite difference scheme for a differential equation. Our first problem involves the following the linear ODE modeling exponential decay,

$$u'(t) = -au(t)\,. \tag{16}$$

3.1 Truncation error of the Forward Euler scheme

We begin with the Forward Euler scheme for discretizing (16):

$$[D_t^+ u = -au]^n\,. \tag{17}$$

The idea behind the truncation error computation is to insert the exact solution u_{e} of the differential equation problem (16) in the discrete equations (17) and find the residual that arises because u_{e} does not solve the discrete equations. Instead, u_{e} solves the discrete equations with a residual R^n:

$$[D_t^+ u_{\mathrm{e}} + au_{\mathrm{e}} = R]^n\,. \tag{18}$$

From (8) it follows that

$$[D_t^+ u_{\mathrm{e}}]^n = u'_{\mathrm{e}}(t_n) + \frac{1}{2}u''_{\mathrm{e}}(t_n)\Delta t + \mathcal{O}(\Delta t^2),$$

[1] `http://tinyurl.com/jvzzcfn/trunc/truncation_errors.py`

which inserted in (18) results in

$$u'_{\mathrm{e}}(t_n) + \frac{1}{2}u''_{\mathrm{e}}(t_n)\Delta t + \mathcal{O}(\Delta t^2) + au_{\mathrm{e}}(t_n) = R^n\,.$$

Now, $u'_{\mathrm{e}}(t_n) + au^n_{\mathrm{e}} = 0$ since u_{e} solves the differential equation. The remaining terms constitute the residual:

$$R^n = \frac{1}{2}u''_{\mathrm{e}}(t_n)\Delta t + \mathcal{O}(\Delta t^2)\,. \tag{19}$$

This is the truncation error R^n of the Forward Euler scheme.

Because R^n is proportional to Δt, we say that the Forward Euler scheme is of first order in Δt. However, the truncation error is just one error measure, and it is not equal to the true error $u^n_{\mathrm{e}} - u^n$. For this simple model problem we can compute a range of different error measures for the Forward Euler scheme, including the true error $u^n_{\mathrm{e}} - u^n$, and all of them have dominating terms proportional to Δt.

3.2 Truncation error of the Crank-Nicolson scheme

For the Crank-Nicolson scheme,

$$[D_t u = -au]^{n+\frac{1}{2}}, \tag{20}$$

we compute the truncation error by inserting the exact solution of the ODE and adding a residual R,

$$[D_t u_{\mathrm{e}} + a\overline{u_{\mathrm{e}}}^t = R]^{n+\frac{1}{2}}\,. \tag{21}$$

The term $[D_t u_{\mathrm{e}}]^{n+\frac{1}{2}}$ is easily computed from (5) by replacing n with $n+\frac{1}{2}$ in the formula,

$$[D_t u_{\mathrm{e}}]^{n+\frac{1}{2}} = u'(t_{n+\frac{1}{2}}) + \frac{1}{24}u'''_{\mathrm{e}}(t_{n+\frac{1}{2}})\Delta t^2 + \mathcal{O}(\Delta t^4)\,.$$

The arithmetic mean is related to $u(t_{n+\frac{1}{2}})$ by (reftrunc:avg:arith) so

$$[a\overline{u_{\mathrm{e}}}^t]^{n+\frac{1}{2}} = u(t_{n+\frac{1}{2}}) + \frac{1}{8}u''(t_n)\Delta t^2 + +\mathcal{O}(\Delta t^4)\,.$$

Inserting these expressions in (21) and observing that $u'_{\mathrm{e}}(t_{n+\frac{1}{2}}) + au_{\mathrm{e}}^{n+\frac{1}{2}} = 0$, because $u_{\mathrm{e}}(t)$ solves the ODE $u'(t) = -au(t)$ at any point t, we find that

$$R^{n+\frac{1}{2}} = \left(\frac{1}{24}u'''_{\mathrm{e}}(t_{n+\frac{1}{2}}) + \frac{1}{8}u''(t_n)\right)\Delta t^2 + \mathcal{O}(\Delta t^4) \tag{22}$$

Here, the truncation error is of second order because the leading term in R is proportional to Δt^2.

At this point it is wise to redo some of the computations above to establish the truncation error of the Backward Euler scheme, see Exercise 4.

3.3 Truncation error of the θ-rule

We may also compute the truncation error of the θ-rule,

$$[\bar{D}_t u = -a\overline{u}^{t,\theta}]^{n+\theta}\,.$$

Our computational task is to find $R^{n+\theta}$ in

$$[\bar{D}_t u_{\mathrm{e}} + a\overline{u_{\mathrm{e}}}^{t,\theta} = R]^{n+\theta}\,.$$

From (9) and (12) we get expressions for the terms with u_{e}. Using that $u'_{\mathrm{e}}(t_{n+\theta}) + au_{\mathrm{e}}(t_{n+\theta}) = 0$, we end up with

$$\begin{aligned} R^{n+\theta} =& (\frac{1}{2} - \theta)u''_{\mathrm{e}}(t_{n+\theta})\Delta t + \frac{1}{2}\theta(1-\theta)u''_{\mathrm{e}}(t_{n+\theta})\Delta t^2 + \\ & \frac{1}{2}(\theta^2 - \theta + 3)u'''_{\mathrm{e}}(t_{n+\theta})\Delta t^2 + \mathcal{O}(\Delta t^3) \end{aligned} \tag{23}$$

For $\theta = 1/2$ the first-order term vanishes and the scheme is of second order, while for $\theta \neq 1/2$ we only have a first-order scheme.

3.4 Using symbolic software

The previously mentioned `truncation_error` module can be used to automate the Taylor series expansions and the process of collecting terms. Here is an example on possible use:

```
from truncation_error import DiffOp
from sympy import *

def decay():
    u, a = symbols('u a')
    diffop = DiffOp(u, independent_variable='t',
                    num_terms_Taylor_series=3)
    D1u = diffop.D(1)   # symbol for du/dt
    ODE = D1u + a*u     # define ODE

    # Define schemes
    FE = diffop['Dtp'] + a*u
    CN = diffop['Dt' ] + a*u
    BE = diffop['Dtm'] + a*u
    theta = diffop['barDt'] + a*diffop['weighted_arithmetic_mean']
    theta = sm.simplify(sm.expand(theta))
    # Residuals (truncation errors)
    R = {'FE': FE-ODE, 'BE': BE-ODE, 'CN': CN-ODE,
         'theta': theta-ODE}
    return R
```

The returned dictionary becomes

```
    decay: {
     'BE': D2u*dt/2 + D3u*dt**2/6,
     'FE': -D2u*dt/2 + D3u*dt**2/6,
     'CN': D3u*dt**2/24,
     'theta': -D2u*a*dt**2*theta**2/2 + D2u*a*dt**2*theta/2 -
               D2u*dt*theta + D2u*dt/2 + D3u*a*dt**3*theta**3/3 -
               D3u*a*dt**3*theta**2/2 + D3u*a*dt**3*theta/6 +
               D3u*dt**2*theta**2/2 - D3u*dt**2*theta/2 + D3u*dt**2/6,
    }
```

The results are in correspondence with our hand-derived expressions.

3.5 Empirical verification of the truncation error

The task of this section is to demonstrate how we can compute the truncation error R numerically. For example, the truncation error of the Forward Euler scheme applied to the decay ODE $u' = -ua$ is

$$R^n = [D_t^+ u_{\mathrm{e}} + a u_{\mathrm{e}}]^n \,. \tag{24}$$

If we happen to know the exact solution $u_{\mathrm{e}}(t)$, we can easily evaluate R^n from the above formula.

To estimate how R varies with the discretization parameter Δt, which has been our focus in the previous mathematical derivations, we first make the assumption that $R = C\Delta t^r$ for appropriate constants C and r and small enough Δt. The rate r can be estimated from a series of experiments where Δt is varied. Suppose we have m experiments $(\Delta t_i, R_i)$, $i = 0, \ldots, m-1$. For two consecutive experiments $(\Delta t_{i-1}, R_{i-1})$ and $(\Delta t_i, R_i)$, a corresponding r_{i-1} can be estimated by

$$r_{i-1} = \frac{\ln(R_{i-1}/R_i)}{\ln(\Delta t_{i-1}/\Delta t_i)}, \tag{25}$$

for $i = 1, \ldots, m-1$. Note that the truncation error R_i varies through the mesh, so (25) is to be applied pointwise. A complicating issue is that R_i and R_{i-1} refer to different meshes. Pointwise comparisons of the truncation error at a certain point in all meshes therefore requires any computed R to be restricted to the *coarsest mesh* and that all finer meshes contain all the points in the coarsest mesh. Suppose we have N_0 intervals in the coarsest mesh. Inserting a superscript n in (25), where n counts mesh points in the coarsest mesh, $n = 0, \ldots, N_0$, leads to the formula

$$r_{i-1}^n = \frac{\ln(R_{i-1}^n/R_i^n)}{\ln(\Delta t_{i-1}/\Delta t_i)} \,. \tag{26}$$

Experiments are most conveniently defined by N_0 and a number of refinements m. Suppose each mesh have twice as many cells N_i as the previous one:

$$N_i = 2^i N_0, \quad \Delta t_i = T N_i^{-1},$$

where $[0, T]$ is the total time interval for the computations. Suppose the computed R_i values on the mesh with N_i intervals are stored in an array `R[i]` (`R` being a list of arrays, one for each mesh). Restricting this R_i function to the coarsest mesh means extracting every N_i/N_0 point and is done as follows:

```
stride = N[i]/N_0
R[i] = R[i][::stride]
```

The quantity `R[i][n]` now corresponds to R_i^n.

In addition to estimating r for the pointwise values of $R = C\Delta t^r$, we may also consider an integrated quantity on mesh i,

$$R_{I,i} = \left(\Delta t_i \sum_{n=0}^{N_i} (R_i^n)^2 \right)^{\frac{1}{2}} \approx \int_0^T R_i(t) dt \,. \tag{27}$$

The sequence $R_{I,i}$, $i = 0, \ldots, m-1$, is also expected to behave as $C\Delta t^r$, with the same r as for the pointwise quantity R, as $\Delta t \to 0$.

The function below computes the R_i and $R_{I,i}$ quantities, plots them and compares with the theoretically derived truncation error (`R_a`) if available.

```
import numpy as np
import scitools.std as plt

def estimate(truncation_error, T, N_0, m, makeplot=True):
    """
    Compute the truncation error in a problem with one independent
    variable, using m meshes, and estimate the convergence
    rate of the truncation error.

    The user-supplied function truncation_error(dt, N) computes
    the truncation error on a uniform mesh with N intervals of
    length dt::

      R, t, R_a = truncation_error(dt, N)

    where R holds the truncation error at points in the array t,
    and R_a are the corresponding theoretical truncation error
    values (None if not available).

    The truncation_error function is run on a series of meshes
    with 2**i*N_0 intervals, i=0,1,...,m-1.
    The values of R and R_a are restricted to the coarsest mesh.
    and based on these data, the convergence rate of R (pointwise)
    and time-integrated R can be estimated empirically.
    """
    N = [2**i*N_0 for i in range(m)]

    R_I = np.zeros(m) # time-integrated R values on various meshes
    R   = [None]*m    # time series of R restricted to coarsest mesh
    R_a = [None]*m    # time series of R_a restricted to coarsest mesh
    dt = np.zeros(m)
    legends_R = [];  legends_R_a = []  # all legends of curves

    for i in range(m):
        dt[i] = T/float(N[i])
        R[i], t, R_a[i] = truncation_error(dt[i], N[i])

        R_I[i] = np.sqrt(dt[i]*np.sum(R[i]**2))

        if i == 0:
            t_coarse = t           # the coarsest mesh

        stride = N[i]/N_0
        R[i] = R[i][::stride]      # restrict to coarsest mesh
        R_a[i] = R_a[i][::stride]
```

```
        if makeplot:
            plt.figure(1)
            plt.plot(t_coarse, R[i], log='y')
            legends_R.append('N=%d' % N[i])
            plt.hold('on')

            plt.figure(2)
            plt.plot(t_coarse, R_a[i] - R[i], log='y')
            plt.hold('on')
            legends_R_a.append('N=%d' % N[i])

    if makeplot:
        plt.figure(1)
        plt.xlabel('time')
        plt.ylabel('pointwise truncation error')
        plt.legend(legends_R)
        plt.savefig('R_series.png')
        plt.savefig('R_series.pdf')
        plt.figure(2)
        plt.xlabel('time')
        plt.ylabel('pointwise error in estimated truncation error')
        plt.legend(legends_R_a)
        plt.savefig('R_error.png')
        plt.savefig('R_error.pdf')

    # Convergence rates
    r_R_I = convergence_rates(dt, R_I)
    print 'R integrated in time; r:',
    print ' '.join(['%.1f' % r for r in r_R_I])
    R = np.array(R)  # two-dim. numpy array
    r_R = [convergence_rates(dt, R[:,n])[-1]
           for n in range(len(t_coarse))]
```

The first `makeplot` block demonstrates how to build up two figures in parallel, using `plt.figure(i)` to create and switch to figure number `i`. Figure numbers start at 1. A logarithmic scale is used on the y axis since we expect that R as a function of time (or mesh points) is exponential. The reason is that the theoretical estimate (19) contains u_{e}'', which for the present model goes like e^{-at}. Taking the logarithm makes a straight line.

The code follows closely the previously stated mathematical formulas, but the statements for computing the convergence rates might deserve an explanation. The generic help function `convergence_rate(h, E)` computes and returns r_{i-1}, $i = 1, \ldots, m-1$ from (26), given Δt_i in `h` and R_i^n in `E`:

```
def convergence_rates(h, E):
    from math import log
    r = [log(E[i]/E[i-1])/log(h[i]/h[i-1])
         for i in range(1, len(h))]
    return r
```

Calling `r_R_I = convergence_rates(dt, R_I)` computes the sequence of rates $r_0, r_1, \ldots, r_{m-2}$ for the model $R_I \sim \Delta t^r$, while the statements

```
R = np.array(R)  # two-dim. numpy array
r_R = [convergence_rates(dt, R[:,n])[-1]
       for n in range(len(t_coarse))]
```

compute the final rate r_{m-2} for $R^n \sim \Delta t^r$ at each mesh point t_n in the coarsest mesh. This latter computation deserves more explanation. Since `R[i][n]` holds the estimated truncation error R_i^n on mesh i, at point t_n in the coarsest mesh, `R[:,n]` picks out the sequence R_i^n for $i = 0, \ldots, m-1$. The `convergence_rate` function computes the rates at t_n, and by indexing `[-1]` on the returned array from `convergence_rate`, we pick the rate r_{m-2}, which we believe is the best estimation since it is based on the two finest meshes.

The `estimate` function is available in a module `trunc_empir.py`[2]. Let us apply this function to estimate the truncation error of the Forward Euler scheme. We need a function `decay_FE(dt, N)` that can compute (24) at the points in a mesh with time step `dt` and `N` intervals:

```
import numpy as np
import trunc_empir

def decay_FE(dt, N):
    dt = float(dt)
    t = np.linspace(0, N*dt, N+1)
    u_e = I*np.exp(-a*t)  # exact solution, I and a are global
    u = u_e  # naming convention when writing up the scheme
    R = np.zeros(N)

    for n in range(0, N):
        R[n] = (u[n+1] - u[n])/dt + a*u[n]

    # Theoretical expression for the trunction error
    R_a = 0.5*I*(-a)**2*np.exp(-a*t)*dt

    return R, t[:-1], R_a[:-1]

if __name__ == '__main__':
    I = 1; a = 2  # global variables needed in decay_FE
    trunc_empir.estimate(decay_FE, T=2.5, N_0=6, m=4, makeplot=True)
```

The estimated rates for the integrated truncation error R_I become 1.1, 1.0, and 1.0 for this sequence of four meshes. All the rates for R^n, computed as `r_R`, are also very close to 1 at all mesh points. The agreement between the theoretical formula (19) and the computed quantity (ref(24)) is very good, as illustrated in Figures 1 and 2. The program `trunc_decay_FE.py`[3] was used to perform the simulations and it can easily be modified to test other schemes (see also Exericse 5).

3.6 Increasing the accuracy by adding correction terms

Now we ask the question: can we add terms in the differential equation that can help increase the order of the truncation error? To be precise, let us revisit

[2] `http://tinyurl.com/jvzzcfn/trunc/trunc_empir.py`
[3] `http://tinyurl.com/jvzzcfn/trunc/trunc_decay_FE.py`

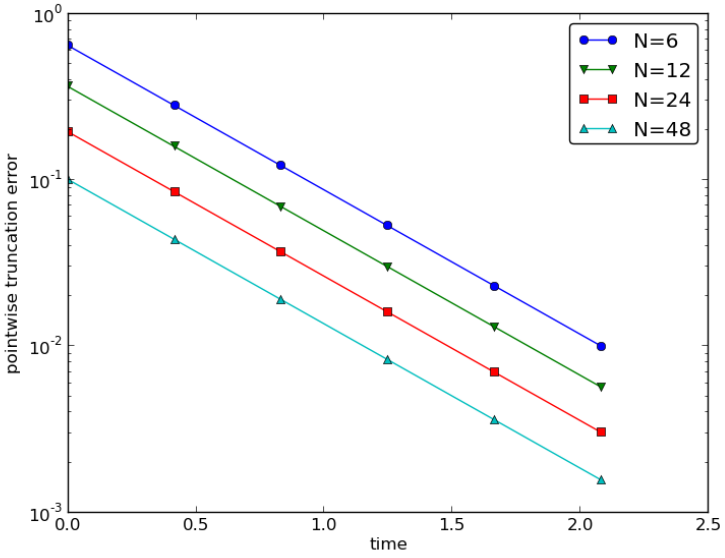

Figure 1: Estimated truncation error at mesh points for different meshes.

the Forward Euler scheme for $u' = -au$, insert the exact solution u_{e}, include a residual R, but also include new terms C:

$$[D_t^+ u_{\mathrm{e}} + a u_{\mathrm{e}} = C + R]^n \,. \tag{28}$$

Inserting the Taylor expansions for $[D_t^+ u_{\mathrm{e}}]^n$ and keeping terms up to 3rd order in Δt gives the equation

$$\frac{1}{2}u''_{\mathrm{e}}(t_n)\Delta t - \frac{1}{6}u'''_{\mathrm{e}}(t_n)\Delta t^2 + \frac{1}{24}u''''_{\mathrm{e}}(t_n)\Delta t^3 + \mathcal{O}(\Delta t^4) = C^n + R^n \,.$$

Can we find C^n such that R^n is $\mathcal{O}(\Delta t^2)$? Yes, by setting

$$C^n = \frac{1}{2}u''_{\mathrm{e}}(t_n)\Delta t,$$

we manage to cancel the first-order term and

$$R^n = \frac{1}{6}u'''_{\mathrm{e}}(t_n)\Delta t^2 + \mathcal{O}(\Delta t^3) \,.$$

The correction term C^n introduces $\frac{1}{2}\Delta t u''$ in the discrete equation, and we have to get rid of the derivative u''. One idea is to approximate u'' by a second-order accurate finite difference formula, $u'' \approx (u^{n+1} - 2u^n + u^{n-1})/\Delta t^2$, but this introduces an additional time level with u^{n-1}. Another approach is to rewrite u'' in terms of u' or u using the ODE:

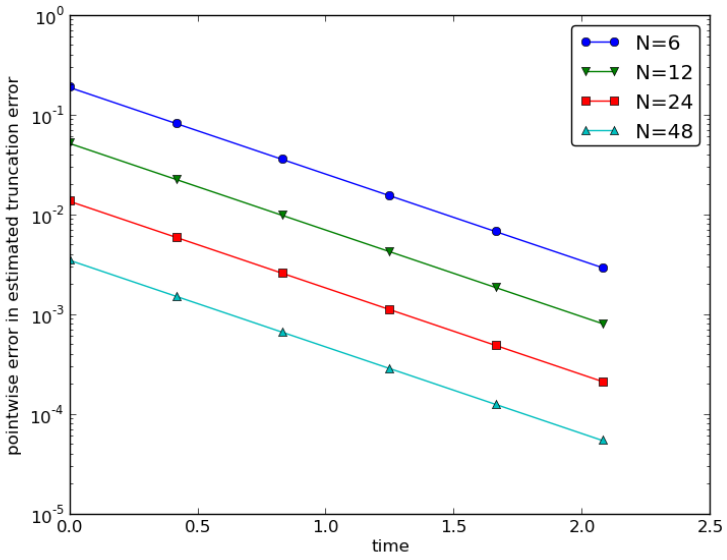

Figure 2: Difference between theoretical and estimated truncation error at mesh points for different meshes.

$$u' = -au \quad \Rightarrow \quad u'' = -au' = -a(-au) = a^2 u\,.$$

This means that we can simply set $C^n = \frac{1}{2}a^2\Delta t u^n$. We can then either solve the discrete equation

$$[D_t^+ u = -au + \frac{1}{2}a^2\Delta t u]^n, \tag{29}$$

or we can equivalently discretize the perturbed ODE

$$u' = -\hat{a}u, \quad \hat{a} = a(1 - \frac{1}{2}a\Delta t), \tag{30}$$

by a Forward Euler method. That is, we replace the original coefficient a by the perturbed coefficient $\hat{a}$. Observe that $\hat{a} \to a$ as $\Delta t \to 0$.

The Forward Euler method applied to (30) results in

$$[D_t^+ u = -a(1 - \frac{1}{2}a\Delta t)u]^n\,.$$

We can control our computations and verify that the truncation error of the scheme above is indeed $\mathcal{O}(\Delta t^2)$.

Another way of revealing the fact that the perturbed ODE leads to a more accurate solution is to look at the amplification factor. Our scheme can be written as

$$u^{n+1} = Au^n, \quad A = 1 - \hat{a}\Delta t = 1 - p + \frac{1}{2}p^2, \quad p = a\Delta t,$$

The amplification factor A as a function of $p = a\Delta t$ is seen to be the first three terms of the Taylor series for the exact amplification factor e^{-p}. The Forward Euler scheme for $u = -au$ gives only the first two terms $1 - p$ of the Taylor series for e^{-p}. That is, using $\hat{a}$ increases the order of the accuracy in the amplification factor.

Instead of replacing u'' by a^2u, we use the relation $u'' = -au'$ and add a term $-\frac{1}{2}a\Delta t u'$ in the ODE:

$$u' = -au - \frac{1}{2}a\Delta t u' \quad \Rightarrow \quad \left(1 + \frac{1}{2}a\Delta t\right) u' = -au\,.$$

Using a Forward Euler method results in

$$\left(1 + \frac{1}{2}a\Delta t\right) \frac{u^{n+1} - u^n}{\Delta t} = -au^n,$$

which after some algebra can be written as

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

This is the same formula as the one arising from a Crank-Nicolson scheme applied to $u' = -au$! It now recommended to do Exercise 6 and repeat the above steps to see what kind of correction term is needed in the Backward Euler scheme to make it second order.

The Crank-Nicolson scheme is a bit more challenging to analyze, but the ideas and techniques are the same. The discrete equation reads

$$[D_t u = -au]^{n+\frac{1}{2}},$$

and the truncation error is defined through

$$[D_t u_{\mathrm{e}} + a\overline{u_{\mathrm{e}}}^t = C + R]^{n+\frac{1}{2}},$$

where we have added a correction term. We need to Taylor expand both the discrete derivative and the arithmetic mean with aid of (5) and (13), respectively. The result is

$$\frac{1}{24}u'''_{\mathrm{e}}(t_{n+\frac{1}{2}})\Delta t^2 + \mathcal{O}(\Delta t^4) + \frac{a}{8}u''_{\mathrm{e}}(t_{n+\frac{1}{2}})\Delta t^2 + \mathcal{O}(\Delta t^4) = C^{n+\frac{1}{2}} + R^{n+\frac{1}{2}}\,.$$

The goal now is to make $C^{n+\frac{1}{2}}$ cancel the Δt^2 terms:

$$C^{n+\frac{1}{2}} = \frac{1}{24}u'''_{\mathrm{e}}(t_{n+\frac{1}{2}})\Delta t^2 + \frac{a}{8}u''_{\mathrm{e}}(t_n)\Delta t^2\,.$$

Using $u' = -au$, we have that $u'' = a^2u$, and we find that $u''' = -a^3u$. We can therefore solve the perturbed ODE problem

$$u' = -\hat{a}u, \quad \hat{a} = a(1 - \frac{1}{12}a^2\Delta t^2),$$

by the Crank-Nicolson scheme and obtain a method that is of fourth order in Δt. Exercise 7 encourages you to implement these correction terms and calculate empirical convergence rates to verify that higher-order accuracy is indeed obtained in real computations.

3.7 Extension to variable coefficients

Let us address the decay ODE with variable coefficients,

$$u'(t) = -a(t)u(t) + b(t),$$

discretized by the Forward Euler scheme,

$$[D_t^+ u = -au + b]^n \,. \tag{31}$$

The truncation error R is as always found by inserting the exact solution $u_{\mathrm{e}}(t)$ in the discrete scheme:

$$[D_t^+ u_{\mathrm{e}} + au_{\mathrm{e}} - b = R]^n \,. \tag{32}$$

Using (8),

$$u_{\mathrm{e}}'(t_n) - \frac{1}{2}u_{\mathrm{e}}''(t_n)\Delta t + \mathcal{O}(\Delta t^2) + a(t_n)u_{\mathrm{e}}(t_n) - b(t_n) = R^n \,.$$

Because of the ODE,

$$u_{\mathrm{e}}'(t_n) + a(t_n)u_{\mathrm{e}}(t_n) - b(t_n) = 0,$$

so we are left with the result

$$R^n = -\frac{1}{2}u_{\mathrm{e}}''(t_n)\Delta t + \mathcal{O}(\Delta t^2) \,. \tag{33}$$

We see that the variable coefficients do not pose any additional difficulties in this case. Exercise 8 takes the analysis above one step further to the Crank-Nicolson scheme.

3.8 Exact solutions of the finite difference equations

Having a mathematical expression for the numerical solution is very valuable in program verification since we then know the exact numbers that the program should produce. Looking at the various formulas for the truncation errors in (5)-(15) in Section 2.4, we see that all but two of the R expressions contains a second or higher order derivative of u_{e}. The exceptions are the geometric and harmonic means where the truncation error involves u_{e}' and even u_{e} in case of the harmonic mean. So, apart from these two means, choosing u_{e} to be a linear

function of t, $u_{\mathrm{e}} = ct + d$ for constants c and d, will make the truncation error vanish since $u''_{\mathrm{e}} = 0$. Consqeuently, the truncation error of a finite difference scheme will be zero since the various approximations used will all be exact. This means that the linear solution is an exact solution of the discrete equations.

In a particular differential equation problem, the reasoning above can be used to determine if we expect a linear u_{e} to fulfill the discrete equations. To actually prove that this is true, we can either compute the truncation error and see that it vanishes, or we can simply insert $u_{\mathrm{e}}(t) = ct + d$ in the scheme and see that it fulfills the equations. The latter method is usually the simplest. It will often be necessary to add some source term to the ODE in order to allow a linear solution.

Many ODEs are discretized by centered differences. From Section 2.4 we see that all the centered difference formulas have truncation errors involving u'''_{e} or higher-order derivatives. A quadratic solution, e.g., $u_{\mathrm{e}}(t) = t^2 + ct + d$, will then make the truncation errors vanish. This observation can be used to test if a quadratic solution will fulfill the discrete equations. Note that a quadratic solution will not obey the equations for a Crank-Nicolson scheme for $u' = -au + b$ because the approximation applies an arithmetic mean, which involves a truncation error with u''_{e}.

3.9 Computing truncation errors in nonlinear problems

The general nonlinear ODE

$$u' = f(u, t), \tag{34}$$

can be solved by a Crank-Nicolson scheme

$$[D_t u' = \overline{f}^t]^{n+\frac{1}{2}}\,. \tag{35}$$

The truncation error is as always defined as the residual arising when inserting the exact solution u_{e} in the scheme:

$$[D_t u'_{\mathrm{e}} - \overline{f}^t = R]^{n+\frac{1}{2}}\,. \tag{36}$$

Using (13) for $\overline{f}^t$ results in

$$\begin{aligned}[\overline{f}^t]^{n+\frac{1}{2}} &= \frac{1}{2}(f(u_{\mathrm{e}}^n, t_n) + f(u_{\mathrm{e}}^{n+1}, t_{n+1})) \\ &= f(u_{\mathrm{e}}^{n+\frac{1}{2}}, t_{n+\frac{1}{2}}) + \frac{1}{8}u''_{\mathrm{e}}(t_{n+\frac{1}{2}})\Delta t^2 + \mathcal{O}(\Delta t^4)\,.\end{aligned}$$

With (5) the discrete equations (36) lead to

$$u'_{\mathrm{e}}(t_{n+\frac{1}{2}}) + \frac{1}{24}u'''_{\mathrm{e}}(t_{n+\frac{1}{2}})\Delta t^2 - f(u_{\mathrm{e}}^{n+\frac{1}{2}}, t_{n+\frac{1}{2}}) - \frac{1}{8}u''(t_{n+\frac{1}{2}})\Delta t^2 + \mathcal{O}(\Delta t^4) = R^{n+\frac{1}{2}}\,.$$

Since $u'_{\mathrm{e}}(t_{n+\frac{1}{2}}) - f(u_{\mathrm{e}}^{n+\frac{1}{2}}, t_{n+\frac{1}{2}}) = 0$, the truncation error becomes

$$R^{n+\frac{1}{2}} = (\frac{1}{24}u_{\mathrm{e}}'''(t_{n+\frac{1}{2}}) - \frac{1}{8}u''(t_{n+\frac{1}{2}}))\Delta t^2\,.$$

The computational techniques worked well even for this nonlinear ODE.

4 Truncation errors in vibration ODEs

4.1 Linear model without damping

The next example on computing the truncation error involves the following ODE for vibration problems:

$$u''(t) + \omega^2 u(t) = 0\,. \tag{37}$$

Here, ω is a given constant.

The truncation error of a centered finite difference scheme. Using a standard, second-ordered, central difference for the second-order derivative time, we have the scheme

$$[D_t D_t u + \omega^2 u = 0]^n\,. \tag{38}$$

Inserting the exact solution u_{e} in this equation and adding a residual R so that u_{e} can fulfill the equation results in

$$[D_t D_t u_{\mathrm{e}} + \omega^2 u_{\mathrm{e}} = R]^n\,. \tag{39}$$

To calculate the truncation error R^n, we use (11), i.e.,

$$[D_t D_t u_{\mathrm{e}}]^n = u_{\mathrm{e}}''(t_n) + \frac{1}{12}u_{\mathrm{e}}''''(t_n)\Delta t^2,$$

and the fact that $u_{\mathrm{e}}''(t) + \omega^2 u_{\mathrm{e}}(t) = 0$. The result is

$$R^n = \frac{1}{12}u_{\mathrm{e}}''''(t_n)\Delta t^2 + \mathcal{O}(\Delta t^4)\,. \tag{40}$$

The truncation error of approximating $u'(0)$. The initial conditions for (37) are $u(0) = I$ and $u'(0) = V$. The latter involves a finite difference approximation. The standard choice

$$[D_{2t}u = V]^0,$$

where u^{-1} is eliminated with the aid of the discretized ODE for $n = 0$, involves a centered difference with an $\mathcal{O}(\Delta t^2)$ truncation error given by (6). The simpler choice

$$[D_t^+ u = V]^0,$$

is based on a forward difference with a truncation error $\mathcal{O}(\Delta t)$. A central question is if this initial error will impact the order of the scheme throughout the simulation. Exercise 11 asks you to quickly perform an experiment to investigate this question.

Computing correction terms. The idea of using correction terms to increase the order of R^n can be applied as described in Section 3.6. We look at

$$[D_t D_t u_{\mathrm{e}} + \omega^2 u_{\mathrm{e}} = C + R]^n,$$

and observe that C^n must be chosen to cancel the Δt^2 term in R^n. That is,

$$C^n = \frac{1}{12} u_{\mathrm{e}}''''(t_n)\Delta t^2 \,.$$

To get rid of the 4th-order derivative we can use the differential equation: $u'' = -\omega^u$, which implies $u'''' = \omega^4 u$. Adding the correction term to the ODE results in

$$u'' + \omega^2(1 - \frac{1}{12}\omega^2\Delta t^2)u = 0\,. \tag{41}$$

Solving this equation by the standard scheme

$$[D_t D_t u + \omega^2(1 - \frac{1}{12}\omega^2\Delta t^2)u = 0]^n,$$

will result in a scheme with trunction error $\mathcal{O}(\Delta t^4)$.

We can use another set of arguments to justify that (41) leads to a higher-order method. Mathematical analysis of the scheme (38) reveals that the numerical frequency $\tilde{\omega}$ is (approximately as $\Delta t \to 0$)

$$\tilde{\omega} = \omega(1 + \frac{1}{24}\omega^2\Delta t^2)\,.$$

One can therefore attempt to replace ω in the ODE by a slightly smaller ω since the numerics will make it larger:

$$[u'' + (\omega(1 - \frac{1}{24}\omega^2\Delta t^2))^2 u = 0\,.$$

Expanding the squared term and omitting the higher-order term Δt^4 gives exactly the ODE (41). Experiments show that u^n is computed to 4th order in Δt.

4.2 Model with damping and nonlinearity

The model (37) can be extended to include damping $\beta u'$, a nonlinear restoring (spring) force $s(u)$, and some known excitation force $F(t)$:

$$mu'' + \beta u' + s(u) = F(t)\,. \tag{42}$$

The coefficient m usually represents the mass of the system. This governing equation can by discretized by centered differences:

$$[mD_t D_t u + \beta D_{2t} u + s(u) = F]^n\,. \tag{43}$$

The exact solution u_{e} fulfills the discrete equations with a residual term:

$$[mD_tD_tu_\mathrm{e} + \beta D_{2t}u_\mathrm{e} + s(u_\mathrm{e}) = F + R]^n\,. \tag{44}$$

Using (11) and (6) we get

$$[mD_tD_tu_\mathrm{e} + \beta D_{2t}u_\mathrm{e}]^n = mu_\mathrm{e}''(t_n) + \beta u_\mathrm{e}'(t_n) + \left(\frac{m}{12}u_\mathrm{e}''''(t_n) + \frac{\beta}{6}u_\mathrm{e}'''(t_n)\right)\Delta t^2 + \mathcal{O}(\Delta t^4)$$

Combining this with the previous equation, we can collect the terms

$$mu_\mathrm{e}''(t_n) + \beta u_\mathrm{e}'(t_n) + \omega^2 u_\mathrm{e}(t_n) + s(u_\mathrm{e}(t_n)) - F^n,$$

and set this sum to zero because u_e solves the differential equation. We are left with the truncation error

$$R^n = \left(\frac{m}{12}u_\mathrm{e}''''(t_n) + \frac{\beta}{6}u_\mathrm{e}'''(t_n)\right)\Delta t^2 + \mathcal{O}(\Delta t^4), \tag{45}$$

so the scheme is of second order.

According to (45), we can add correction terms

$$C^n = \left(\frac{m}{12}u_\mathrm{e}''''(t_n) + \frac{\beta}{6}u_\mathrm{e}'''(t_n)\right)\Delta t^2,$$

to the right-hand side of the ODE to obtain a fourth-order scheme. However, expressing u'''' and u''' in terms of lower-order derivatives is now harder because the differential equation is more complicated:

$$\begin{aligned}
u''' &= \frac{1}{m}(F' - \beta u'' - s'(u)u'),\\
u'''' &= \frac{1}{m}(F'' - \beta u''' - s''(u)(u')^2 - s'(u)u''),\\
&= \frac{1}{m}(F'' - \beta\frac{1}{m}(F' - \beta u'' - s'(u)u') - s''(u)(u')^2 - s'(u)u'')\,.
\end{aligned}$$

It is not impossible to discretize the resulting modified ODE, but it is up to debate whether correction terms are feasible and the way to go. Computing with a smaller Δt is usually always possible in these problems to achieve the desired accuracy.

4.3 Extension to quadratic damping

Instead of the linear damping term $\beta u'$ in (42) we now consider quadratic damping $\beta|u'|u'$:

$$mu'' + \beta|u'|u' + s(u) = F(t)\,. \tag{46}$$

A centered difference for u' gives rise to a nonlinearity, which can be linearized using a geometric mean: $[|u'|u']^n \approx |[u']^{n-\frac{1}{2}}|[u']^{n+\frac{1}{2}}$. The resulting scheme becomes

$$[mD_tD_tu]^n + \beta|[D_tu]^{n-\frac{1}{2}}|[D_tu]^{n+\frac{1}{2}} + s(u^n) = F^n\,. \tag{47}$$

The truncation error is defined through

$$[mD_tD_tu_{\mathrm{e}}]^n + \beta|[D_tu_{\mathrm{e}}]^{n-\frac{1}{2}}|[D_tu_{\mathrm{e}}]^{n+\frac{1}{2}} + s(u_{\mathrm{e}}^n) - F^n = R^n\,. \tag{48}$$

We start with expressing the truncation error of the geometric mean. According to (14),

$$|[D_tu_{\mathrm{e}}]^{n-\frac{1}{2}}|[D_tu_{\mathrm{e}}]^{n+\frac{1}{2}} = [|D_tu_{\mathrm{e}}|D_tu_{\mathrm{e}}]^n - \frac{1}{4}u'(t_n)^2\Delta t^2 + \frac{1}{4}u(t_n)u''(t_n)\Delta t^2 + \mathcal{O}(\Delta t^4)\,.$$

Using (5) for the D_tu_{e} factors results in

$$[|D_tu_{\mathrm{e}}|D_tu_{\mathrm{e}}]^n = |u'_{\mathrm{e}} + \frac{1}{24}u'''_{\mathrm{e}}(t_n)\Delta t^2 + \mathcal{O}(\Delta t^4)|(u'_{\mathrm{e}} + \frac{1}{24}u'''_{\mathrm{e}}(t_n)\Delta t^2 + \mathcal{O}(\Delta t^4))$$

We can remove the absolute value since it essentially gives a factor 1 or -1 only. Calculating the product, we have the leading-order terms

$$[D_tu_{\mathrm{e}}D_tu_{\mathrm{e}}]^n = (u'_{\mathrm{e}}(t_n))^2 + \frac{1}{12}u_{\mathrm{e}}(t_n)u'''_{\mathrm{e}}(t_n)\Delta t^2 + \mathcal{O}(\Delta t^4)\,.$$

With

$$m[D_tD_tu_{\mathrm{e}}]^n = mu''_{\mathrm{e}}(t_n) + \frac{m}{12}u''''_{\mathrm{e}}(t_n)\Delta t^2 + \mathcal{O}(\Delta t^4),$$

and using the differential equation on the form $mu'' + \beta(u')^2 + s(u) = F$, we end up with

$$R^n = (\frac{m}{12}u''''_{\mathrm{e}}(t_n) + \frac{\beta}{12}u_{\mathrm{e}}(t_n)u'''_{\mathrm{e}}(t_n))\Delta t^2 + \mathcal{O}(\Delta t^4)\,.$$

This result demonstrates that we have second-order accuracy also with quadratic damping. The key elements that lead to the second-order accuracy is that the difference approximations are $\mathcal{O}(\Delta t^2)$ *and* the geometric mean approximation is also of $\mathcal{O}(\Delta t^2)$.

4.4 The general model formulated as first-order ODEs

The second-order model (46) can be formulated as a first-order system,

$$u' = v, \tag{49}$$

$$v' = \frac{1}{m}\left(F(t) - \beta|v|v - s(u)\right)\,. \tag{50}$$

The system (49)-(49) can be solved either by a forward-backward scheme or a centered scheme on a staggered mesh.

The forward-backward scheme. The discretization is based on the idea of stepping (49) forward in time and then using a backward difference in (50) with the recently computed (and therefore known) u:

$$[D_t^+ u = v]^n, \tag{51}$$

$$[D_t^- v = \frac{1}{m}(F(t) - \beta|v|v - s(u))]^{n+1}\,. \tag{52}$$

The term $|v|v$ gives rise to a nonlinearity $|v^{n+1}|v^{n+1}$, which can be linearized as $|v^n|v^{n+1}$:

$$[D_t^+ u = v]^n, \tag{53}$$

$$[D_t^- v]^{n+1} = \frac{1}{m}(F(t_{n+1}) - \beta|v^n|v^{n+1} - s(u^{n+1}))\,. \tag{54}$$

Each ODE will have a truncation error when inserting the exact solutions u_e and v_e in (51)-(52):

$$[D_t^+ u_\mathrm{e} = v_\mathrm{e} + R_u]^n, \tag{55}$$

$$[D_t^- v_\mathrm{e}]^{n+1} = \frac{1}{m}(F(t_{n+1}) - \beta|v_\mathrm{e}(t_n)|v_\mathrm{e}(t_{n+1}) - s(u_\mathrm{e}(t_{n+1}))) + R_v^{n+1}\,. \tag{56}$$

Application of (8) and (7) in (55) and (56), respectively, gives

$$u_\mathrm{e}'(t_n) + \frac{1}{2}u_\mathrm{e}''(t_n)\Delta t + \mathcal{O}(\Delta t^2) = v_\mathrm{e}(t_n) + R_u^n, \tag{57}$$

$$v_\mathrm{e}'(t_{n+1}) - \frac{1}{2}v_\mathrm{e}''(t_{n+1})\Delta t + \mathcal{O}(\Delta t^2) = \frac{1}{m}(F(t_{n+1}) - \beta|v_\mathrm{e}(t_n)|v_\mathrm{e}(t_{n+1}) + s(u_\mathrm{e}(t_{n+1})) + R_v^n\,. \tag{58}$$

Since $u_\mathrm{e}' = v_\mathrm{e}$, (57) gives

$$R_u^n = \frac{1}{2}u_\mathrm{e}''(t_n)\Delta t + \mathcal{O}(\Delta t^2)\,.$$

In (58) we can collect the terms that constitute the ODE, but the damping term has the wrong form. Let us drop the absolute value in the damping term for simplicity. Adding a substracting the right form $v^{n+1}v^{n+1}$ helps:

$$v_\mathrm{e}'(t_{n+1}) - \frac{1}{m}(F(t_{n+1}) - \beta v_\mathrm{e}(t_{n+1})v_\mathrm{e}(t_{n+1}) + s(u_\mathrm{e}(t_{n+1})) + (\beta v_\mathrm{e}(t_n)v_\mathrm{e}(t_{n+1}) - \beta v_\mathrm{e}(t_{n+1})v_\mathrm{e}(t_{n+1}))),$$

which reduces to

$$\frac{\beta}{m}v_{\mathrm{e}}(t_{n+1}(v_{\mathrm{e}}(t_n)-v_{\mathrm{e}}(t_{n+1})) = \frac{\beta}{m}v_{\mathrm{e}}(t_{n+1}[D_t^- v_{\mathrm{e}}]^{n+1}\Delta t$$
$$= \frac{\beta}{m}v_{\mathrm{e}}(t_{n+1}(v_{\mathrm{e}}'(t_{n+1})\Delta t + -\frac{1}{2}v_{\mathrm{e}}'''(t_{n+1})\Delta t^{+}\mathcal{O}(\Delta t^3))\,.$$

We end with R_u^n and R_v^{n+1} as $\mathcal{O}(\Delta t)$, simply because all the building blocks in the schemes (the forward and backward differences and the linearization trick) are only first-order accurate. However, this analysis is misleading: the building blocks play together in a way that makes the scheme second-order accurate. This is shown by considering an alternative, yet equivalent, formulation of the above scheme.

A centered scheme on a staggered mesh. We now introduce a staggered mesh where we seek u at mesh points t_n and v at points $t_{n+\frac{1}{2}}$ in between the u points. The staggered mesh makes it easy to formulate centered differences in the system (49)-(49):

$$[D_t u = v]^{n-\frac{1}{2}}, \tag{59}$$
$$[D_t v = \frac{1}{m}(F(t) - \beta|v|v - s(u))]^n\,. \tag{60}$$

The term $|v^n|v^n$ causes trouble since v^n is not computed, only $v^{n-\frac{1}{2}}$ and $v^{n+\frac{1}{2}}$. Using geometric mean, we can express $|v^n|v^n$ in terms of known quantities: $|v^n|v^n \approx |v^{n-\frac{1}{2}}|v^{n+\frac{1}{2}}$. We then have

$$[D_t u]^{n-\frac{1}{2}} = v^{n-\frac{1}{2}}, \tag{61}$$
$$[D_t v]^n = \frac{1}{m}(F(t_n) - \beta|v^{n-\frac{1}{2}}|v^{n+\frac{1}{2}} - s(u^n))\,. \tag{62}$$

The truncation error in each equation fulfills

$$[D_t u_{\mathrm{e}}]^{n-\frac{1}{2}} = v_{\mathrm{e}}(t_{n-\frac{1}{2}}) + R_u^{n-\frac{1}{2}},$$
$$[D_t v_{\mathrm{e}}]^n = \frac{1}{m}(F(t_n) - \beta|v_{\mathrm{e}}(t_{n-\frac{1}{2}})|v_{\mathrm{e}}(t_{n+\frac{1}{2}}) - s(u^n)) + R_v^n\,.$$

The truncation error of the centered differences is given by (5), and the geometric mean approximation analysis can be taken from (14). These results lead to

$$u_{\mathrm{e}}'(t_{n-\frac{1}{2}}) + \frac{1}{24}u_{\mathrm{e}}'''(t_{n-\frac{1}{2}})\Delta t^2 + \mathcal{O}(\Delta t^4) = v_{\mathrm{e}}(t_{n-\frac{1}{2}}) + R_u^{n-\frac{1}{2}},$$

and

$$v_{\mathrm{e}}'(t_n) = \frac{1}{m}(F(t_n) - \beta|v_{\mathrm{e}}(t_n)|v_{\mathrm{e}}(t_n) + \mathcal{O}(\Delta t^2) - s(u^n)) + R_v^n\,.$$

The ODEs fulfilled by u_{e} and v_{e} are evident in these equations, and we achieve second-order accuracy for the truncation error in both equations:

$$R_u^{n-\frac{1}{2}} = \mathcal{O}(\Delta t^2), \quad R_v^n = \mathcal{O}(\Delta t^2)\,.$$

Comparing (61)-(62) with (53)-(54), we can hopefully realize that these schemes are equivalent (which becomes clear when we implement both). The obvious advantage with the staggered mesh approach is that we can all the way use second-order accurate building blocks and in this way concince ourselves that the resulting scheme has an error of $\mathcal{O}(\Delta t^2)$.

5 Truncation errors in wave equations

5.1 Linear wave equation in 1D

The standard, linear wave equation in 1D for a function $u(x,t)$ reads

$$\frac{\partial^2 u}{\partial t^2} = c^2\frac{\partial^2 u}{\partial x^2} + f(x,t), \quad x\in(0,L),\ t\in(0,T], \tag{63}$$

where c is the constant wave velocity of the physical medium $[0,L]$. The equation can also be more compactly written as

$$u_{tt} = c^2 u_{xx} + f, \quad x\in(0,L),\ t\in(0,T], \tag{64}$$

Centered, second-order finite differences are a natural choice for discretizing the derivatives, leading to

$$[D_tD_t u = c^2 D_xD_x u + f]_i^n\,. \tag{65}$$

Inserting the exact solution $u_{\mathrm{e}}(x,t)$ in (65) makes this function fulfill the equation if we add the term R:

$$[D_tD_t u_{\mathrm{e}} = c^2 D_xD_x u_{\mathrm{e}} + f + R]_i^n \tag{66}$$

Our purpose is to calculate the truncation error R. From (11) we have that

$$[D_tD_t u_{\mathrm{e}}]_i^n = u_{\mathrm{e},tt}(x_i,t_n) + \frac{1}{12}u_{\mathrm{e},tttt}(x_i,t_n)\Delta t^2 + \mathcal{O}(\Delta t^4),$$

when we use a notation taking into account that u_{e} is a function of two variables and that derivatives must be partial derivatives. The notation $u_{\mathrm{e},tt}$ means $\partial^2 u_{\mathrm{e}}/\partial t^2$.

The same formula may also be applied to the x-derivative term:

$$[D_xD_x u_{\mathrm{e}}]_i^n = u_{\mathrm{e},xx}(x_i,t_n) + \frac{1}{12}u_{\mathrm{e},xxxx}(x_i,t_n)\Delta x^2 + \mathcal{O}(\Delta x^4),$$

Equation (68) now becomes

$$u_{\mathrm{e},tt} + \frac{1}{12}u_{\mathrm{e},tttt}(x_i,t_n)\Delta t^2 = c^2 u_{\mathrm{e},xx} + c^2\frac{1}{12}u_{\mathrm{e},xxxx}(x_i,t_n)\Delta x^2 + f(x_i,t_n) + \mathcal{O}(\Delta t^4,\Delta x^4) + R_i^n\,.$$

Because u_{e} fulfills the partial differential equation (PDE) (64), the first, third, and fifth terms cancel out, and we are left with

$$R_i^n = \frac{1}{12}u_{\mathrm{e},tttt}(x_i,t_n)\Delta t^2 - c^2\frac{1}{12}u_{\mathrm{e},xxxx}(x_i,t_n)\Delta x^2 + \mathcal{O}(\Delta t^4,\Delta x^4), \tag{67}$$

showing that the scheme (65) is of second order in the time and space mesh spacing.

5.2 Finding correction terms

Can we add correction terms to the PDE and increase the order of R_i^n in (67)? The starting point is

$$[D_tD_tu_{\mathrm{e}} = c^2D_xD_xu_{\mathrm{e}} + f + C + R]_i^n \tag{68}$$

From the previous analysis we simply get (67) again, but now with C:

$$R_i^n + C_i^n = \frac{1}{12}u_{\mathrm{e},tttt}(x_i,t_n)\Delta t^2 - c^2\frac{1}{12}u_{\mathrm{e},xxxx}(x_i,t_n)\Delta x^2 + \mathcal{O}(\Delta t^4,\Delta x^4)\,. \tag{69}$$

The idea is to let C_i^n cancel the Δt^2 and Δx^2 terms to make $R_i^n = \mathcal{O}(\Delta t^4,\Delta x^4)$:

$$C_i^n = \frac{1}{12}u_{\mathrm{e},tttt}(x_i,t_n)\Delta t^2 - c^2\frac{1}{12}u_{\mathrm{e},xxxx}(x_i,t_n)\Delta x^2\,.$$

Essentially, it means that we add a new term

$$C = \frac{1}{12}\left(u_{tttt}\Delta t^2 - c^2u_{xxxx}\Delta x^2\right),$$

to the right-hand side of the PDE. We must either discretize these 4th-order derivatives directly or rewrite them in terms of lower-order derivatives with the aid of the PDE. The latter approach is more feasible. From the PDE we have that

$$\frac{\partial^2}{\partial t^2} = c^2\frac{\partial^2}{\partial x^2},$$

so

$$u_{tttt} = c^2u_{xxtt}, \quad u_{xxxx} = c^{-2}u_{ttxx}\,.$$

Assuming u is smooth enough that $u_{xxtt} = u_{ttxx}$, these relations lead to

$$C = \frac{1}{12}((c^2\Delta t^2 - \Delta x^2)u_{xx})_{tt}\,.$$

A natural discretization is

$$C_i^n = \frac{1}{12}((c^2\Delta t^2 - \Delta x^2)[D_xD_xD_tD_tu]_i^n\,.$$

Writing out $[D_xD_xD_tD_tu]_i^n$ as $[D_xD_x(D_tD_tu)]_i^n$ gives

$$\frac{1}{\Delta t^2}\Bigg(\frac{u_{i+1}^{n+1} - 2u_{i+1}^n + u_{i+1}^{n-1}}{\Delta x^2} - 2 \\ \frac{u_i^{n+1} - 2u_i^n + u_i^{n-1}}{\Delta x^2} + \qquad \frac{u_{i-1}^{n+1} - 2u_{i-1}^n + u_{i-1}^{n-1}}{\Delta x^2}\Bigg)$$

Now the unknown values u_{i+1}^{n+1}, u_i^{n+1}, and u_{i-1}^{n+1} are *coupled*, and we must solve a tridiagonal system to find them. This is in principle straightforward, but it results in an implicit finite difference schemes, while we had a convenient explicit scheme without the correction terms.

5.3 Extension to variable coefficients

Now we address the variable coefficient version of the linear 1D wave equation,

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial}{\partial x}\left(\lambda(x)\frac{\partial u}{\partial x}\right),$$

or written more compactly as

$$u_{tt} = (\lambda u_x)_x\,. \tag{70}$$

The discrete counterpart to this equation, using arithmetic mean for λ and centered differences, reads

$$[D_tD_tu = D_x\overline{\lambda}^xD_xu]_i^n\,. \tag{71}$$

The truncation error is the residual R in the equation

$$[D_tD_tu_\mathrm{e} = D_x\overline{\lambda}^xD_xu_\mathrm{e} + R]_i^n\,. \tag{72}$$

The difficulty in the present is how to compute the truncation error of the term $[D_x\overline{\lambda}^xD_xu_\mathrm{e}]_i^n$.

We start by writing out the outer operator:

$$[D_x\overline{\lambda}^xD_xu_\mathrm{e}]_i^n = \frac{1}{\Delta x}\left([\overline{\lambda}^xD_xu_\mathrm{e}]_{i+\frac{1}{2}}^n - [\overline{\lambda}^xD_xu_\mathrm{e}]_{i-\frac{1}{2}}^n\right). \tag{73}$$

With the aid of (5) and (13) we have

$$
\begin{aligned}
[D_x u_{\mathrm{e}}]^n_{i+\frac{1}{2}} &= u_{\mathrm{e},x}(x_{i+\frac{1}{2}}, t_n) + \frac{1}{24} u_{\mathrm{e},xxx}(x_{i+\frac{1}{2}}, t_n)\Delta x^2 + \mathcal{O}(\Delta x^4), \\
[\overline{\lambda}^x]_{i+\frac{1}{2}} &= \lambda(x_{i+\frac{1}{2}}) + \frac{1}{8}\lambda''(x_{i+\frac{1}{2}})\Delta x^2 + \mathcal{O}(\Delta x^4), \\
[\overline{\lambda}^x D_x u_{\mathrm{e}}]^n_{i+\frac{1}{2}} &= (\lambda(x_{i+\frac{1}{2}}) + \frac{1}{8}\lambda''(x_{i+\frac{1}{2}})\Delta x^2 + \mathcal{O}(\Delta x^4))\times \\
&\quad (u_{\mathrm{e},x}(x_{i+\frac{1}{2}}, t_n) + \frac{1}{24} u_{\mathrm{e},xxx}(x_{i+\frac{1}{2}}, t_n)\Delta x^2 + \mathcal{O}(\Delta x^4)) \\
&= \lambda(x_{i+\frac{1}{2}}) u_{\mathrm{e},x}(x_{i+\frac{1}{2}}, t_n) + \lambda(x_{i+\frac{1}{2}})\frac{1}{24} u_{\mathrm{e},xxx}(x_{i+\frac{1}{2}}, t_n)\Delta x^2 + \\
&\quad u_{\mathrm{e},x}(x_{i+\frac{1}{2}})\frac{1}{8}\lambda''(x_{i+\frac{1}{2}})\Delta x^2 + \mathcal{O}(\Delta x^4) \\
&= [\lambda u_{\mathrm{e},x}]^n_{i+\frac{1}{2}} + G^n_{i+\frac{1}{2}}\Delta x^2 + \mathcal{O}(\Delta x^4),
\end{aligned}
$$

where we have introduced the short form

$$
G^n_{i+\frac{1}{2}} = (\frac{1}{24} u_{\mathrm{e},xxx}(x_{i+\frac{1}{2}}, t_n)\lambda((x_{i+\frac{1}{2}}) + u_{\mathrm{e},x}(x_{i+\frac{1}{2}}, t_n)\frac{1}{8}\lambda''(x_{i+\frac{1}{2}}))\Delta x^2 \,.
$$

Similarly, we find that

$$
[\overline{\lambda}^x D_x u_{\mathrm{e}}]^n_{i-\frac{1}{2}} = [\lambda u_{\mathrm{e},x}]^n_{i-\frac{1}{2}} + G^n_{i-\frac{1}{2}}\Delta x^2 + \mathcal{O}(\Delta x^4) \,.
$$

Inserting these expressions in the outer operator (73) results in

$$
\begin{aligned}
[D_x\overline{\lambda}^x D_x u_{\mathrm{e}}]^n_i &= \frac{1}{\Delta x}([\overline{\lambda}^x D_x u_{\mathrm{e}}]^n_{i+\frac{1}{2}} - [\overline{\lambda}^x D_x u_{\mathrm{e}}]^n_{i-\frac{1}{2}}) \\
&= \frac{1}{\Delta x}([\lambda u_{\mathrm{e},x}]^n_{i+\frac{1}{2}} + G^n_{i+\frac{1}{2}}\Delta x^2 - [\lambda u_{\mathrm{e},x}]^n_{i-\frac{1}{2}} - G^n_{i-\frac{1}{2}}\Delta x^2 + \mathcal{O}(\Delta x^4)) \\
&= [D_x \lambda u_{\mathrm{e},x}]^n_i + [D_x G]^n_i \Delta x^2 + \mathcal{O}(\Delta x^4) \,.
\end{aligned}
$$

The reason for $\mathcal{O}(\Delta x^4)$ in the remainder is that there are coefficients in front of this term, say $H\Delta x^4$, and the subtraction and division by Δx results in $[D_x H]^n_i \Delta x^4$.

We can now use (5) to express the D_x operator in $[D_x \lambda u_{\mathrm{e},x}]^n_i$ as a derivative and a truncation error:

$$
[D_x \lambda u_{\mathrm{e},x}]^n_i = \frac{\partial}{\partial x}\lambda(x_i) u_{\mathrm{e},x}(x_i, t_n) + \frac{1}{24}(\lambda u_{\mathrm{e},x})_{xxx}(x_i, t_n)\Delta x^2 + \mathcal{O}(\Delta x^4) \,.
$$

Expressions like $[D_x G]^n_i \Delta x^2$ can be treated in an identical way,

$$
[D_x G]^n_i \Delta x^2 = G_x(x_i, t_n)\Delta x^2 + \frac{1}{24} G_{xxx}(x_i, t_n)\Delta x^4 + \mathcal{O}(\Delta x^4) \,.
$$

There will be a number of terms with the Δx^2 factor. We lump these now into $\mathcal{O}(\Delta x^2)$. The result of the truncation error analysis of the spatial derivative is therefore summarized as

$$[D_x\overline{\lambda}^x D_x u_{\mathrm{e}}]_i^n = \frac{\partial}{\partial x}\lambda(x_i)u_{\mathrm{e},x}(x_i,t_n) + \mathcal{O}(\Delta x^2)\,.$$

After having treated the $[D_tD_tu_{\mathrm{e}}]_i^n$ term as well, we achieve

$$R_i^n = \mathcal{O}(\Delta x^2) + \frac{1}{12}u_{\mathrm{e},tttt}(x_i,t_n)\Delta t^2\,.$$

The main conclusion is that the scheme is of second-order in time and space also in this variable coefficient case. The key ingredients for second order are the centered differences and the arithmetic mean for λ: all those building blocks feature second-order accuracy.

5.4 1D wave equation on a staggered mesh

5.5 Linear wave equation in 2D/3D

The two-dimensional extension of (63) takes the form

$$\frac{\partial^2 u}{\partial t^2} = c^2\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + f(x,y,t), \quad (x,y)\in(0,L)\times(0,H),\ t\in(0,T], \tag{74}$$

where now $c(x,y)$ is the constant wave velocity of the physical medium $[0,L]\times[0,H]$. In the compact notation, the PDE (74) can be written

$$u_{tt} = c^2(u_{xx}+u_{yy}) + f(x,y,t), \quad (x,y)\in(0,L)\times(0,H),\ t\in(0,T], \tag{75}$$

in 2D, while the 3D version reads

$$u_{tt} = c^2(u_{xx}+u_{yy}+u_{zz}) + f(x,y,z,t), \tag{76}$$

for $(x,y,z)\in(0,L)\times(0,H)\times(0,B)$ and $t\in(0,T]$.

Approximating the second-order derivatives by the standard formula (11) yields the scheme

$$[D_tD_tu = c^2(D_xD_xu + D_yD_yu) + f]_{i,j,k}^n\,. \tag{77}$$

The truncation error is found from

$$[D_tD_tu_{\mathrm{e}} = c^2(D_xD_xu_{\mathrm{e}} + D_yD_yu_{\mathrm{e}}) + f + R]^n\,. \tag{78}$$

The calculations from the 1D case can be repeated to the terms in the y and z directions. Collecting terms that fulfill the PDE, we end up with

$$R_{i,j,k}^n = [\frac{1}{12}u_{\mathrm{e},tttt}\Delta t^2 - c^2\frac{1}{12}\left(u_{\mathrm{e},xxxx}\Delta x^2 + u_{\mathrm{e},yyyy}\Delta x^2 + u_{\mathrm{e},zzzz}\Delta z^2\right)]_{i,j,k}^n + \mathcal{O}(\Delta t^4,\Delta x^4,\Delta y^4,\Delta z^4)\,. \tag{79}$$

6 Truncation errors in diffusion equations

6.1 Linear diffusion equation in 1D

The standard, linear, 1D diffusion equation takes the form

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2} + f(x,t), \quad x \in (0,L),\ t \in (0,T], \tag{80}$$

where $\alpha > 0$ is the constant diffusion coefficient. A more compact form of the diffusion equation is $u_t = \alpha u_{xx} + f$.

The spatial derivative in the diffusion equation, $\alpha u_x x$, is commonly discretized as $[D_x D_x u]_i^n$. The time-derivative, however, can be treated by a variety of methods.

The Forward Euler scheme in time. Let us start with the simple Forward Euler scheme:

$$[D_t^+ u = \alpha D_x D_x u + f]^n \,.$$

The truncation error arises as the residual R when inserting the exact solution u_{e} in the discrete equations:

$$[D_t^+ u_{\mathrm{e}} = \alpha D_x D_x u_{\mathrm{e}} + f + R]_i^n \,.$$

Now, using (8) and (11), we can transform the difference operators to derivatives:

$$u_{\mathrm{e},t}(x_i,t_n) + \frac{1}{2}u_{\mathrm{e},tt}(t_n)\Delta t + \mathcal{O}(\Delta t^2) = \alpha u_{\mathrm{e},xx}(x_i,t_n) + \frac{\alpha}{12}u_{\mathrm{e},xxxx}(x_i,t_n)\Delta x^2 + \mathcal{O}(\Delta x^4) + f(x_i,t_n) + R_i^n \,.$$

The terms $u_{\mathrm{e},t}(x_i,t_n) - \alpha u_{\mathrm{e},xx}(x_i,t_n) - f(x_i,t_n)$ vansih because u_{e} solves the PDE. The truncation error then becomes

$$R_i^n = \frac{1}{2}u_{\mathrm{e},tt}(t_n)\Delta t + \mathcal{O}(\Delta t^2) - \frac{\alpha}{12}u_{\mathrm{e},xxxx}(x_i,t_n)\Delta x^2 + \mathcal{O}(\Delta x^4) \,.$$

The Crank-Nicolson scheme in time. The Crank-Nicolson method consists of using a centered difference for u_t and an arithmetic average of the $u_x x$ term:

$$[D_t u]_i^{n+\frac{1}{2}} = \alpha \frac{1}{2}([D_x D_x u]_i^n + [D_x D_x u]_i^{n+1} + f_i^{n+\frac{1}{2}} \,.$$

The equation for the truncation error is

$$[D_t u_{\mathrm{e}}]_i^{n+\frac{1}{2}} = \alpha \frac{1}{2}([D_x D_x u_{\mathrm{e}}]_i^n + [D_x D_x u_{\mathrm{e}}]_i^{n+1}) + f_i^{n+\frac{1}{2}} + R_i^{n+\frac{1}{2}} \,.$$

To find the truncation error, we start by expressing the arithmetic average in terms of values at time $t_{n+\frac{1}{2}}$. According to (13),

$$\frac{1}{2}([D_xD_xu_\mathrm{e}]_i^n + [D_xD_xu_\mathrm{e}]_i^{n+1}) = [D_xD_xu_\mathrm{e}]_i^{n+\frac{1}{2}} + \frac{1}{8}[D_xD_xu_{\mathrm{e},tt}]_i^{n+\frac{1}{2}}\Delta t^2 + \mathcal{O}(\Delta t^4)\,.$$

With (11) we can express the difference operator D_xD_xu in terms of a derivative:

$$[D_xD_xu_\mathrm{e}]_i^{n+\frac{1}{2}} = u_{\mathrm{e},xx}(x_i, t_{n+\frac{1}{2}}) + \frac{1}{12}u_{\mathrm{e},xxxx}(x_i, t_{n+\frac{1}{2}})\Delta x^2 + \mathcal{O}(\Delta x^4)\,.$$

The error term from the arithmetic mean is similarly expanded,

$$\frac{1}{8}[D_xD_xu_{\mathrm{e},tt}]_i^{n+\frac{1}{2}}\Delta t^2 = \frac{1}{8}u_{\mathrm{e},ttxx}(x_i, t_{n+\frac{1}{2}})\Delta t^2 + \mathcal{O}(\Delta t^2\Delta x^2)$$

The time derivative is analyzed using (5):

$$[D_tu]_i^{n+\frac{1}{2}} = u_{\mathrm{e},t}(x_i, t_{n+\frac{1}{2}}) + \frac{1}{24}u_{\mathrm{e},ttt}(x_i, t_{n+\frac{1}{2}})\Delta t^2 + \mathcal{O}(\Delta t^4)\,.$$

Summing up all the contributions and notifying that

$$u_{\mathrm{e},t}(x_i, t_{n+\frac{1}{2}}) = \alpha u_{\mathrm{e},xx}(x_i, t_{n+\frac{1}{2}}) + f(x_i, t_{n+\frac{1}{2}}),$$

the truncation error is given by

$$\begin{aligned} R_i^{n+\frac{1}{2}} = &\frac{1}{8}u_{\mathrm{e},xx}(x_i, t_{n+\frac{1}{2}})\Delta t^2 + \frac{1}{12}u_{\mathrm{e},xxxx}(x_i, t_{n+\frac{1}{2}})\Delta x^2 + \\ &\frac{1}{24}u_{\mathrm{e},ttt}(x_i, t_{n+\frac{1}{2}})\Delta t^2 + + \mathcal{O}(\Delta x^4) + \mathcal{O}(\Delta t^4) + \mathcal{O}(\Delta t^2\Delta x^2) \end{aligned}$$

6.2 Linear diffusion equation in 2D/3D

6.3 A nonlinear diffusion equation in 2D

7 Exercises

Exercise 1: Truncation error of a weighted mean

Derive the truncation error of the weighted mean in (12).

Hint. Expand u_e^{n+1} and u_e^n around $t_{n+\theta}$.
Filename: `trunc_weighted_mean.pdf`.

Exercise 2: Simulate the error of a weighted mean

We consider the weighted mean

$$u_\mathrm{e}(t_n) \approx \theta u_\mathrm{e}^{n+1} + (1-\theta)u_\mathrm{e}^n\,.$$

Choose some specific function for $u_\mathrm{e}(t)$ and compute the error in this approximation for a sequence of decreasing $\Delta t = t_{n+1} - t_n$ and for $\theta = 0, 0.25, 0.5, 0.75, 1$. Assuming that the error equals $C\Delta t^r$, for some constants C and r, compute r for the two smallest Δt values for each choice of θ and compare with the truncation error (12). Filename: `trunc_theta_avg.py`.

Exercise 3: Verify a truncation error formula

Set up a numerical experiment as explained in Section 3.5 for verifying the formula (10). Filename: `trunc_backward_2level.py`.

Exercise 4: Truncation error of the Backward Euler scheme

Derive the truncation error of the Backward Euler scheme for the decay ODE $u' = -au$ with constant a. Extend the analysis to cover the variable-coefficient case $u' = -a(t)u + b(t)$. Filename: `trunc_decay_BE.py`.

Exercise 5: Empirical estimation of truncation errors

Use the ideas and tools from Section 3.5 to estimate the rate of the truncation error of the Backward Euler and Crank-Nicolson schemes applied to the exponential decay model $u' = -au$, $u(0) = I$.

Hint. In the Backward Euler scheme, the truncation error can be estimated at mesh points $n = 1, \ldots, N$, while the truncation error must be estimated at midpoints $t_{n+\frac{1}{2}}$, $n = 0, \ldots, N-1$ for the Crank-Nicolson scheme. The `truncation_error(dt, N)` function to be supplied to the `estimate` function needs to carefully implement these details and return the right `t` array such that `t[i]` is the time point corresponding to the quantities `R[i]` and `R_a[i]`.

Filename: `trunc_decay_BNCN.py`.

Exercise 6: Correction term for a Backward Euler scheme

Consider the model $u' = -au$, $u(0) = I$. Use the ideas of Section 3.6 to add a correction term to the ODE such that the Backward Euler scheme applied to the perturbed ODE problem is of second order in Δt. Find the amplification factor.

Filename: `trunc_decay_BE_corr.pdf`.

Exercise 7: Verify the effect of correction terms

The program `decay_convrate.py`[4] solves $u' = -au$, $u(0) = I$, by the θ-rule and computes convergence rates. Copy this file and adjust a in the `solver` function such that it incorporates correction terms. Run the program to verify that the error from the Forward and Backward Euler schemes with perturbed a is $\mathcal{O}(\Delta t^2)$, while the error arising from the Crank-Nicolson scheme with perturbed a is $\mathcal{O}(\Delta t^4)$. Filename: `trunc_decay_corr_verify.py`.

Exercise 8: Truncation error of the Crank-Nicolson scheme

The variable-coefficient ODE $u' = -a(t)u + b(t)$ can be discretized in two different ways by the Crank-Nicolson scheme, depending on whether we use averages for a and b or compute them at the midpoint $t_{n+\frac{1}{2}}$:

$$[D_t u = -a\overline{u}^t + b]^{n+\frac{1}{2}}, \tag{81}$$

$$[D_t u = \overline{-au + b}^t]^{n+\frac{1}{2}}\,. \tag{82}$$

Compute the truncation error in both cases. Filename: `trunc_decay_CN_vc.pdf`.

[4] `http://tinyurl.com/jvzzcfn/decay/decay_convrate.py`

Exercise 9: Truncation error of $u' = f(u,t)$

Consider the general nonlinear first-order scalar ODE

$$u'(t) = f(u(t), t)\,.$$

Show that the truncation error in the Forward Euler scheme,

$$[D_t^+ u = f(u,t)]^n,$$

and in the Backward Euler scheme,

$$[D_t^- u = f(u,t)]^n,$$

both are of first order, regardless of what f is.

Showing the order of the truncation error in the Crank-Nicolson scheme,

$$[D_t u = f(u,t)]^{n+\frac{1}{2}},$$

is somewhat more involved: Taylor expand u_{e}^n, u_{e}^{n+1}, $f(u_{\mathrm{e}}^n, t_n)$, and $f(u_{\mathrm{e}}^{n+1}, t_{n+1})$ around $t_{n+\frac{1}{2}}$, and use that

$$\frac{df}{dt} = \frac{\partial f}{\partial u}u' + \frac{\partial f}{\partial t}\,.$$

Check that the derived truncation error is consistent with previous results for the case $f(u,t) = -au$. Filename: `trunc_nonlinear_ODE.pdf`.

Exercise 10: Truncation error of $[D_t D_t u]^n$

Derive the truncation error of the finite difference approximation (11) to the second-order derivative. Filename: `trunc_d2u.pdf`.

Exercise 11: Investigate the impact of approximating $u'(0)$

Section 4.1 describes two ways of discretizing the initial conditon $u'(0) = V$ for a vibration model $u'' + \omega^2 u = 0$: a centered difference $[D_{2t}u = V]^0$ or a forward difference $[D_t^+ u = V]^0$. The program `vib_undamped.py`[5] solves $u'' + \omega^2 u = 0$ with $[D_{2t}u = 0]^0$ and features a function `convergence_rates` for computing the order of the error in the numerical solution. Modify this program such that it applies the forward difference $[D_t^+ u = 0]^0$ and report how this simpler and more convenient approximation impacts the overall convergence rate of the scheme. Filename: `trunc_vib_ic_fw.py`.

Exercise 12: Investigate the accuracy of a simplified scheme

Consider the ODE

$$mu'' + \beta|u'|u' + s(u) = F(t)\,.$$

[5] `http://tinyurl.com/jvzzcfn/vib/vib_undamped.py`

The term $|u'|u'$ quickly gives rise to nonlinearities and complicates the scheme. Why not simply apply a backward difference to this term such that it only involves known values? That is, we propose to solve

$$[mD_tD_tu + \beta|D_t^-u|D_t^-u + s(u) = F]^n \,.$$

Drop the absolute value for simplicity and find the truncation error of the scheme. Perform numerical experiments with the scheme and compared with the one based on centered differences. Can you illustrate the accuracy loss visually in real computations, or is the asymptotic analysis here mainly of theoretical interest? Filename: `trunc_vib_bw_damping.pdf`.

Index