Solving nonlinear ODE and PDE problems

Hans Petter Langtangen

edical Computing, Simula Research Laboratory and Department of Informatics, Universi

Dec 14, 2014

Note: Preliminary version (expect typos).

Contents

In a linear differential equation all terms involving the unknown fu
are linear in the unknown functions or their derivatives. Linear here mea
the unknown function or a derivative of it is multiplied by a number or a
function. All other differential equations are non-linear.

The easiest way to see if an equation is nonlinear is to spot nonlinea
where the unknown functions or their derivatives are multiplied by eac
For example, in

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

the terms involving the unknown function u are linear: u' contains the de
of the unknown function multiplied by unity, and au contains the u
function multiplied by a known function. However,

$$u'(t) = u(t)(1 - u(t)),$$

is nonlinear because of the term $-u^2$ where the unknown function is mu
by itself. Also

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

is nonlinear because of the term uu_x where the unknown function app
a product with itself or one if its derivatives. Another example of a no
equation is

$$u'' + \sin(u) = 0,$$

because $\sin(u)$ contains products of u,

$$\sin(u) = u - \frac{1}{3}u^3 + \ldots$$

Mathematical proof.

To really prove mathematically that some differential equation in an unknown u is linear, show for each term $T(u)$ that with $u = au_1 + bu_2$ for constants a and b,

$$T(au_1 + bu_2) = aT(u_1) + bT(u_2)\,.$$

For example, the term $T(u) = (\sin^2 t)u'(t)$ is linear because

$$T(au_1+bu_2) = (\sin^2 t)(au_1(t)+bu_2(t)) = a(\sin^2 t)u_1(t)+b(\sin^2 t)u_2(t) = aT(u_1)+b$$

However, $T(u) = \sin u$ is nonlinear because

$$T(au_1 + bu_2) = \sin(au_1 + bu_2) \neq a\sin u_1 + b\sin u_2\,.$$

A series of forthcoming examples will explain how to tackle nonlinear differtial equations with various techniques.

Introduction of basic concepts

onsider the (scaled) logistic equation

$$u'(t) = u(t)(1 - u(t))\,. \tag{1}$$

his is a nonlinear differential equation which will be solved by different strategies the following. A time discretization of (1) will either lead to a linear algebraic quation or a nonlinear algebraic equation at each time level. In the former ase, the time discretization method transforms the nonlinear ODE into linear ubproblems at each time level, and the solution is straightforward to find nce linear algebraic equations are easy to solve by hand. However, when the me discretization leads to nonlinear algebraic equations, we cannot (except in ery rare cases) solve these without turning to approximate, iterative solution ethods.

The following subsections first introduce various methods using (1):

- explicit time discretization methods (with no need to solve nonlinear algebraic equations)
- implicit Backward Euler discretization, leading to nonlinear algebraic equations solved by

  - an exact analytical technique
  - Picard iteration based on manual linearization
  - a single Picard step
  - Newton's method
- Implicit Crank-Nicolson discretization and linearization via a ge mean formula

Thereafter, we compare the performance of the various approaches. Des simplicity of (1), the conclusions reveal typical features of the various r in much more complicated nonlinear PDE problems.

1.1 Linearization by explicit time discretization

A Forward Euler method to solve (1) results in

$$\frac{u^{n+1} - u^n}{\Delta t} = u^n(1 - u^n),$$

which is a *linear* algebraic equation for the unknown value u^{n+1}. The nonl in the original equation poses in this case no difficulty in the discrete a equation. Any other explicit scheme in time will also give only linear a equations to solve. For example, a typical 2nd-order Runge-Kutta met (1) reads,

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

The first step is linear in the unknown u^*. Then u^* is known in the ne which is linear in the unknown u^{n+1} .

1.2 Exact solution of nonlinear equations

Switching to a Backward Euler scheme for (1),

$$\frac{u^n - u^{n-1}}{\Delta t} = u^n(1 - u^n),$$

results in a nonlinear algebraic equation for the unknown value u^n. The e is of quadratic type:

$$\Delta t(u^n)^2 + (1 - \Delta t)u^n - u^{n-1} = 0\,.$$

We shall now introduce a shorter and often cleaner notation for nc algebraic equations at a given time level. The notation is inspired by the notation, i.e., variable names, used in a program, especially in more ac

artial differential equation problems. The unknown in the algebraic equation is enoted by u, while $u^{(1)}$ is the value of the unknown at the previous time level n general $u^{(\ell)}$ is the value of the unknown ℓ levels back in time). The notation ill be frequently used in later sections. What is meant by u (the exact solution f the PDE problem, the numerical approximation to the exact solution, or the nknown solution at a certain time level) should be evident from the context.

The quadratic equation for the unknown u^n in (2) can with the new notation e written

$$F(u) = \Delta t u^2 + (1 - \Delta t)u - u^{(1)} = 0\,. \tag{3}$$

he solution is readily found to be

$$u = \frac{1}{2\Delta t}\left(-1 + \Delta t \pm \sqrt{(1-\Delta t)^2 - 4\Delta t u^{(1)}}\right)\,. \tag{4}$$

Now we encounter a fundamental challenge with nonlinear algebraic equations: ıe equation may have more than one solution. How do we pick the right solution? ı the present simple case we can expand the square root in a series in Δt and uncate after the linear term since the Backward Euler scheme will introduce n error proportional to Δt anyway. Using `sympy` we find the following Taylor eries expansions of the roots:

```
>>> import sympy as sp
>>> dt, u_1, u = sp.symbols('dt u_1 u')
>>> r1, r2 = sp.solve(dt*u**2 + (1-dt)*u - u_1, u)  # find roots
>>> r1
(dt - sqrt(dt**2 + 4*dt*u_1 - 2*dt + 1) - 1)/(2*dt)
>>> r2
(dt + sqrt(dt**2 + 4*dt*u_1 - 2*dt + 1) - 1)/(2*dt)
>>> print r1.series(dt, 0, 2)
-1/dt + 1 - u_1 + dt*(u_1**2 - u_1) + O(dt**2)
>>> print r2.series(dt, 0, 2)
u_1 + dt*(-u_1**2 + u_1) + O(dt**2)
```

We see that the `r1` root, corresponding to a minus sign in front of the square oot in (4), behaves as $1/\Delta t$ and will therefore blow up as $\Delta t \to 0$! Therefore, nly the `r2` root is of relevance in this case.

.3 Linearization

When the time integration of an ODE results in a nonlinear algebraic equation, e must normally find its solution by defining a sequence of linear equations nd hope that the solutions of these linear equations converge to the desired olution of the nonlinear algebraic equation. Usually this means solving the linear quation repeatedly in an iterative fashion. Alternatively, the nonlinear equation an sometimes be approximated by one linear equation, and consequently there no need for iteration.

Constructing a linear equation from a nonlinear one requires *linearization* f each nonlinear term. This can be done manually as in Picard iteration, or

fully algorithmically as in Newton's method. Examples will best illustr to linearize nonlinear problems.

1.4 Picard iteration

Let us write (3) in a more compact form

$$F(u) = au^2 + bu + c = 0,$$

with $a = \Delta t$, $b = 1 - \Delta t$, and $c = -u^{(1)}$. Let u^- be an available approx of the unknown u. Then we can linearize the term u^2 simply by writi The resulting equation, $\hat{F}(u) = 0$, is now linear and hence easy to solve

$$F(u) \approx \hat{F}(u) = au^- u + bu + c = 0\,.$$

Since the equation $\hat{F} = 0$ is only approximate, the solution u does not e exact solution u_e of the exact equation $F(u_\mathrm{e}) = 0$, but we can hope t closer to u_e than u^- is, and hence it makes sense to repeat the proced set $u^- = u$ and solve $\hat{F}(u) = 0$ again.

The idea of turning a nonlinear equation into a linear one by u approximation u^- of u in nonlinear terms is a widely used approach tl under many names: *fixed-point iteration*, the method of *successive subst nonlinear Richardson iteration*, and *Picard iteration*. We will stick to tl name.

Picard iteration for solving the nonlinear equation arising from the Ba Euler discretization of the logistic equation can be written as

$$u = -\frac{c}{au^- + b}, \quad u^- \leftarrow u\,.$$

The iteration is started with the value of the unknown at the previous tir $u^- = u^{(1)}$.

Some prefer an explicit iteration counter as superscript in the mathe notation. Let u^k be the computed approximation to the solution in iter In iteration $k+1$ we want to solve

$$au^k u^{k+1} + bu^{k+1} + c = 0 \quad \Rightarrow \quad u^{k+1} = -\frac{c}{au^k + b}, \quad k = 0, 1, \ldots$$

Since we need to perform the iteration at every time level, the time level is often also included:

$$au^{n,k}u^{n,k+1} + bu^{n,k+1} - u^{n-1} = 0 \quad \Rightarrow \quad u^{n,k+1} = \frac{u^n}{au^{n,k} + b}, \quad k = 0$$

with the start value $u^{n,0} = u^{n-1}$ and the final converged value $u^n =$ sufficiently large k.

However, we will normally apply a mathematical notation in our final f that is as close as possible to what we aim to write in a computer code a it becomes natural to use u and u^- instead of u^{k+1} and u^k or $u^{n,k+1}$ a

topping criteria. The iteration method can typically be terminated when ıe change in the solution is smaller than a tolerance ϵ_u:

$$|u - u^-| \leq \epsilon_u,$$

r when the residual in the equation is sufficiently small (ϵ_r),

$$|F(u)| = |au^2 + bu + c| < \epsilon_r \,.$$

. single Picard iteration. Instead of iterating until a stopping criterion is ılfilled, one may iterate a specific number of times. Just one Picard iteration is opular as this corresponds to the intuitive idea of approximating a nonlinear ɛrm like $(u^n)^2$ by $u^{n-1}u^n$. This follows from the linearization u^-u^n and the ıitial choice of $u^- = u^{n-1}$ at time level t_n. In other words, a single Picard eration corresponds to using the solution at the previous time level to linearize onlinear terms. The resulting discretization becomes

$$\frac{u^n - u^{n-1}}{\Delta t} = u^n(1 - u^{n-1}), \tag{5}$$

hich is a linear algebraic equation in the unknown u^n, and therefore we can ısily solve for u^n, and there is no need for any alternative notation.

We shall later refer to the strategy of taking one Picard step, or equivalently, nearizing terms with use of the solution at the previous time step, as the *Picard1* ıethod. It is a widely used approach in science and technology, but with some mitations if Δt is not sufficiently small (as will be illustrated later).

Notice.

Equation (5) does not correspond to a "pure" finite difference method where the equation is sampled at a point and derivatives replaced by differences (because the u^{n-1} term on the right-hand side must then be u^n). The best interpretation of the scheme (5) is a Backward Euler difference combined with a single (perhaps insufficient) Picard iteration at each time level, with the value at the previous time level as start for the Picard iteration.

.5 Linearization by a geometric mean

/e consider now a Crank-Nicolson discretization of (1). This means that the me derivative is approximated by a centered difference,

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

ritten out as

$$\frac{u^{n+1} - u^n}{\Delta t} = u^{n+\frac{1}{2}} - (u^{n+\frac{1}{2}})^2 \,. \tag{6}$$

The term $u^{n+\frac{1}{2}}$ is normally approximated by an arithmetic mean,

$$u^{n+\frac{1}{2}} \approx \frac{1}{2}(u^n + u^{n+1}),$$

such that the scheme involves the unknown function only at the time leve we actually compute it. The same arithmetic mean applied to the nc term gives

$$(u^{n+\frac{1}{2}})^2 \approx \frac{1}{4}(u^n + u^{n+1})^2,$$

which is nonlinear in the unknown u^{n+1}. However, using a *geometric n* $(u^{n+\frac{1}{2}})^2$ is a way of linearizing the nonlinear term in (6):

$$(u^{n+\frac{1}{2}})^2 \approx u^n u^{n+1} \,.$$

Using an arithmetic mean on the linear $u^{n+\frac{1}{2}}$ term in (6) and a geometr for the second term, results in a linearized equation for the unknown u

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

which can readily be solved:

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

This scheme can be coded directly, and since there is no nonlinear a equation to iterate over, we skip the simplified notation with u for u $u^{(1)}$ for u^n. The technique with using a geometric average is an exa transforming a nonlinear algebraic equation to a linear one, without a for iterations.

The geometric mean approximation is often very effective for lin quadratic nonlinearities. Both the arithmetic and geometric mean app tions have truncation errors of order Δt^2 and are therefore compatible truncation error $\mathcal{O}(\Delta t)$ of the centered difference approximation for u Crank-Nicolson method.

Applying the operator notation for the means and finite differen linearized Crank-Nicolson scheme for the logistic equation can be co expressed as

$$[D_t u = \overline{u}^t + \overline{u^2}^{t,g}]^{n+\frac{1}{2}} \,.$$

Remark.

If we use an arithmetic instead of a geometric mean for the nonl term in (6), we end up with a nonlinear term $(u^{n+1})^2$. This term c linearized as u^-u^{n+1} in a Picard iteration approach and in particul

$u^n u^{n+1}$ in a Picard1 iteration approach. The latter gives a scheme almost identical to the one arising from a geometric mean (the difference in u^{n+1} being $\frac{1}{4}\Delta t u^n(u^{n+1}-u^n) \approx \frac{1}{4}\Delta t^2 u' u$, i.e., a difference of $\mathcal{O}(\Delta t^2)$).

.6 Newton's method

he Backward Euler scheme (2) for the logistic equation leads to a nonlinear lgebraic equation (3). Now we write any nonlinear algebraic equation in the eneral and compact form

$$F(u) = 0\,.$$

ewton's method linearizes this equation by approximating $F(u)$ by its Taylor eries expansion around a computed value u^- and keeping only the linear part:

$$\begin{aligned} F(u) &= F(u^-) + F'(u^-)(u-u^-) + \frac{1}{2}F''(u^-)(u-u^-)^2 + \cdots \\ &\approx F(u^-) + F'(u^-)(u-u^-) = \hat{F}(u)\,. \end{aligned}$$

he linear equation $\hat{F}(u) = 0$ has the solution

$$u = u^- - \frac{F(u^-)}{F'(u^-)}\,.$$

xpressed with an iteration index in the unknown, Newton's method takes on ıe more familiar mathematical form

$$u^{k+1} = u^k - \frac{F(u^k)}{F'(u^k)}, \quad k = 0, 1, \ldots$$

It can be shown that the error in iteration $k+1$ of Newton's method is the quare of the error in iteration k, a result referred to as *quadratic convergence*. his means that for small errors the method converges very fast, and in particular uch faster than Picard iteration and other iteration methods. (The proof of ıis result is found in most textbooks on numerical analysis.) However, the uadratic convergence appears only if u^k is sufficiently close to the solution. urther away from the solution the method can easily converge very slowly or iverge. The reader is encouraged to do Exercise 2 to get a better understanding ɔr the behavior of the method.

Application of Newton's method to the logistic equation discretized by the ackward Euler method is straightforward as we have

$$F(u) = au^2 + bu + c, \quad a = \Delta t,\ b = 1 - \Delta t,\ c = -u^{(1)},$$

nd then

$$F'(u) = 2au + b\,.$$

The iteration method becomes

$$u = u^- + \frac{a(u^-)^2 + bu^- + c}{2au^- + b}, \quad u^- \leftarrow u\,.$$

At each time level, we start the iteration by setting $u^- = u^{(1)}$. Stopping as listed for the Picard iteration can be used also for Newton's method

An alternative mathematical form, where we write out a, b, and c, a time level counter n and an iteration counter k, takes the form

$$u^{n,k+1} = u^{n,k} + \frac{\Delta t(u^{n,k})^2 + (1-\Delta t)u^{n,k} - u^{n-1}}{2\Delta t u^{n,k} + 1 - \Delta t}, \quad u^{n,0} = u^{n-1}, \quad k =$$

A program implementation is much closer to (7) than to (8), but the better aligned with the established mathematical notation used in the lit

1.7 Relaxation

One iteration in Newton's method or Picard iteration consists of solving problem $\hat{F}(u) = 0$. Sometimes convergence problems arise because t solution u of $\hat{F}(u) = 0$ is "too far away" from the previously computed u^-. A remedy is to introduce a relaxation, meaning that we first solve $\hat{F}$ for a suggested value u^* and then we take u as a weighted mean of what u^-, and what our linearized equation $\hat{F} = 0$ suggests, u^*:

$$u = \omega u^* + (1-\omega)u^-\,.$$

The parameter ω is known as a *relaxation parameter*, and a choice $\omega <$ prevent divergent iterations.

Relaxation in Newton's method can be directly incorporated in tl iteration formula:

$$u = u^- - \omega\frac{F(u^-)}{F'(u^-)}\,.$$

1.8 Implementation and experiments

The program `logistic.py`[1] contains implementations of all the metl scribed above. Below is an extract of the file showing how the Picard and methods are implemented for a Backward Euler discretization of the equation.

[1] `http://tinyurl.com/nm5587k/nonlin/logistic.py`

```
def BE_logistic(u0, dt, Nt, choice='Picard',
                eps_r=1E-3, omega=1, max_iter=1000):
    if choice == 'Picard1':
        choice = 'Picard'
        max_iter = 1

    u = np.zeros(Nt+1)
    iterations = []
    u[0] = u0
    for n in range(1, Nt+1):
        a = dt
        b = 1 - dt
        c = -u[n-1]

        if choice == 'Picard':

            def F(u):
                return a*u**2 + b*u + c

            u_ = u[n-1]
            k = 0
            while abs(F(u_)) > eps_r and k < max_iter:
                u_ = omega*(-c/(a*u_ + b)) + (1-omega)*u_
                k += 1
            u[n] = u_
            iterations.append(k)

        elif choice == 'Newton':

            def F(u):
                return a*u**2 + b*u + c

            def dF(u):
                return 2*a*u + b

            u_ = u[n-1]
            k = 0
            while abs(F(u_)) > eps_r and k < max_iter:
                u_ = u_ - F(u_)/dF(u_)
                k += 1
            u[n] = u_
            iterations.append(k)
    return u, iterations
```

The Crank-Nicolson method utilizing a linearization based on the geometric ıean gives a simpler algorithm:

```
def CN_logistic(u0, dt, Nt):
    u = np.zeros(Nt+1)
    u[0] = u0
    for n in range(0, Nt):
        u[n+1] = (1 + 0.5*dt)/(1 + dt*u[n] - 0.5*dt)*u[n]
    return u
```

We may run experiments with the model problem (1) and the different ɪrategies for dealing with nonlinearities as described above. For a quite coarse me resolution, $\Delta t = 0.9$, use of a tolerance $\epsilon_r = 0.1$ in the stopping criterion ıtroduces an iteration error, especially in the Picard iterations, that is visibly

much larger than the time discretization error due to a large Δt. illustrated by comparing the upper two plots in Figure 1. The one to t has a stricter tolerance $\epsilon = 10^{-3}$, which leads to all the curves correspoı Picard and Newton iteration to be on top of each other (and no changes visually observed by reducing ϵ_r further). The reason why Newton's meth much better than Picard iteration in the upper left plot is that Newton's with one step comes far below the ϵ_r tolerance, while the Picard iteratic on average 7 iterations to bring the residual down to $\epsilon_r = 10^{-1}$, whi insufficient accuracy in the solution of the nonlinear equation. It is obvi the Picard1 method gives significant errors in addition to the time discre unless the time step is as small as in the lower right plot.

The *BE exact* curve corresponds to using the exact solution of the qı equation at each time level, so this curve is only affected by the Backwa time discretization. The *CN gm* curve corresponds to the theoretical accurate Crank-Nicolson discretization, combined with a geometric m linearization. This curve appear as more accurate, especially if we take in the lower right with a small Δt and an appropriately small ϵ_r valu exact curve.

When it comes to the need for iterations, Figure 2 displays the nu iterations required at each time level for Newton's method and Picard i The smaller Δt is, the better starting value we have for the iteration, faster the convergence is. With $\Delta t = 0.9$ Picard iteration requires on 32 iterations per time step, but this number is dramatically reduced reduced.

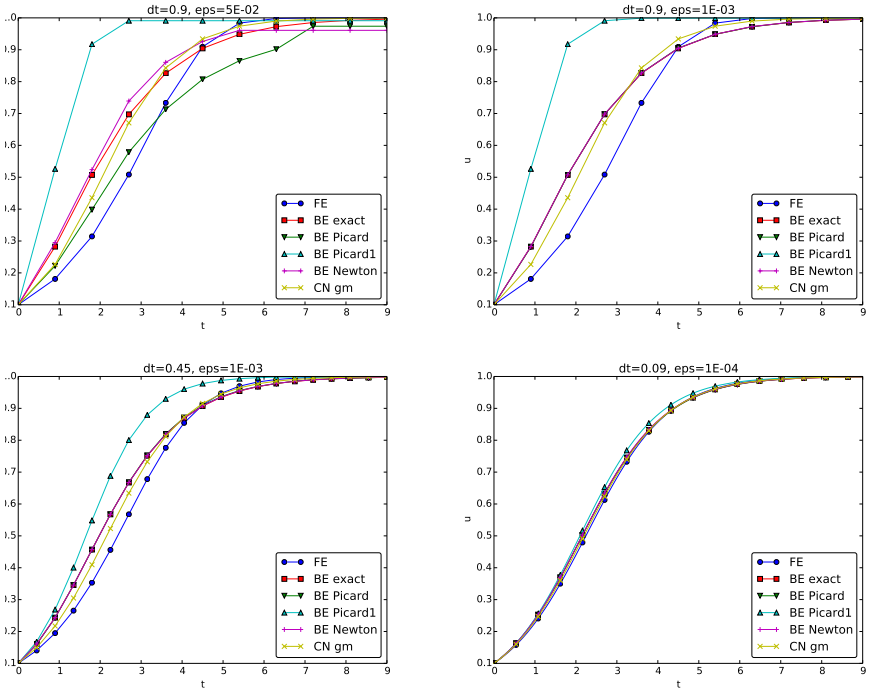
However, introducing relaxation and a parameter $\omega = 0.8$ imm reduces the average of 32 to 7, indicating that for the large $\Delta t = 0.9$ iteration takes too long steps. An approximately optimal value for ω case is 0.5, which results in an average of only 2 iterations! Even more d impact of ω appears when $\Delta t = 1$: Picard iteration does not convergence iterations, but $\omega = 0.5$ again brings the average number of iterations do

Remark. The simple Crank-Nicolson method with a geometric mean quadratic nonlinearity gives visually more accurate solutions than the B Euler discretization. Even with a tolerance of $\epsilon_r = 10^{-3}$, all the met treating the nonlinearities in the Backward Euler discretization gives gra cannot be distinguished. So for accuracy in this problem, the time discre is much more crucial than ϵ_r. Ideally, one should estimate the error in t discretization, as the solution progresses, and set ϵ_r accordingly.

1.9 Generalization to a general nonlinear ODE

Let us see how the various methods in the previous sections can be ap the more generic model

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



igure 1: The impact of solution strategies and for four different time step ngths on the solution.

here f is a nonlinear function of u.

xplicit time discretization. Explicit ODE methods like the Forward Euler cheme, Runge-Kutta methods, Adams-Bashforth methods all evaluate f at me levels where u is already computed, so nonlinearities in f do not pose any ifficulties.

ackward Euler discretization. Approximating u' by a backward difference ads to a Backward Euler scheme, which can be written as

$$F(u^n) = u^n - \Delta t\, f(u^n, t_n) - u^{n-1} = 0,$$

alternatively

$$F(u) = u - \Delta t\, f(u, t_n) - u^{(1)} = 0\,.$$

simple Picard iteration, not knowing anything about the nonlinear structure f f, must approximate $f(u, t_n)$ by $f(u^-, t_n)$:

$$\hat{F}(u) = u - \Delta t\, f(u^-, t_n) - u^{(1)}\,.$$

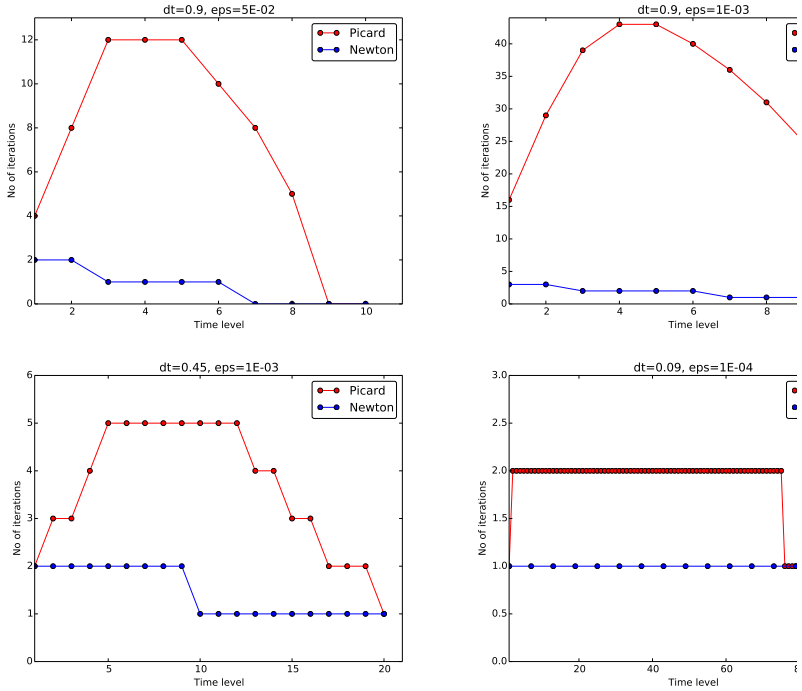


Figure 2: Comparison of the number of iterations at various time le Picard and Newton iteration.

The iteration starts with $u^- = u^{(1)}$ and proceeds with repeating

$$u^* = \Delta t\, f(u^-, t_n) + u^{(1)}, \quad u = \omega u^* + (1-\omega)u^-, \quad u^- \leftarrow u,$$

until a stopping criterion is fulfilled.

Explicit vs implicit treatment of nonlinear terms.

Evaluating f for a known u^- is referred to as *explicit* treatment of f, if $f(u,t)$ has some structure, say $f(u,t) = u^3$, parts of f can involv known u, as in the manual linearization like $(u^-)^2 u$, and then the treat of f is "more implicit" and "less explicit". This terminology is inspir time discretization of $u' = f(u,t)$, where evaluating f for known u v gives explicit schemes, while treating f or parts of f implicitly, ma contribute to the unknown terms in the equation at the new time le

Explicit treatment of f usually means stricter conditions on Δ stability of time discretization schemes. The same goes for itera

techniques for nonlinear algebraic equations: the more f can involve unknowns to be solved for, the faster the convergence may be.

We may say that $f(u,t) = u^3$ is treated explicitly if we evaluate f as $(u^-)^3$, partially implicit if we linearize as $(u^-)^2 u$ and fully implicit if we represent f by u^3. (Of course, the fully implicit representation will require further linearization, but with $f(u,t) = u^2$ a fully implicit treatment is possible if the resulting quadratic equation is solved with a formula.)

For the ODE $u' = -u^3$ with $f(u,t) = -u^3$ and coarse time resolution $\Delta t = 0.4$, Picard iteration with $(u^-)^2 u$ requires 8 iterations with $\epsilon_r = 10^{-3}$ for the first time step, while $(u^-)^3$ leads to 22 iterations. After about 10 time steps both approaches are down to about 2 iterations per time step, but this example shows a potential of treating f more implicitly.

A trick to treat f implicitly in Picard iteration is to evaluate it as $f(u^-,t)u/u^-$. For a polynomial f, $f(u,t) = u^m$, this corresponds to $(u^-)^{m-1}u$. Sometimes this more implicit treatment has no effect, as with $f(u,t) = exp(-u)$ and $f(u,t) = \ln(1+u)$, but with $f(u,t) = sin(2(u+1))$, the $f(u^-,t)u/u^-$ trick leads to 7, 9, and 11 iterations during the first three steps, while $f(u^-,t)$ demands 17, 21, and 20 iterations. (Experiments can be done with the code `ODE_Picard_tricks.py`[a].)

[a] `http://tinyurl.com/nm5587k/nonlin/ODE_Picard_tricks.py`

Newton's method applied to $u' = f(u,t)$ requires the computation of the derivative

$$F'(u) = 1 - \Delta t \frac{\partial f}{\partial u}(u, t_n)\,.$$

Starting with the solution at the previous time level, $u^- = u^{(1)}$, we can just use the standard formula

$$u = u^- - \omega \frac{F(u^-)}{F'(u^-)} = u^- - \omega \frac{u^{(1)} + \Delta t f(u^-, t_n)}{1 - \Delta t \frac{\partial}{\partial u} f(u^-, t_n)}\,. \tag{11}$$

Crank-Nicolson discretization. The standard Crank-Nicolson scheme with arithmetic mean approximation of f takes the form

$$\frac{u^{n+1} - u^n}{\Delta t} = \frac{1}{2}(f(u^{n+1}, t_{n+1}) + f(u^n, t_n))\,.$$

We can write the scheme as a nonlinear algebraic equation

$$F(u) = u - u^{(1)} - \Delta t \frac{1}{2} f(u, t_{n+1}) - \Delta t \frac{1}{2} f(u^{(1)}, t_n) = 0\,. \tag{12}$$

A Picard iteration scheme must in general employ the linearization

$$\hat{F}(u) = u - u^{(1)} - \Delta t \frac{1}{2} f(u^-, t_{n+1}) - \Delta t \frac{1}{2} f(u^{(1)}, t_n),$$

while Newton's method can apply the general formula (11) with $F(u)$ (12) and

$$F'(u) = 1 - \frac{1}{2}\Delta t \frac{\partial f}{\partial u}(u, t_{n+1})\,.$$

1.10 Systems of ODEs

We may write a system of ODEs

$$\begin{aligned}
\frac{d}{dt}u_0(t) &= f_0(u_0(t), u_1(t), \ldots, u_N(t), t),\\
\frac{d}{dt}u_1(t) &= f_1(u_0(t), u_1(t), \ldots, u_N(t), t),\\
&\vdots\\
\frac{d}{dt}u_m(t) &= f_m(u_0(t), u_1(t), \ldots, u_N(t), t),
\end{aligned}$$

as

$$u' = f(u,t), \quad u(0) = U_0,$$

if we interpret u as a vector $u = (u_0(t), u_1(t), \ldots, u_N(t))$ and f as a function with components $(f_0(u,t), f_1(u,t), \ldots, f_N(u,t))$.

Solution methods for scalar ODEs normally generalize in a straightforward way to systems of ODEs simply by using vector arithmetics instead of scalar arithmetics, which corresponds to applying the scalar scheme to each component of the system. For example, here is a backward difference scheme applied to each component,

$$\begin{aligned}
\frac{u_0^n - u_0^{n-1}}{\Delta t} &= f_0(u^n, t_n),\\
\frac{u_1^n - u_1^{n-1}}{\Delta t} &= f_1(u^n, t_n),\\
&\vdots\\
\frac{u_N^n - u_N^{n-1}}{\Delta t} &= f_N(u^n, t_n),
\end{aligned}$$

which can be written more compactly in vector form as

$$\frac{u^n - u^{n-1}}{\Delta t} = f(u^n, t_n)\,.$$

This is a *system of algebraic equations*,

$$u^n - \Delta t\, f(u^n, t_n) - u^{n-1} = 0,$$

r written out

$$u_0^n - \Delta t\, f_0(u^n, t_n) - u_0^{n-1} = 0,$$
$$\vdots$$
$$u_N^n - \Delta t\, f_N(u^n, t_n) - u_N^{n-1} = 0\,.$$

As specific example, a 2×2 system for the oscillations of a pendulum subject
› gravity and air drag reads

$$\dot{\omega} = -\sin\theta - \beta\omega|\omega|, \tag{14}$$
$$\dot{\theta} = \omega, \tag{15}$$

here β is a dimensionless parameter (this is the scaled, dimensionless version
f the original, physical model). The unknown components of the system are
ıe angle $\theta(t)$ and the angular velocity $\omega(t)$. We introduce $u_0 = \omega$ and $u_1 = \theta$,
hich leads to

$$u_0' = f_0(u,t) = -\sin u_1 - \beta u_0|u_0|,$$
$$u_1' = f_1(u,t) = u_0\,.$$

. Crank-Nicolson scheme reads

$$\frac{u_0^{n+1} - u_0^n}{\Delta t} = -\sin u_1^{n+\frac{1}{2}} - \beta u_0^{n+\frac{1}{2}}|u_0^{n+\frac{1}{2}}|$$
$$\approx -\sin\left(\frac{1}{2}(u_1^{n+1} + u_1 n)\right) - \beta\frac{1}{4}(u_0^{n+1} + u_0^n)|u_0^{n+1} + u_0^n|, \tag{16}$$
$$\frac{u_1^{n+1} - u_1^n}{\Delta t} = v_0^{n+\frac{1}{2}} \approx \frac{1}{2}(u_0^{n+1} + u_0^n)\,. \tag{17}$$

his is a *coupled system* of two nonlinear algebraic equations in two unknowns
$^{n+1}_0$ and u_1^{n+1}.

Using the notation u_0 and u_1 for the unknowns u_0^{n+1} and u_1^{n+1} in this system,
riting $u_0^{(1)}$ and $u_1^{(1)}$ for the previous values u_0^n and u_1^n, multiplying by Δt and
ıoving the terms to the left-hand sides, gives

$$u_0 - u_0^{(1)} + \Delta t\,\sin\left(\frac{1}{2}(u_1 + u_1^{(1)})\right) + \frac{1}{4}\Delta t\beta(u_0 + u_0^{(1)})|u_0 + u_0^{(1)}| = 0, \tag{18}$$
$$u_1 - u_1^{(1)} - \frac{1}{2}\Delta t(u_0 + u_0^{(1)}) = 0\,. \tag{19}$$

)bviously, we have a need for solving systems of nonlinear algebraic equations,
hich is the topic of the next section.

2 Systems of nonlinear algebraic equations

Implicit time discretization methods for a system of ODEs, or a PDE, *systems* of nonlinear algebraic equations, written compactly as

$$F(u) = 0,$$

where u is a vector of unknowns $u = (u_0, \ldots, u_N)$, and F is a vector f $F = (F_0, \ldots, F_N)$. The system at the end of Section 1.10 fits this notati $N = 2$, $F_0(u)$ given by the left-hand side of (18), while $F_1(u)$ is the le side of (19).

Sometimes the equation system has a special structure because of th lying problem, e.g.,

$$A(u)u = b(u),$$

with $A(u)$ as an $(N+1)\times(N+1)$ matrix function of u and b as a vector f $b = (b_0, \ldots, b_N)$.

We shall next explain how Picard iteration and Newton's method applied to systems like $F(u) = 0$ and $A(u)u = b(u)$. The exposition has on ideas and practical computations. More theoretical considerations, ir quite general results on convergence properties of these methods, can b in Kelley [1].

2.1 Picard iteration

We cannot apply Picard iteration to nonlinear equations unless there special structure. For the commonly arising case $A(u)u = b(u)$ we can l the product $A(u)u$ to $A(u^-)u$ and $b(u)$ as $b(u^-)$. That is, we use tl previously computed approximation in A and b to arrive at a *linear sys* u:

$$A(u^-)u = b(u^-)\,.$$

A relaxed iteration takes the form

$$A(u^-)u^* = b(u^-), \quad u = \omega u^* + (1-\omega)u^-\,.$$

In other words, we solve a system of nonlinear algebraic equations as a s of linear systems.

Algorithm for relaxed Picard iteration.

Given $A(u)u = b(u)$ and an initial guess u^-, iterate until convergenc

1. solve $A(u^-)u^* = b(u^-)$ with respect to u^*
2. $u = \omega u^* + (1-\omega)u^-$

3. $u^- \leftarrow u$

"Until convergence" means that the iteration is stopped when the change in ıe unknown, $||u - u^-||$, or the residual $||A(u)u - b||$, is sufficiently small, see ection 2.3 for more details.

.2 Newton's method

he natural starting point for Newton's method is the general nonlinear vector quation $F(u) = 0$. As for a scalar equation, the idea is to approximate F around known value u^- by a linear function $\hat{F}$, calculated from the first two terms of Taylor expansion of F. In the multi-variate case these two terms become

$$F(u^-) + J(u^-) \cdot (u - u^-),$$

here J is the *Jacobian* of F, defined by

$$J_{i,j} = \frac{\partial F_i}{\partial u_j}\,.$$

o, the original nonlinear system is approximated by

$$\hat{F}(u) = F(u^-) + J(u^-) \cdot (u - u^-) = 0,$$

hich is linear in u and can be solved in a two-step procedure: first solve $\delta u = -F(u^-)$ with respect to the vector δu and then update $u = u^- + \delta u$. A elaxation parameter can easily be incorporated:

$$u = \omega(u^- + \delta u) + (1 - \omega)u^- = u^- + \omega\delta u\,.$$

Algorithm for Newton's method.

Given $F(u) = 0$ and an initial guess u^-, iterate until convergence:

1. solve $J\delta u = -F(u^-)$ with respect to δu
2. $u = u^- + \omega\delta u$
3. $u^- \leftarrow u$

For the special system with structure $A(u)u = b(u)$,

$$F_i = \sum_k A_{i,k}(u)u_k - b_i(u),$$

one gets

$$J_{i,j} = \sum_k \frac{\partial A_{i,k}}{\partial u_j}u_k + A_{i,j} - \frac{\partial b_i}{\partial u_j}\,.$$

We realize that the Jacobian needed in Newton's method consists of A in the Picard iteration plus two additional terms arising from the differe Using the notation $A'(u)$ for $\partial A/\partial u$ (a quantity with three indices: ∂A_i and $b'(u)$ for $\partial b/\partial u$ (a quantity with two indices: $\partial b_i/\partial u_j$), we can w linear system to be solved as

$$(A + A'u + b')\delta u = -Au + b,$$

or

$$(A(u^-) + A'(u^-)u^- + b'(u^-))\delta u = -A(u^-)u^- + b(u^-)\,.$$

Rearranging the terms demonstrates the difference from the system s each Picard iteration:

$$\underbrace{A(u^-)(u^- + \delta u) - b(u^-)}_{\text{Picard system}} + \gamma(A'(u^-)u^- + b'(u^-))\delta u = 0\,.$$

Here we have inserted a parameter γ such that $\gamma = 0$ gives the Picard and $\gamma = 1$ gives the Newton system. Such a parameter can be handy in s to easily switch between the methods.

Combined algorithm for Picard and Newton iteration.

Given $A(u)$, $b(u)$, and an initial guess u^-, iterate until convergence:

1. solve $(A+\gamma(A'(u^-)u^-+b'(u^-)))\delta u = -A(u^-)u^-+b(u^-)$ with re to δu
2. $u = u^- + \omega\delta u$
3. $u^- \leftarrow u$

$\gamma = 1$ gives a Newton method while $\gamma = 0$ corresponds to Picard iterat

2.3 Stopping criteria

Let $||\cdot||$ be the standard Eucledian vector norm. Four termination crit much in use:

- Absolute change in solution: $||u - u^-|| \leq \epsilon_u$

- Relative change in solution: $||u - u^-|| \leq \epsilon_u ||u_0||$, where u_0 denotes the start value of u^- in the iteration
- Absolute residual: $||F(u)|| \leq \epsilon_r$
- Relative residual: $||F(u)|| \leq \epsilon_r ||F(u_0)||$

o prevent divergent iterations to run forever, one terminates the iterations hen the current number of iterations k exceeds a maximum value $k_{\max}$.

The relative criteria are most used since they are not sensitive to the characteristic size of u. Nevertheless, the relative criteria can be misleading when ıe initial start value for the iteration is very close to the solution, since an nnecessary reduction in the error measure is enforced. In such cases the absolute :iteria work better. It is common to combine the absolute and relative measures f the size of the residual, as in

$$||F(u)|| \leq \epsilon_{rr} ||F(u_0)|| + \epsilon_{ra}, \tag{21}$$

here ϵ_{rr} is the tolerance in the relative criterion and ϵ_{ra} is the tolerance in the bsolute criterion. With a very good initial guess for the iteration (typically the olution of a differential equation at the previous time level), the term $||F(u_0)||$ small and ϵ_{ra} is the dominating tolerance. Otherwise, $\epsilon_{rr}||F(u_0)||$ and the elative criterion dominates.

With the change in solution as criterion we can formulate a combined absolute nd relative measure of the change in the solution:

$$||\delta u|| \leq \epsilon_{ur} ||u_0|| + \epsilon_{ua}, \tag{22}$$

The ultimate termination criterion, combining the residual and the change ı solution with a test on the maximum number of iterations allow, can be xpressed as

$$||F(u)|| \leq \epsilon_{rr} ||F(u_0)|| + \epsilon_{ra} \quad \text{or} \quad ||\delta u|| \leq \epsilon_{ur} ||u_0|| + \epsilon_{ua} \quad \text{or} \quad k > k_{\max}\,. \tag{23}$$

.4 Example: A nonlinear ODE model from epidemiology

he simplest model spreading of a disease, such as a flu, takes the form of a × 2 ODE system

$$S' = -\beta SI, \tag{24}$$
$$I' = \beta SI - \nu I, \tag{25}$$

here $S(t)$ is the number of people who can get ill (susceptibles) and $I(t)$ is the umber of people who are ill (infected). The constants $\beta > 0$ and $\nu > 0$ must be iven along with initial conditions $S(0)$ and $I(0)$.

Implicit time discretization. A Crank-Nicolson scheme leads to system of nonlinear algebraic equations in the unknowns S^{n+1} and I^{n+}

$$\frac{S^{n+1} - S^n}{\Delta t} = -\beta [SI]^{n+\frac{1}{2}} \approx -\frac{\beta}{2}(S^n I^n + S^{n+1} I^{n+1}),$$
$$\frac{I^{n+1} - I^n}{\Delta t} = \beta [SI]^{n+\frac{1}{2}} - \nu I^{n+\frac{1}{2}} \approx \frac{\beta}{2}(S^n I^n + S^{n+1} I^{n+1}) - \frac{\nu}{2}(I^n + I$$

Introducing S for S^{n+1}, $S^{(1)}$ for S^n, I for I^{n+1}, $I^{(1)}$ for I^n, we can rew system as

$$F_S(S, I) = S - S^{(1)} + \frac{1}{2}\Delta t \beta (S^{(1)} I^{(1)} + SI) = 0,$$
$$F_I(S, I) = I - I^{(1)} - \frac{1}{2}\Delta t \beta (S^{(1)} I^{(1)} + SI) + \frac{1}{2}\Delta t \nu (I^{(1)} + I) = 0\,.$$

A Picard iteration. We assume that we have approximations S^- an S and I. A way of linearizing the only nonlinear term SI is to write the $F_S = 0$ equation and $S^- I$ in the $F_I = 0$ equation, which also *decou* equations. Solving the resulting linear equations with respect to the un S and I gives

$$S = \frac{S^{(1)} - \frac{1}{2}\Delta t \beta S^{(1)} I^{(1)}}{1 + \frac{1}{2}\Delta t \beta I^-},$$
$$I = \frac{I^{(1)} + \frac{1}{2}\Delta t \beta S^{(1)} I^{(1)}}{1 - \frac{1}{2}\Delta t \beta S^- + \nu}\,.$$

Before a new iteration, we must update $S^- \leftarrow S$ and $I^- \leftarrow I$.

Newton's method. The nonlinear system (28)-(29) can be written as 0 with $F = (F_S, F_I)$ and $u = (S, I)$. The Jacobian becomes

$$J = \begin{pmatrix} \frac{\partial}{\partial S} F_S & \frac{\partial}{\partial I} F_S \\ \frac{\partial}{\partial S} F_I & \frac{\partial}{\partial I} F_I \end{pmatrix} = \begin{pmatrix} 1 + \frac{1}{2}\Delta t \beta I & \frac{1}{2}\Delta t \beta S \\ -\frac{1}{2}\Delta t \beta I & 1 + \frac{1}{2}\Delta t \beta S + \frac{1}{2}\Delta t \nu \end{pmatrix}$$

The Newton system $J(u^-)\delta u = -F(u^-)$ to be solved in each iteration

$$\begin{pmatrix} 1 + \frac{1}{2}\Delta t \beta I^- & \frac{1}{2}\Delta t \beta S^- \\ -\frac{1}{2}\Delta t \beta I^- & 1 - \frac{1}{2}\Delta t \beta S^- + \frac{1}{2}\Delta t \nu \end{pmatrix} \begin{pmatrix} \delta S \\ \delta I \end{pmatrix} =$$
$$\begin{pmatrix} S^- - S^{(1)} + \frac{1}{2}\Delta t \beta (S^{(1)} I^{(1)} + S^- I^-) \\ I^- - I^{(1)} - \frac{1}{2}\Delta t \beta (S^{(1)} I^{(1)} + S^- I^-) + \frac{1}{2}\Delta t \nu (I^{(1)} + I^-) \end{pmatrix}$$

emark. For this particular system of ODEs, explicit time integration methods ork very well. Even a Forward Euler scheme is fine, but the 4-th order Runge-utta method is an excellent balance between high accuracy, high efficiency, and mplicity.

Linearization at the differential equation level

he attention is now turned to nonlinear partial differential equations (PDEs) nd application of the techniques explained above for ODEs. The model problem a nonlinear diffusion equation

$$\frac{\partial u}{\partial t} = \nabla\cdot(\alpha(u)\nabla u) + f(u), \qquad \boldsymbol{x}\in\Omega,\ t\in(0,T], \tag{30}$$

$$-\alpha(u)\frac{\partial u}{\partial n} = g, \qquad \boldsymbol{x}\in\partial\Omega_N,\ t\in(0,T], \tag{31}$$

$$u = u_0, \qquad \boldsymbol{x}\in\partial\Omega_D,\ t\in(0,T]\,. \tag{32}$$

Our aim is to discretize the problem in time and then present techniques or linearizing the time-discrete PDE problem "at the PDE level" such that e transform the nonlinear stationary PDE problems at each time level into a equence of linear PDE problems, which can be solved using any method for near PDEs. This strategy avoids the solution systems of nonlinear algebraic quations. In Section 4 we shall take the opposite (and more common) approach: iscretize the nonlinear problem in time and space first, and then solve the esulting nonlinear algebraic equations at each time level by the methods of ection 2.

.1 Explicit time integration

he nonlinearities in the PDE are trivial to deal with if we choose an explicit me integration method for (30), such as the Forward Euler method:

$$[D_t^+ u = \nabla\cdot(\alpha(u)\nabla u) + f(u)]^n,$$

r written out,

$$\frac{u^{n+1}-u^n}{\Delta t} = \nabla\cdot(\alpha(u^n)\nabla u^n) + f(u^n),$$

hich is a linear equation in the unknown u^{n+1} with solution

$$u^{n+1} = u^n + \Delta t\nabla\cdot(\alpha(u^n)\nabla u^n) + \Delta t f(u^n)\,.$$

The disadvantage with this discretization is usually thought to be the stability iterion

$$\Delta t \leq \frac{1}{\max\alpha}(\Delta x^2 + \Delta y^2 + \Delta z^2),$$

for the case $f = 0$ and a standard 2nd-order finite difference discretiz space with mesh cell sizes Δx, Δy, and Δz in the various spatial direct

3.2 Backward Euler scheme and Picard iteration

A Backward Euler scheme for (30) reads

$$[D_t^- u = \nabla\cdot(\alpha(u)\nabla u) + f(u)]^n\,.$$

Written out,

$$\frac{u^n - u^{n-1}}{\Delta t} = \nabla\cdot(\alpha(u^n)\nabla u^n) + f(u^n)\,.$$

This is a nonlinear PDE for the unknown function $u^n(\boldsymbol{x})$. Such a PDE viewed as a time-independent PDE where $u^{n-1}(\boldsymbol{x})$ is a known function

We introduce a Picard iteration with k as iteration counter. A linearization of the $\nabla\cdot\alpha(u^n)\nabla u^n$ term in iteration $k+1$ is to use the pr computed $u^{n,k}$ approximation in the diffusion coefficient: $\alpha(u^{n,k})$. The n source term is treated similarly: $f(u^{n,k})$. The unknown function $u^{n,k}$ fulfills the linear PDE

$$\frac{u^{n,k+1} - u^{n-1}}{\Delta t} = \nabla\cdot(\alpha(u^{n,k})\nabla u^{n,k+1}) + f(u^{n,k})\,.$$

The initial guess for the Picard iteration at this time level can be take solution at the previous time level: $u^{n,0} = u^{n-1}$.

We can alternatively apply the implementation-friendly notation corresponds to the unknown we want to solve for, i.e., $u^{n,k+1}$ above, is the most recently computed value, $u^{n,k}$ above. Moreover, $u^{(1)}$ den unknown function at the previous time level, u^{n-1} above. The PDE to b in a Picard iteration then looks like

$$\frac{u - u^{(1)}}{\Delta t} = \nabla\cdot(\alpha(u^-)\nabla u) + f(u^-)\,.$$

At the beginning of the iteration we start with the value from the previc level: $u^- = u^{(1)}$, and after each iteration, u^- is updated to u.

Remark on notation.

The previous derivations of the numerical scheme for time discretiza of PDEs have, strictly speaking, somewhat sloppy notation. A precise notation must distinguish clearly between the exact soluti the PDE problem, here denoted $u_{\mathrm{e}}(\boldsymbol{x},t)$, and the exact solution o spatial problem, arising after time discretization at each time level, (33) is an example. The latter is here represented as $u^n(\boldsymbol{x},t)$ and approximation to $u_{\mathrm{e}}(\boldsymbol{x},t_n)$. Then we have another approximation $u^$ to $u^n(\boldsymbol{x})$ when solving the nonlinear PDE problem for u^n by iter

methods, as in (34). Alternatively, we introduce $u(\boldsymbol{x})$ as a synonym for $u^{n,k}(\boldsymbol{x})$ (and also $u^{(1)}(\boldsymbol{x})$) as in (35).

However, we will usually state the PDE problem in terms of u and quickly redefine the symbol u to mean the numerical approximation, while u_{e} is not explicitly introduced unless we need to talk about the exact solution and the approximate solution at the same time.

.3 Backward Euler scheme and Newton's method

t time level n we have to solve the stationary PDE (33), this time with Newton's ıethod. Normally, Newton's method is defined for systems of *algebraic equations*, ut the idea of the method can be applied at the PDE level too.

inearization via Taylor expansions. Let $u^{n,k}$ be an approximation to the nknown u^n. We seek a better approximation on the form

$$u^n = u^{n,k} + \delta u\,. \tag{36}$$

he idea is to insert (36) in (33), Taylor expand the nonlinearities and keep nly the terms that are linear in δu. Then we can solve a linear PDE for the orrection δu and use (36) to find a new approximation $u^{n,k+1} = u^{n,k} + \delta u$ to ^{n}.

Inserting (36) in (33) gives

$$\frac{u^{n,k}+\delta u - u^{n-1}}{\Delta t} = \nabla\cdot(\alpha(u^{n,k}+\delta u)\nabla(u^{n,k}+\delta u)) + f(u^{n,k}+\delta u)\,. \tag{37}$$

/e can Taylor expand $\alpha(u^{n,k}+\delta u)$ and $f(u^{n,k}+\delta u)$:

$$\alpha(u^{n,k}+\delta u) = \alpha(u^{n,k}) + \frac{d\alpha}{du}(u^{n,k})\delta u + \mathcal{O}(\delta u^2) \approx \alpha(u^{n,k}) + \alpha'(u^{n,k})\delta u,$$
$$f(u^{n,k}+\delta u) = f(u^{n,k}) + \frac{df}{du}(u^{n,k})\delta u + \mathcal{O}(\delta u^2) \approx f(u^{n,k}) + f'(u^{n,k})\delta u\,.$$

ıserting the linear approximations of α and f in (37) results in

$$\begin{aligned}\frac{u^{n,k}+\delta u - u^{n-1}}{\Delta t} &= \nabla\cdot(\alpha(u^{n,k})\nabla u^{n,k}) + f(u^{n,k}) + \\ &\quad \nabla\cdot(\alpha(u^{n,k})\nabla\delta u) + \nabla\cdot(\alpha'(u^{n,k})\delta u\nabla u^{n,k}) + \\ &\quad \nabla\cdot(\alpha'(u^{n,k})\delta u\nabla\delta u) + f'(u^{n,k})\delta u\,.\end{aligned} \tag{38}$$

he term $\alpha'(u^{n,k})\delta u\nabla\delta u$ is $\mathcal{O}(\delta u^2)$ and therefore omitted. Reorganizing the quation gives a PDE for δu that we can write in short form as

$$\delta F(\delta u; u^{n,k}) = -F(u^{n,k}),$$

where

$$\begin{aligned}F(u^{n,k}) &= \frac{u^{n,k}-u^{n-1}}{\Delta t} - \nabla\cdot(\alpha(u^{n,k})\nabla u^{n,k}) + f(u^{n,k}),\\ \delta F(\delta u; u^{n,k}) &= -\frac{1}{\Delta t}\delta u + \nabla\cdot(\alpha(u^{n,k})\nabla\delta u) + \\ &\quad \nabla\cdot(\alpha'(u^{n,k})\delta u\nabla u^{n,k}) + f'(u^{n,k})\delta u\,.\end{aligned}$$

Note that δF is a linear function of δu, and F contains only terms t known, such that the PDE for δu is indeed linear.

Observations.

The notational form $\delta F = -F$ resembles the Newton system $J\delta u = -$ systems of algebraic equations, with δF as $J\delta u$. The unknown vecto linear system of algebraic equations enters the system as a linear ope in terms of a matrix-vector product ($J\delta u$), while at the PDE level we a linear differential operator instead (δF).

Similarity with Picard iteration. We can rewrite the PDE for slightly different way too if we define $u^{n,k}+\delta u$ as $u^{n,k+1}$.

$$\begin{aligned}\frac{u^{n,k+1}-u^{n-1}}{\Delta t} &= \nabla\cdot(\alpha(u^{n,k})\nabla u^{n,k+1}) + f(u^{n,k}) \\ &\quad + \nabla\cdot(\alpha'(u^{n,k})\delta u\nabla u^{n,k}) + f'(u^{n,k})\delta u\,.\end{aligned}$$

Note that the first line is the same PDE as arise in the Picard iteration, w remaining terms arise from the differentiations that are an inherent in in Newton's method.

Implementation. For coding we want to introduce u for u^n, u^- for u $u^{(1)}$ for u^{n-1}. The formulas for F and δF are then more clearly writte

$$\begin{aligned}F(u^-) &= \frac{u^- - u^{(1)}}{\Delta t} - \nabla\cdot(\alpha(u^-)\nabla u^-) + f(u^-),\\ \delta F(\delta u; u^-) &= -\frac{1}{\Delta t}\delta u + \nabla\cdot(\alpha(u^-)\nabla\delta u) + \\ &\quad \nabla\cdot(\alpha'(u^-)\delta u\nabla u^-) + f'(u^-)\delta u\,.\end{aligned}$$

he form that orders the PDE as the Picard iteration terms plus the Newton ıethod's derivative terms becomes

$$\frac{u-u^{(1)}}{\Delta t} = \nabla\cdot(\alpha(u^-)\nabla u) + f(u^-) + \gamma(\nabla\cdot(\alpha'(u^-)(u-u^-)\nabla u^-) + f'(u^-)(u-u^-))\,. \quad (44)$$

he Picard and full Newton versions correspond to $\gamma=0$ and $\gamma=1$, respectively.

)erivation with alternative notation. Some may prefer to derive the lin-ırized PDE for δu using the more compact notation. We start with inserting $^n = u^- + \delta u$ to get

$$\frac{u^- + \delta u - u^{n-1}}{\Delta t} = \nabla\cdot(\alpha(u^- + \delta u)\nabla(u^- + \delta u)) + f(u^- + \delta u)\,.$$

aylor expanding,

$$\alpha(u^- + \delta u) \approx \alpha(u^-) + \alpha'(u^-)\delta u,$$
$$f(u^- + \delta u) \approx f(u^-) + f'(u^-)\delta u,$$

nd inserting these expressions gives a less cluttered PDE for δu:

$$\frac{u^- + \delta u - u^{n-1}}{\Delta t} = \nabla\cdot(\alpha(u^-)\nabla u^-) + f(u^-) + \nabla\cdot(\alpha(u^-)\nabla\delta u) + \nabla\cdot(\alpha'(u^-)\delta u\nabla u^-) + \nabla\cdot(\alpha'(u^-)\delta u\nabla\delta u) + f'(u^-)\delta u\,.$$

.4 Crank-Nicolson discretization

Crank-Nicolson discretization of (30) applies a centered difference at $t_{n+\frac{1}{2}}$:

$$[D_t u = \nabla\cdot(\alpha(u)\nabla u) + f(u)]^{n+\frac{1}{2}}\,.$$

ince u is not known at $t_{n+\frac{1}{2}}$ we need to express the terms on the right-hand side ia unknowns u^n and u^{n+1}. The standard technique is to apply an arithmetic verage,

$$u^{n+\frac{1}{2}} \approx \frac{1}{2}(u^n + u^{n+1})\,.$$

owever, with nonlinear terms we have many choices of formulating an arithmetic ıean:

$$[f(u)]^{n+\frac{1}{2}} \approx f(\frac{1}{2}(u^n + u^{n+1})) = [f(\overline{u}^t)]^{n+\frac{1}{2}},$$
$$[f(u)]^{n+\frac{1}{2}} \approx \frac{1}{2}(f(u^n) + f(u^{n+1})) = [\overline{f(u)}^t]^{n+\frac{1}{2}},$$
$$[\alpha(u)\nabla u]^{n+\frac{1}{2}} \approx \alpha(\frac{1}{2}(u^n + u^{n+1}))\nabla(\frac{1}{2}(u^n + u^{n+1})) = [\alpha(\overline{u}^t)\nabla\overline{u}^t]^{n+\frac{1}{2}},$$
$$[\alpha(u)\nabla u]^{n+\frac{1}{2}} \approx \frac{1}{2}(\alpha(u^n) + \alpha(u^{n+1}))\nabla(\frac{1}{2}(u^n + u^{n+1})) = [\overline{\alpha(u)}^t\nabla\overline{u}^t]^{n+}$$
$$[\alpha(u)\nabla u]^{n+\frac{1}{2}} \approx \frac{1}{2}(\alpha(u^n)\nabla u^n + \alpha(u^{n+1})\nabla u^{n+1}) = [\overline{\alpha(u)\nabla u}^t]^{n+\frac{1}{2}}\,.$$

A big question is whether there are significant differences in accuracy taking the products of arithmetic means or taking the arithmetic r products. Exercise 5 investigates this question, and the answer is t approximation is $\mathcal{O}(\Delta t^2)$ in both cases.

4 Discretization of 1D stationary nonlinea ferential equations

Section 3 presents methods for linearizing time-discrete PDEs directly discretization in space. We can alternatively carry out the discretization and of the time-discrete nonlinear PDE problem and get a system of n algebraic equations, which can be solved by Picard iteration or Newton's as presented in Section 2. This latter approach will now be described i

We shall work with the 1D problem

$$-(\alpha(u)u')' + au = f(u), \quad x\in(0,L), \quad \alpha(u(0))u'(0) = C,\ u(L) = L$$

The problem (50) arises from the stationary limit of a diffusion equ

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(\alpha(u)\frac{\partial u}{\partial x}\right) + au + f(u),$$

as $t\to\infty$ and $\partial u/\partial t\to 0$. Alternatively, the problem (50) arises at ea level from implicit time discretization of (51). For example, a Backwar scheme for (51) with $a=0$ leads to

$$\frac{u^n - u^{n-1}}{\Delta t} = \frac{\partial}{\partial x}\left(\alpha(u^n)\frac{\partial u^n}{\partial x}\right) - f(u^n)\,.$$

Introducing $u(x)$ for $u^n(x)$, $u^{(1)}$ for u^{n-1}, and letting $f(u)$ in (50) be $u^{n-1}/\Delta t$ in (52), gives (50) with $a = 1/\Delta t$.

.1 Finite difference discretizations

he nonlinearity in the differential equation (50) poses no more difficulty than a ariable coefficient, as in $(\alpha(x)u')'$. We can therefore use a standard approach o discretizing the Laplace term with a variable coefficient:

$$[-D_x\alpha D_x u + au = f]_i\,.$$

/riting this out for a uniform mesh with points $x_i = i\Delta x$, $i = 0, \ldots, N_x$, leads o

$$-\frac{1}{\Delta x^2}\left(\alpha_{i+\frac{1}{2}}(u_{i+1} - u_i) - \alpha_{i-\frac{1}{2}}(u_i - u_{i-1})\right) + au_i = f(u_i)\,. \tag{53}$$

his equation is valid at all the mesh points $i = 0, 1, \ldots, N_x - 1$. At $i = N_x$ e have the Dirichlet condition $u_i = 0$. The only difference from the case with $(x)u')'$ and $f(x)$ is that now α and f are functions of u and not only on x: $(u(x))u')'$ and $f(u(x))$.

The quantity $\alpha_{i+\frac{1}{2}}$, evaluated between two mesh points, needs a comment. ince α depends on u and u is only known at the mesh points, we need to express $_{i+\frac{1}{2}}$ in terms of u_i and u_{i+1}. For this purpose we use an arithmetic mean, lthough a harmonic mean is also common in this context if α features large umps. There are two choices of arithmetic means:

$$\alpha_{i+\frac{1}{2}} \approx \alpha(\frac{1}{2}(u_i + u_{i+1}) = [\alpha(\overline{u}^x)]^{i+\frac{1}{2}}, \tag{54}$$

$$\alpha_{i+\frac{1}{2}} \approx \frac{1}{2}(\alpha(u_i) + \alpha(u_{i+1})) = [\overline{\alpha(u)}^x]^{i+\frac{1}{2}} \tag{55}$$

quation (53) with the latter approximation then looks like

$$-\frac{1}{2\Delta x^2}\left((\alpha(u_i) + \alpha(u_{i+1}))(u_{i+1} - u_i) - (\alpha(u_{i-1}) + \alpha(u_i))(u_i - u_{i-1})\right) + au_i = f(u_i), \tag{56}$$

r written more compactly,

$$[-D_x\overline{\alpha}^x D_x u + au = f]_i\,.$$

At mesh point $i = 0$ we have the boundary condition $\alpha(u)u' = C$, which is iscretized by

$$[\alpha(u)D_{2x}u = C]_0,$$

ıeaning

$$\alpha(u_0)\frac{u_1 - u_{-1}}{2\Delta x} = C\,. \tag{57}$$

The fictitious value u_{-1} can be eliminated with the aid of (56) for $i = 0$. F (56) should be solved with respect to u_{i-1} and that value (for $i = 0$) sh inserted in (57), but it is algebraically much easier to do it the other way Alternatively, one can use a ghost cell $[-\Delta x, 0]$ and update the u_{-1} the ghost cell according to (57) after every Picard or Newton iteratio an approach means that we use a known u_{-1} value in (56) from the p iteration.

4.2 Solution of algebraic equations

The structure of the equation system. The nonlinear algebraic ec (56) are of the form $A(u)u = b(u)$ with

$$\begin{aligned} A_{i,i} &= \frac{1}{2\Delta x^2}(\alpha(u_{i-1}) + 2\alpha(u_i)\alpha(u_{i+1})) + a,\\ A_{i,i-1} &= -\frac{1}{2\Delta x^2}(\alpha(u_{i-1}) + \alpha(u_i)),\\ A_{i,i+1} &= -\frac{1}{2\Delta x^2}(\alpha(u_i) + \alpha(u_{i+1})),\\ b_i &= f(u_i)\,. \end{aligned}$$

The matrix $A(u)$ is tridiagonal: $A_{i,j} = 0$ for $j > 1 + 1$ and $j < i - 1$.

The above expressions are valid for internal mesh points $1 \leq i \leq N_x$ $i = 0$ we need to express $u_{i-1} = u_{-1}$ in terms of u_1 using (57):

$$u_{-1} = u_1 - \frac{2\Delta x}{\alpha(u_0)}C\,.$$

This value must be inserted in $A_{0,0}$. The expression for $A_{i,i+1}$ applies f and $A_{i,i-1}$ does not enter the system when $i = 0$.

Regarding the last equation, its form depends on whether we incl Dirichlet condition $u(L) = D$, meaning $u_{N_x} = D$, in the nonlinear a equation system or not. Suppose we choose $(u_0, u_1, \ldots, u_{N_x-1})$ as un later referred to as *systems without Dirichlet conditions*. The last equatic sponds to $i = N_x - 1$. It involves the boundary value u_{N_x}, which is substit D. If the unknown vector includes the boundary value, $(u_0, u_1, \ldots, u_{N_x}$ referred to as *system including Dirichlet conditions*, the equation for $i =$ just involves the unknown u_{N_x}, and the final equation becomes u_N corresponding to $A_{i,i} = 1$ and $b_i = D$ for $i = N_x$.

Picard iteration. The obvious Picard iteration scheme is to use pr computed values of u_i in $A(u)$ and $b(u)$, as described more in detail in S With the notation u^- for the most recently computed value of u, the system $F(u) \approx \hat{F}(u) = A(u^-)u - b(u^-)$, with $F = (F_0, F_1, \ldots, F_$ $(u_0, u_1, \ldots, u_m)$. The index m is N_x if the system includes the Dirichlet c as a separate equation and $N_x - 1$ otherwise. The matrix $A(u^-)$ is trid

› the solution procedure is to fill a tridiagonal matrix data structure and the ght-hand side vector with the right numbers and call a Gaussian elimination ›utine for tridiagonal linear systems.

To write out all the mathematical details in a specific case, let us look at ıe case $N_x = 2$. We use u_i^- for the i-th component in u^-. In case we omit the ›irichlet condition from the system we get the following 2×2 system,

$$\begin{pmatrix} A_{0,0} & A_{0,1} \\ A_{1,0} & A_{1,1} \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \end{pmatrix}$$

he matrix and right-hand side entries are given by

$$A_{0,0} = \frac{1}{2\Delta x^2}(\alpha(u_{-1}^-) + 2\alpha(u_0^-) + \alpha(u_1^-)) + a \tag{59}$$

$$A_{0,1} = -\frac{1}{2\Delta x^2}(\alpha(u_0^-) + \alpha(u_1^-)), \tag{60}$$

$$A_{1,0} = -\frac{1}{2\Delta x^2}(\alpha(u_0^-) + \alpha(u_1^-)), \tag{61}$$

$$A_{1,1} = \frac{1}{2\Delta x^2}(\alpha(u_0^-) + 2\alpha(u_1^-) + \alpha(u_2)) + a, \tag{62}$$

$$b_0 = f(u_0^-), \tag{63}$$

$$b_1 = f(u_1^-), \tag{64}$$

here u_{-1} must be substituted by (58), and u_2 by D.

The system with the Dirichlet condition becomes

$$\begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} \\ A_{1,0} & A_{1,1} & A_{1,2} \\ A_{2,0} & A_{2,1} & A_{2,2} \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix},$$

ith entries for $A_{i,j}$ and b_i as above for $i, j = 1, 2$, keeping u_2 as unknown in ·1,1, and

$$A_{0,2} = A_{2,0} = A_{2,1} = 0,\ A_{1,2} = -\frac{1}{2\Delta x^2}(\alpha(u_1) + \alpha(u_2)),\ A_{2,2} = 1,\ b_2 = D\,. \tag{65}$$

Newton's method. The Jacobian must be derived in order to use Newton's ıethod. Here it means that we need to differentiate $F(u) = A(u)u - b(u)$ with ›spect to the unknown parameters $u_0, u_1, \ldots, u_m$ ($m = N_x$ or $m = N_x - 1$, epending on whether the Dirichlet condition is included in the nonlinear system '$(u) = 0$ or not). Nonlinear equation number i has the structure

$$_i = A_{i,i-1}(u_{i-1}, u_i)u_{i-1} + A_{i,i}(u_{i-1}, u_i, u_{i+1})u_i + A_{i,i+1}(u_i, u_{i+1})u_{i+1} - b_i(u_i)\,.$$

Computing the Jacobian requires careful differentiation. For example,

$$\begin{aligned}\frac{\partial}{\partial u_i}(A_{i,i}(u_{i-1}, u_i, u_{i+1})u_i) &= \frac{\partial A_{i,i}}{\partial u_i}u_i + A_{i,i}\frac{\partial u_i}{\partial u_i} \\ &= \frac{\partial}{\partial u_i}(\frac{1}{2\Delta x^2}(\alpha(u_{i-1}) + 2\alpha(u_i) + \alpha(u_{i+1})) + \\ &\quad \frac{1}{2\Delta x^2}(\alpha(u_{i-1}) + 2\alpha(u_i) + \alpha(u_{i+1})) + a \\ &= \frac{1}{2\Delta x^2}(2\alpha'(u_i)u_i + \alpha(u_{i-1}) + 2\alpha(u_i) + \alpha(u\end{aligned}$$

The complete Jacobian becomes

$$\begin{aligned}J_{i,i} = \frac{\partial F_i}{\partial u_i} &= \frac{\partial A_{i,i-1}}{\partial u_i}u_{i-1} + \frac{\partial A_{i,i}}{\partial u_i}u_i + A_{i,i} + \frac{\partial A_{i,i+1}}{\partial u_i}u_{i+1} - \frac{\partial b_i}{\partial u_i} \\ &= \frac{1}{2\Delta x^2}(-\alpha'(u_i)u_{i-1} + 2\alpha'(u_i)u_i + \alpha(u_{i-1}) + 2\alpha(u_i) + \alpha(u_{i+} \\ &\quad a - \frac{1}{2\Delta x^2}\alpha'(u_i)u_{i+1} - b'(u_i), \\ J_{i,i-1} = \frac{\partial F_i}{\partial u_{i-1}} &= \frac{\partial A_{i,i-1}}{\partial u_{i-1}}u_{i-1} + A_{i-1,i} + \frac{\partial A_{i,i}}{\partial u_{i-1}}u_i - \frac{\partial b_i}{\partial u_{i-1}} \\ &= \frac{1}{2\Delta x^2}(-\alpha'(u_{i-1})u_{i-1} - (\alpha(u_{i-1}) + \alpha(u_i)) + \alpha'(u_{i-1})u_i), \\ J_{i,i+1} = \frac{\partial A_{i,i+1}}{\partial u_{i-1}}u_{i+1} &+ A_{i+1,i} + \frac{\partial A_{i,i}}{\partial u_{i+1}}u_i - \frac{\partial b_i}{\partial u_{i+1}} \\ &= \frac{1}{2\Delta x^2}(-\alpha'(u_{i+1})u_{i+1} - (\alpha(u_i) + \alpha(u_{i+1})) + \alpha'(u_{i+1})u_i)\,.\end{aligned}$$

The explicit expression for nonlinear equation number i, $F_i(u_0, u_1, \ldots$ from moving the (u_i) term in (56) to the left-hand side:

$$\begin{aligned}F_i = -\frac{1}{2\Delta x^2}&((\alpha(u_i) + \alpha(u_{i+1}))(u_{i+1} - u_i) - (\alpha(u_{i-1}) + \alpha(u_i))(u_i - \\ &+ au_i - f(u_i) = 0\,.\end{aligned}$$

At the boundary point $i = 0$, u_{-1} must be replaced using the formu When the Dirichlet condition at $i = N_x$ is not a part of the equation the last equation $F_m = 0$ for $m = N_x - 1$ involves the quantity u_{N_x}– must be replaced by D. If u_{N_x} is treated as an unknown in the system, equation $F_m = 0$ has $m = N_x$ and reads

$$F_{N_x}(u_0, \ldots, u_{N_x}) = u_{N_x} - D = 0\,.$$

Similar replacement of u_{-1} and u_{N_x} must be done in the Jacobian for and last row. When u_{N_x} is included as an unknown, the last row in the J must help implement the condition $\delta u_{N_x} = 0$, since we assume that u ‹

ıe right Dirichlet value at the beginning of the iteration ($u_{N_x} = D$), and then ıe Newton update should be zero for $i = 0$, i.e., $\delta u_{N_x} = 0$. This also forces the ght-hand side to be $b_i = 0$, $i = N_x$.

We have seen, and can see from the present example, that the linear system ı Newton's method contains all the terms present in the system that arises ı the Picard iteration method. The extra terms in Newton's method can be ıultiplied by a factor such that it is easy to program one linear system and set ıis factor to 0 or 1 to generate the Picard or Newton system.

.3 Galerkin-type discretizations

or a Galerkin-type discretization, which may be developed into a finite element ıethod, we first need to derive the variational problem. Let V be an appropriate ınction space with basis functions $\{\psi_i\}_{i\in\mathcal{I}_s}$. Because of the Dirichlet condition t $x = L$ we require $\psi_i(L) = 0$, $i \in \mathcal{I}_s$. The approximate solution is written as $= D + \sum_{j\in\mathcal{I}_s} c_j\psi_j$, where the term D can be viewed as a boundary function eeded to implement the Dirichlet condition $u(L) = D$.

Using Galerkin's method, we multiply the differential equation by any $v \in V$ nd integrate terms with second-order derivatives by parts:

$$\int_0^L \alpha(u)u'v'\,\mathrm{d}x + \int_0^L auv\,\mathrm{d}x = \int_0^L f(u)v\,\mathrm{d}x + [\alpha(u)u'v]_0^L, \quad \forall v \in V\,.$$

he Neumann condition at the boundary $x = 0$ is inserted in the boundary term:

$$[\alpha(u)u'v]_0^L = \alpha(u(L))u'(L)v(L) - \alpha(u(0))u'(0)v(0) = 0 - Cv(0) = -Cv(0)\,.$$

Recall that since $\psi_i(L) = 0$, any linear combination v of the basis functions also anishes at $x = L$: $v(L) = 0$.) The variational problem is then: find $u \in V$ such nat

$$\int_0^L \alpha(u)u'v'\,\mathrm{d}x + \int_0^L auv\,\mathrm{d}x = \int_0^L f(u)v\,\mathrm{d}x - Cv(0), \quad \forall v \in V\,. \tag{67}$$

To derive the algebraic equations, we note that $\forall v \in V$ is equivalent with $= \psi_i$ for $i \in \mathcal{I}_s$. Setting $u = D + \sum_j c_j\psi_j$ and sorting terms results in the near system

$$\sum_{\in\mathcal{I}_s}\left(\int_0^L \alpha(D + \sum_{k\in\mathcal{I}_s} c_k\psi_k)\psi_j'\psi_i'\,\mathrm{d}x\right)c_j = \int_0^L f(D+\sum_{k\in\mathcal{I}_s} c_k\psi_k)\psi_i\,\mathrm{d}x - C\psi_i(0), \quad i \in \tag{68}$$

Fundamental integration problem. Methods that use the Gale weighted residual principle face a fundamental difficulty in nonlinear pr how can we integrate a terms like $\int_0^L \alpha(\sum_k c_k\psi_k)\psi_i'\psi_j'\,\mathrm{d}x$ and $\int_0^L f(\sum_k c_k$ when we do not know the c_k coefficients in the argument of the α functi can resort to numerical integration, provided an approximate $\sum_k c_k\psi_k$ used for the argument u in f and α. This is the approach used in cc programs.

However, if we want to look more mathematically into the structur algebraic equations generated by the finite element method in nonlinear p and compare such equations with those arising in the finite difference met need techniques that enable integration of expressions like $\int_0^L f(\sum_k c_k\psi$ *by hand*. Two such techniques will be shown: the group finite elem numerical integration based on the nodes only. Both techniques are appr but they allow us to see the difference equations in the finite element n

4.4 Finite element basis functions

Introduction of finite element basis functions φ_i means setting

$$\psi_i = \varphi_{\nu(i)}, \quad i \in \mathcal{I}_s,$$

where degree of freedom number $\nu(i)$ in the mesh corresponds to u number i (c_i). In the present example, we use all the basis functions ex last at $i = N_n - 1$, i.e., $\mathcal{I}_s = \{0, \ldots, N_n - 2\}$, and $\nu(j) = j$. The expans can be taken as

$$u = D + \sum_{j\in\mathcal{I}_s} c_j\varphi_{\nu(j)},$$

but it is more common in a finite element context to use a boundary f $B = \sum_{j\in I_b} U_j\varphi_j$, where U_j are prescribed Dirichlet conditions for d freedom number j and U_j is the corresponding value.

$$u = D\varphi_{N_n-1} + \sum_{j\in\mathcal{I}_s} c_j\varphi_{\nu(j)}\,.$$

In the general case with u prescribed as U_j at some nodes $j \in I_b$, we se

$$u = \sum_{j\in I_b} U_j\varphi_j + \sum_{j\in\mathcal{I}_s} c_j\varphi_{\nu(j)},$$

where $c_j = u(x^{\nu(j)})$. That is, $\nu(j)$ maps unknown number j to the corres node number $\nu(j)$ such that $c_j = u(x^{\nu(j)})$.

4.5 The group finite element method

Finite element approximation of functions of u. Since we already u as $\sum_j \varphi_j u(x_j)$, we may use the same approximation for other func well. For example,

$$f(u) \approx \sum_j f(x_j)\varphi_j,$$

here $f(x_j)$ is the value of f at node j. Since f is a function of u, $f(x_j) = (u(x_j))$. Introducing u_j as a short form for $u(x_j)$, we can write

$$f(u) \approx \sum_j f(u_j)\varphi_j\,.$$

his approximation is known as the *group finite element method* or the *product pproximation* technique. The index j runs over all node numbers in the mesh.

The principal advantages of the group finite element method are two-fold:

1. Complicated nonlinear expressions can be simplified to increase the efficiency of numerical computations.
2. One can derive *symbolic forms* of the difference equations arising from the finite element method in nonlinear problems. The symbolic form is useful for comparing finite element and finite difference equations of nonlinear differential equation problems.

elow, we shall explore point 2 to see exactly how the finite element method reates more complex expressions in the resulting linear system (the difference quations) that the finite difference method does. It turns out that is very difficult o see what kind of turns in the difference equations that arise from $\int f(u)\varphi_i\,\mathrm{d}x$ ithout using the group finite element method or numerical integration utilizing ne nodes only.

Note, however, that an expression like $\int f(u)\varphi_i\,\mathrm{d}x$ causes no problems in a omputer program as the integral is calculated by numerical integration using n existing approximation of u in $f(u)$ such that the integrand can be sampled t any spatial point.

implified problem. Our aim now is the derive symbolic expressions for the ifference equations arising from the finite element method in nonlinear problems nd compare the expressions with those arising in the finite difference method. o this, let us simplify the model problem and set $a = 0$, $\alpha = 1$, $f(u) = u^2$, and ave Neumann conditions at both ends such that we get a very simple nonlinear roblem $-u'' = u^2$, $u'(0) = 1$, $u'(L) = 0$. The variational form is then

$$\int_0^L u'v'\,\mathrm{d}x = \int_0^L u^2 v\,\mathrm{d}x - v(0), \quad \forall v \in V\,.$$

he term with $u'v'$ is well known so the only new feature is the term $\int u^2 v\,\mathrm{d}x$.

To make the distance from finite element equations to finite difference equaons as short as possible, we shall substitute c_j in the sum $u = \sum_j c_j\varphi_j$ by $_j = u(x_j)$ since c_j is the value of u at node j. (In the more general case with irichlet conditions as well, we have a sum $\sum_j c_j\varphi_{\nu(j)}$ where c_j is replaced by

$u(x_{\nu(j)})$. We can then introduce some other counter k such that it is me to write $u = \sum_k u_k\varphi_k$, where k runs over appropriate node numbers quantity u_j in $\sum_j u_j\varphi_j$ is the same as u at mesh point number j in tl difference method, which is commonly denoted u_j.

Integrating nonlinear functions. Consider the term $\int u^2 v\,\mathrm{d}x$ in tl tional formulation with $v = \varphi_i$ and $u = \sum_k \varphi_k u_k$:

$$\int_0^L (\sum_k u_k\varphi_k)^2\varphi_i\,\mathrm{d}x\,.$$

Evaluating this integral for P1 elements (see Problem 10) results in the ex

$$\frac{h}{12}(u_{i-1}^2 + 2u_i(u_{i-1} + u_{i+1}) + 6u_i^2 + u_{i+1}^2),$$

to be compared with the simple value u_i^2 that would arise in a finite di discretization when u^2 is sampled at mesh point x_i. More complicat functions give rise to much more lengthy expressions, if it is possible t out the integral symbolically at all.

Application of the group finite element method. Let use the gro element method to derive the terms in the difference equation correspo $f(u)$ in the differential equation. We have

$$\int_0^L f(u)\varphi_i\,\mathrm{d}x \approx \int_0^L (\sum_j \varphi_j f(u_j))\varphi_i\,\mathrm{d}x = \sum_j \left(\int_0^L \varphi_i\varphi_j\,\mathrm{d}x\right) f(u_j$$

We recognize this expression as the mass matrix M, arising from $\int$ times the vector $f = (f(u_0), f(u_1), \ldots,)$: Mf. The associated term difference equations are, for P1 elements,

$$\frac{h}{6}(f(u_{i-1}) + 4f(u_i) + f(u_{i+1}))\,.$$

Occasionally, we want to interpret this expression in terms of finite diff and to this end a rewrite of this expression is convenient:

$$\frac{h}{6}(f(u_{i-1}) + 4f(u_i) + f(u_{i+1})) = h[f(u) - \frac{h^2}{6}D_xD_xf(u)]_i\,.$$

That is, the finite element treatment of $f(u)$ (when using a group finite method) gives the same term as in a finite difference approach, $f(u_i)$, diffusion term which is the 2nd-order discretization of $\frac{1}{6}h^2 f''(x_i)$.

We may lump the mass matrix through integration with the Trapezoi so that M becomes diagonal in the finite element method. In that case t term in the differential equation gives rise to a single term $hf(u_i)$, just a finite difference method.

.6 Numerical integration of nonlinear terms

et us reconsider a term $\int f(u)v\,\mathrm{d}x$ as treated in the previous section, but ow we want to integrate this term numerically. Such an approach can lead to asy-to-interpret formulas if we apply a numerical integration rule that samples ıe integrand at the node points x_i only, because at such points, $\varphi_j(x_i) = 0$ if $\neq i$, which leads to great simplifications.

The term in question takes the form

$$\int_0^L f(\sum_k u_k\varphi_k)\varphi_i\,\mathrm{d}x\,.$$

valuation of the integrand at a node x_ℓ leads to a collapse of the sum $\sum_k u_k\varphi_k$ o one term because

$$\sum_k u_k\varphi_k(x_\ell) = u_\ell\,.$$

$$f(\sum_k u_k \underbrace{\varphi_k(x_\ell)}_{\delta_{k\ell}})\underbrace{\varphi_i(x_\ell)}_{\delta_{i\ell}} = f(u_\ell)\delta_{i\ell},$$

here we have used the Kronecker delta: $\delta_{ij} = 0$ if $i \neq j$ and $\delta_{ij} = 1$ if $i = j$.

Considering the Trapezoidal rule for integration, where the integration points re the nodes, we have

$$\int_0^L f(\sum_k u_k\varphi_k)(x)\varphi_i(x)\,\mathrm{d}x \approx h\sum_{\ell=0}^{N_n} f(u_\ell)\delta_{i\ell} - \mathcal{C} = hf(u_i)\,.$$

his is the same representation of the f term as in the finite difference method. he term $\mathcal{C}$ contains the evaluations of the integrand at the ends with weight $\frac{1}{2}$, eeded to make a true Trapezoidal rule:

$$\mathcal{C} = \frac{h}{2}f(u_0)\varphi_i(0) + \frac{h}{2}f(u_{N_n-1})\varphi_i(L)\,.$$

he answer $hf(u_i)$ must therefore be multiplied by $\frac{1}{2}$ if $i = 0$ or $i = N_n - 1$. ote that $\mathcal{C} = 0$ for $i = 1, \ldots, N_n - 2$.

One can alternatively use the Trapezoidal rule on the reference cell and ssemble the contributions. It is a bit more labor in this context, but working on ıe reference cell is safer as that approach is guaranteed to handle discontinuous erivatives of finite element functions correctly (not important in this particular xample), while the rule above was derived with the assumption that f is ontinuous at the integration points.

The conclusion is that it suffices to use the Trapezoidal rule if one wants o derive the difference equations in the finite element method and make them milar to those arising in the finite difference method. The Trapezoidal rule as sufficient accuracy for P1 elements, but for P2 elements one should turn to impson's rule.

4.7 Finite element discretization of a variable coef Laplace term

Turning back to the model problem (50), it remains to calculate the cont of the $(\alpha u')'$ and boundary terms to the difference equations. The int the variational form corresponding to $(\alpha u')'$ is

$$\int_0^L \alpha(\sum_k c_k\psi_k)\psi_i'\psi_j'\,\mathrm{d}x\,.$$

Numerical integration utilizing a value of $\sum_k c_k\psi_k$ from a previous i must in general be used to compute the integral. Now our aim is to i symbolically, as much as we can, to obtain some insight into how the finite method approximates this term. To be able to derive symbolic expressi must either turn to the group finite element method or numerical integr the node points. Finite element basis functions φ_i are now used.

Group finite element method. We set $\alpha(u) \approx \sum_k \alpha(u_k)\varphi_k$, and write

$$\int_0^L \alpha(\sum_k c_k\varphi_k)\varphi_i'\varphi_j'\,\mathrm{d}x \approx \sum_k(\underbrace{\int_0^L \varphi_k\varphi_i'\varphi_j'\,\mathrm{d}x}_{L_{i,j,k}})\alpha(u_k) = \sum_k L_{i,j,k}\alpha(u$$

Further calculations are now easiest to carry out in the reference cell. elements we can compute $L_{i,j,k}$ for the two k values that are relevant reference cell. Turning to local indices, one gets

$$L^{(e)}_{r,s,t} = \frac{1}{2h}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}, \quad t = 0, 1,$$

where $r, s, t = 0, 1$ are indices over local degrees of freedom in the refere $(i = q(e, r),\ j = q(e, s)$, and $k = q(e, t))$. The sum $\sum_k L_{i,j,k}\alpha(u_k)$ at level becomes $\sum_{t=0}^1 L^{(e)}_{r,s,t}\alpha(\tilde{u}_t)$, where $\tilde{u}_t$ is $u(x_{q(e,t)})$, i.e., the value of u node number t in cell number e. The element matrix becomes

$$\frac{1}{2}(\alpha(\tilde{u}_0) + \alpha(\tilde{u}^{(1)}))\frac{1}{h}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}.$$

As usual, we employ a left-to-right numbering of cells and nodes. Row n in the global matrix gets contributions from the first row of the element in cell i and the last row of the element matrix in cell $i - 1$. In cell $i - 1$ the sum $\alpha(\tilde{u}_0) + \alpha(\tilde{u}^{(1)})$ corresponds to $\alpha(u_{i-1}) + \alpha(u_i)$. The sa becomes $\alpha(u_i) + \alpha(u_{i+1})$ in cell number i. We can with this insight a the contributions to row number i in the global matrix:

$$\frac{1}{2h}(-(\alpha(u_{i-1}) + \alpha(u_i)), \quad \alpha(u_{i-1}) + 2\alpha(u_i) + \alpha(u_{i+1}), \quad \alpha(u_i) + \alpha(u$$

Multiplying by the vector of unknowns u_i results in a formula that can be arranged to

$$-\frac{1}{h}(\frac{1}{2}(\alpha(u_i)+\alpha(u_{i+1}))(u_{i+1}-u_i)-\frac{1}{2}(\alpha(u_{i-1})+\alpha(u_i))(u_i-u_{i-1})), \quad (70)$$

which is nothing but the standard finite difference discretization of $-(\alpha(u)u')'$ with an arithmetic mean of $\alpha(u)$ (and the usual factor h because of the integration in the finite element method).

Numerical integration at the nodes. Instead of using the group finite element method and exact integration we can turn to the Trapezoidal rule for computing $\int_0^L \alpha(\sum_k u_k\varphi_k)\varphi_i'\varphi_j'\,\mathrm{d}x$, again at the cell level since that is most convenient when we deal with discontinuous functions φ_i':

$$\begin{aligned}\int_{-1}^1 \alpha(\sum_t \tilde{u}_t\tilde{\varphi}_t)\tilde{\varphi}_r'\tilde{\varphi}_s'\frac{h}{2}dX &= \int_{-1}^1 \alpha(\sum_{t=0}^1 \tilde{u}_t\tilde{\varphi}_t)\frac{2}{h}\frac{d\tilde{\varphi}_r}{dX}\frac{2}{h}\frac{d\tilde{\varphi}_s}{dX}\frac{h}{2}dX\\ &= \frac{1}{2h}(-1)^r(-1)^s\int_{-1}^1 \alpha(\sum_{t=0}^1 u_t\tilde{\varphi}_t(X))dX\\ &\approx \frac{1}{2h}(-1)^r(-1)^s\alpha(\sum_{t=0}^1 \tilde{\varphi}_t(-1)\tilde{u}_t)+\alpha(\sum_{t=0}^1 \tilde{\varphi}_t(1)\tilde{u}_t)\\ &= \frac{1}{2h}(-1)^r(-1)^s(\alpha(\tilde{u}_0)+\alpha(\tilde{u}^{(1)}))\,. \quad (71)\end{aligned}$$

The element matrix in (71) is identical to the one in (69), showing that the group finite element method and Trapezoidal integration are equivalent with a standard finite discretization of a nonlinear Laplace term $(\alpha(u)u')'$ using an arithmetic mean for α: $[D_x\overline{x}D_xu]_i$.

Remark about integration in the physical x coordinate.

We might comment on integration in the physical coordinate system too. The common Trapezoidal rule in Section 4.6 cannot be used to integrate derivatives like φ_i', because the formula is derived under the assumption of a continuous integrand. One must instead use the more basic version of the Trapezoidal rule where all the trapezoids are summed up. This is straightforward, but I think it is even more straightforward to apply the Trapezoidal rule on the reference cell and assemble the contributions.

The term $\int auv\,\mathrm{d}x$ in the variational form is linear and gives these terms in the algebraic equations:

$$\frac{ah}{6}(u_{i-1}+4u_i+u_{i+1}) = ah[u-\frac{h^2}{6}D_xD_xu]_i\,.$$

The final term in the variational form is the Neumann condition at the ary: $Cv(0) = C\varphi_i(0)$. With a left-to-right numbering only $i=0$ wil contribution $Cv(0) = C\delta_{i0}$ (since $\varphi_i(0)\neq 0$ only for $i=0$).

Summary.

For the equation

$$-(\alpha(u)u')' + au = f(u),$$

P1 finite elements results in difference equations where

- the term $-(\alpha(u)u')'$ becomes $-h[D_x\overline{\alpha(u)}^xD_xu]_i$ if the group element method or Trapezoidal integration is applied,
- $f(u)$ becomes $hf(u_i)$ with Trapezoidal integration or the "mas trix" representation $h[f(u)-\frac{h}{6}D_xD_xf(u)]_i$ if computed by a g finite element method,
- au leads to the "mass matrix" form $ah[u-\frac{h}{6}D_xD_xu]_i$.

As we now have explicit expressions for the nonlinear difference eq also in the finite element method, a Picard or Newton method can be de shown for the finite difference method. However, our efforts in deriving s forms of the difference equations in the finite element method was motiva desire to see how nonlinear terms in differential equations make the finite and difference method different. For practical calculations in computer p we apply numerical integration, normally the more accurate Gauss-L quadrature rules, to the integrals directly. This allows us to easily *eval* nonlinear algebraic equations for a given numerical approximation of denoted u^-). To *solve* the nonlinear algebraic equations we need to ap Picard iteration method or Newton's method to the variational form dir shown next.

4.8 Picard iteration defined from the variational for

We address again the problem (50) with the corresponding variational fo Our aim is to define a Picard iteration based on this variational form wit attempt to compute integrals symbolically as in the previous three sectio idea in Picard iteration is to use a previously computed u value in the n functions $\alpha(u)$ and $f(u)$. Let u^- be the available approximation to u f previous iteration. The linearized variational form for Picard iteration

$$\int_0^L (\alpha(u^-)u'v' + auv)\,\mathrm{d}x = \int_0^L f(u^-)v\,\mathrm{d}x - Cv(0), \quad \forall v \in V\,. \tag{72}$$

his is a linear problem $a(u,v) = L(v)$ with bilinear and linear forms

$$a(u,v) = \int_0^L (\alpha(u^-)u'v' + auv)\,\mathrm{d}x, \quad L(v) = \int_0^L f(u^-)v\,\mathrm{d}x - Cv(0)\,.$$

Iake sure to distinguish the coefficient a in auv from the differential equation om the a in the abstract bilinear form notation $a(\cdot,\cdot)$.

The linear system associated with (72) is computed the standard way. Tech- ically, we are back to solving $-(\alpha(x)u')' + au = f(x)$. The unknown u is sought n the form $u = B(x) + \sum_{j\in\mathcal{I}_s} c_j\psi_j$, with $B(x) = D$ and $\psi_i = \varphi_{\nu(i)}$, $\nu(i) = i+1$, nd $\mathcal{I}_s = \{0,1,\ldots,N = N_n - 2\}$.

.9 Newton's method defined from the variational form

pplication of Newton's method to the nonlinear variational form (67) arising om the problem (50) requires identification of the nonlinear algebraic equations $_i = 0$. Although we originally denoted the unknowns in nonlinear algebraic quations by $u_0,\ldots,u_N$, it is in the present context most natural to have the nknowns as $c_0,\ldots,c_N$ and write

$$F_i(c_0,\ldots,c_N) = 0, \quad i \in \mathcal{I}_s,$$

nd define the Jacobian as $J_{i,j} = \partial F_i/\partial c_j$ for $i,j \in \mathcal{I}_s$.

The specific form of the equations $F_i = 0$ follows from the variational form

$$\int_0^L (\alpha(u)u'v' + auv)\,\mathrm{d}x = \int_0^L f(u)v\,\mathrm{d}x - Cv(0), \quad \forall v \in V,$$

y choosing $v = \psi_i$, $i \in \mathcal{I}_s$, and setting $u = \sum_{j\in\mathcal{I}_s} c_j\psi_j$, maybe with a boundary unction to incorporate Dirichlet conditions.

With $v = \psi_i$ we get

$$F_i = \int_0^L (\alpha(u)u'\psi_i' + au\psi_i - f(u)\psi_i)\,\mathrm{d}x + C\psi_i(0) = 0, \quad i \in \mathcal{I}_s\,. \tag{73}$$

ı the differentiations leading to the Jacobian we will frequently use the results

$$\frac{\partial u}{\partial c_j} = \frac{\partial}{\partial c_j}\sum_k c_k\psi_k = \psi_j, \quad \frac{\partial u'}{\partial c_j} = \frac{\partial}{\partial c_j}\sum_k c_k\psi_k' = \psi_j'\,.$$

he derivation of the Jacobian of (73) goes as

$$\begin{aligned}
J_{i,j} = \frac{\partial F_i}{\partial c_j} &= \int_0^L \frac{\partial}{\partial c_j}(\alpha(u)u'\psi_i' + au\psi_i - f(u)\psi_i)\,\mathrm{d}x \\
&= \int_0^L ((\alpha'(u)\frac{\partial u}{\partial c_j}u' + \alpha(u)\frac{\partial u'}{\partial c_j})\psi_i' + a\frac{\partial u}{\partial c_j}\psi_i - f'(u)\frac{\partial u}{\partial c_j}\psi \\
&= \int_0^L ((\alpha'(u)\psi_j u' + \alpha(u)\psi_j')\psi_i' + a\psi_j\psi_i - f'(u)\psi_j\psi_i)\,\mathrm{d}x \\
&= \int_0^L (\alpha'(u)u'\psi_i'\psi_j + \alpha(u)\psi_i'\psi_j' + (a - f(u))\psi_i\psi_j)\,\mathrm{d}x
\end{aligned}$$

When calculating the right-hand side vector F_i and the coefficient $J_{i,j}$ in the linear system to be solved in each Newton iteration, one mu previously computed u, denoted by u^-, for the symbol u in (73) and (74 this notation we have

$$F_i = \int_0^L \left(\alpha(u^-)u^{-\prime}\psi_i' + (a - f(u^-))\psi_i\right)\,\mathrm{d}x - C\psi_i(0), \quad i \in \mathcal{I}_s,$$

$$J_{i,j} = \int_0^L (\alpha'(u^-)u^{-\prime}\psi_i'\psi_j + \alpha(u^-)\psi_i'\psi_j' + (a - f(u^-))\psi_i\psi_j)\,\mathrm{d}x, \quad i,j$$

These expressions can be used for any basis $\{\psi_i\}_{i\in\mathcal{I}_s}$. Choosing finite functions for ψ_i, one will normally want to compute the integral contribu by cell, working in a reference cell. To this end, we restrict the integratio cell and transform the cell to $[-1,1]$. The most recently computed approx u^- to u becomes $\tilde{u}^- = \sum_t \tilde{u}_t^{-1}\tilde{\varphi}_t(X)$ over the reference element, wher the value of u^- at global node (or degree of freedom) $q(e,t)$ correspondin local node t (or degree of freedom). The formulas (75) and (76) then ch

$$\tilde{F}_r^{(e)} = \int_{-1}^1 \left(\alpha(\tilde{u}^-)\tilde{u}^{-\prime}\tilde{\varphi}_r' + (a - f(\tilde{u}^-))\tilde{\varphi}_r\right)\det J\,\mathrm{d}X - C\tilde{\varphi}_r(0),$$

$$\tilde{J}_{r,s}^{(e)} = \int_{-1}^1 (\alpha'(\tilde{u}^-)\tilde{u}^{-\prime}\tilde{\varphi}_r'\tilde{\varphi}_s + \alpha(\tilde{u}^-)\tilde{\varphi}_r'\tilde{\varphi}_s' + (a - f(\tilde{u}^-))\tilde{\varphi}_r\tilde{\varphi}_s)\det J\,\mathrm{d}X$$

with $r,s \in I_d$ runs over the local degrees of freedom.

Many finite element programs require the user to provide F_i and $J_{i,}$ programs, like FEniCS[2], are capable of automatically deriving $J_{i,j}$ specified.

Dirichlet conditions. Incorporation of the Dirichlet values by ass contributions from all degrees of freedom and then modifying the linear

[2] `http://fenicsproject.org`

an be obviously be applied to Picard iteration as that method involves a standard near system. In the Newton system, however, the unknown is a correction u to the solution. Dirichlet conditions are implemented by inserting them in ne initial guess u^- for the Newton iteration and implementing $\delta u_i = 0$ for all nown degrees of freedom. The manipulation of the linear system follows exactly ne algorithm in the linear problems, the only difference being that the known alues are zero.

5 Multi-dimensional PDE problems

he fundamental ideas in the derivation of F_i and $J_{i,j}$ in the 1D model problem easily generalized to multi-dimensional problems. Nevertheless, the expressions nvolved are slightly different, with derivatives in x replaced by ∇, so we present ome examples below in detail.

.1 Finite element discretization

s an example, Backward Euler discretization of the PDE

$$u_t = \nabla \cdot (\alpha(u)\nabla u) + f(u), \tag{79}$$

ives the nonlinear time-discrete PDEs

$$u^n - \Delta t \nabla \cdot (\alpha(u^n)\nabla u^n) + f(u^n) = u^{n-1}\,.$$

Ve may alternatively write this equation with u for u^n and $u^{(1)}$ for u^{n-1}:

$$u - \Delta t \nabla \cdot (\alpha(u)\nabla u) - \Delta t f(u) = u^{(1)}\,.$$

(Note that the mathematical meaning of the symbol u changes in these quations: $u(\boldsymbol{x}, t)$ is the exact solution of (79), $u^n(\boldsymbol{x})$ is an approximation to the xact solution at $t = t_n$, while $u(\boldsymbol{x})$ in the latter equation is a synonym for u^n. elow, this $u(\boldsymbol{x})$ will be approximated by a new $u = \sum_k c_k\psi_k(\boldsymbol{x})$ in space, and nen the actual u symbol used in the Picard and Newton iterations is a further pproximation of $\sum_k c_k\psi_k$ arising from the nonlinear iteration algorithm.)

Let us assume homogeneous Neumann conditions on the entire boundary for mplicity in the boundary term. The variational form becomes: find $u \in V$ such nat

$$\int_\Omega (uv + \Delta t\,\alpha(u)\nabla u \cdot \nabla v - \Delta t f(u)v - u^{(1)}v)\,\mathrm{d}x = 0, \quad \forall v \in V\,. \tag{80}$$

he nonlinear algebraic equations follow from setting $v = \psi_i$ and using the epresentation $u = \sum_k c_k\psi_k$, which we just write as

$$F_i = \int_\Omega (u\psi_i + \Delta t\,\alpha(u)\nabla u \cdot \nabla\psi_i - \Delta t f(u)\psi_i - u^{(1)}\psi_i)\,\mathrm{d}x\,. \tag{81}$$

Picard iteration needs a linearization where we use the most recent approx u^- to u in α and f:

$$F_i \approx \hat{F}_i = \int_\Omega (u\psi_i + \Delta t\,\alpha(u^-)\nabla u \cdot \nabla\psi_i - \Delta t f(u^-)\psi_i - u^{(1)}\psi_i)\,\mathrm{d}x\,.$$

The equations $\hat{F}_i = 0$ are now linear and we can easily derive a linear $\sum_{j\in\mathcal{I}_s} A_{i,j}c_j = b_i$, $i \in \mathcal{I}_s$, for the unknown coefficients $\{c_i\}_{i\in\mathcal{I}_s}$ by in $u = \sum_j c_j\psi_j$. We get

$$A_{i,j} = \int_\Omega (\varphi_j\psi_i + \Delta t\,\alpha(u^-)\nabla\varphi_j \cdot \nabla\psi_i)\,\mathrm{d}x, \quad b_i = \int_\Omega (\Delta t f(u^-)\psi_i + u^{(1)}$$

In Newton's method we need to evaluate F_i with the known value u

$$F_i \approx \hat{F}_i = \int_\Omega (u^-\psi_i + \Delta t\,\alpha(u^-)\nabla u^- \cdot \nabla\psi_i - \Delta t f(u^-)\psi_i - u^{(1)}\psi_i)\,\mathrm{d}x$$

The Jacobian is obtained by differentiating (81) and using

$$\frac{\partial u}{\partial c_j} = \sum_k \frac{\partial}{\partial c_j} c_k\psi_k = \psi_j,$$
$$\frac{\partial \nabla u}{\partial c_j} = \sum_k \frac{\partial}{\partial c_j} c_k\nabla\psi_k = \nabla\psi_j\,.$$

The result becomes

$$J_{i,j} = \frac{\partial F_i}{\partial c_j} = \int_\Omega (\psi_j\psi_i + \Delta t\,\alpha'(u)\psi_j\nabla u \cdot \nabla\psi_i + \Delta t\,\alpha(u)\nabla\psi_j \cdot \nabla\psi_i$$
$$\Delta t f'(u)\psi_j\psi_i)\,\mathrm{d}x\,.$$

The evaluation of $J_{i,j}$ as the coefficient matrix in the linear system in N method applies the known approximation u^- for u:

$$J_{i,j} = \int_\Omega (\psi_j\psi_i + \Delta t\,\alpha'(u^-)\psi_j\nabla u^- \cdot \nabla\psi_i + \Delta t\,\alpha(u^-)\nabla\psi_j \cdot \nabla\psi_i -$$
$$\Delta t f'(u^-)\psi_j\psi_i)\,\mathrm{d}x\,.$$

Hopefully, this example also shows how convenient the notation with u is: the unknown to be computed is always u and linearization by inserting (previously computed) values is a matter of adding an underscore. One great advantage of this quick notation in software [2].

Non-homogeneous Neumann conditions. A natural physical flux condition for the PDE (79) takes the form of a non-homogeneous Neumann condition

$$-\alpha(u)\frac{\partial u}{\partial n} = g, \quad \boldsymbol{x} \in \partial\Omega_N, \tag{88}$$

where g is a prescribed function and $\partial\Omega_N$ is a part of the boundary of the domain Ω. From integrating $\int_\Omega \nabla\cdot(\alpha\nabla u)\,\mathrm{d}x$ by parts, we get a boundary term

$$\int_{\partial\Omega_N} \alpha(u)\frac{\partial u}{\partial u} v\,\mathrm{d}s\,. \tag{89}$$

Inserting the condition (88) into this term results in an integral over prescribed values:

$$-\int_{\partial\Omega_N} gv\,\mathrm{d}s\,.$$

The nonlinearity in the $\alpha(u)$ coefficient condition (88) therefore does not contribute with a nonlinearity in the variational form.

Robin conditions. Heat conduction problems often apply a kind of Newton's cooling law, also known as a Robin condition, at the boundary:

$$-\alpha(u)\frac{\partial u}{\partial u} = h(u)(u - T_s(t)), \quad \boldsymbol{x} \in \partial\Omega_R, \tag{90}$$

where $h(u)$ is a heat transfer coefficient between the body (Ω) and its surroundings, T_s is the temperature of the surroundings, and $\partial\Omega_R$ is a part of the boundary where this Robin condition applies. The boundary integral (89) now becomes

$$\int_{\partial\Omega_R} h(u)(u - T_s(T))v\,\mathrm{d}s\,.$$

In many physical applications, $h(u)$ can be taken as constant, and then the boundary term is linear in u, otherwise it is nonlinear and contributes to the Jacobian in a Newton method. Linearization in a Picard method will typically use a known value in h, but keep the u in $u - T_s$ as unknown: $h(u^-)(u - T_s(t))$. Exercise 14 asks you to carry out the details.

.2 Finite difference discretization

A typical diffusion equation

$$u_t = \nabla\cdot(\alpha(u)\nabla u) + f(u),$$

can be discretized by (e.g.) a Backward Euler scheme, which in 2D can be written

$$[D_t^- u = D_x\overline{\alpha(u)}^x D_x u + D_y\overline{\alpha(u)}^y D_y u + f(u)]^n_{i,j}\,.$$

We do not dive into the details of handling boundary conditions now. I and Neumann conditions are handled as in a corresponding linear, v coefficient diffusion problems.

Writing the scheme out, putting the unknown values on the left-ha and known values on the right-hand side, and introducing $\Delta x = \Delta y = h$ some writing, one gets

$$\begin{aligned} u^n_{i,j} - \frac{\Delta t}{h^2}(&\frac{1}{2}(\alpha(u^n_{i,j}) + \alpha(u^n_{i+1,j}))(u^n_{i+1,j} - u^n_{i,j}) \\ &- \frac{1}{2}(\alpha(u^n_{i-1,j}) + \alpha(u^n_{i,j}))(u^n_{i,j} - u^n_{i-1,j}) \\ &+ \frac{1}{2}(\alpha(u^n_{i,j}) + \alpha(u^n_{i,j+1}))(u^n_{i,j+1} - u^n_{i,j}) \\ &- \frac{1}{2}(\alpha(u^n_{i,j-1}) + \alpha(u^n_{i,j}))(u^n_{i,j} - u^n_{i-1,j-1})) - \Delta t f(u^n_{i,j}) = u^n_{i,} \end{aligned}$$

This defines a nonlinear algebraic system on the form $A(u)u = b(u)$.

Picard iteration. The most recently computed values u^- of u^n can in α and f for a Picard iteration, or equivalently, we solve $A(u^-)u = b(u$ result is a linear system of the same type as arising from $u_t = \nabla\cdot(\alpha($ $f(\boldsymbol{x}, t)$.

The Picard iteration scheme can also be expressed in operator nota

$$[D_t^- u = D_x\overline{\alpha(u^-)}^x D_x u + D_y\overline{\alpha(u^-)}^y D_y u + f(u^-)]^n_{i,j}\,.$$

Newton's method. As always, Newton's method is technically more i than Picard iteration. We first define the nonlinear algebraic equatio solved, drop the superscript n (use u for u^n), and introduce $u^{(1)}$ for u^n

$$\begin{aligned} F_{i,j} = u_{i,j} - \frac{\Delta t}{h^2}(&\\ &\frac{1}{2}(\alpha(u_{i,j}) + \alpha(u_{i+1,j}))(u_{i+1,j} - u_{i,j}) - \\ &\frac{1}{2}(\alpha(u_{i-1,j}) + \alpha(u_{i,j}))(u_{i,j} - u_{i-1,j}) + \\ &\frac{1}{2}(\alpha(u_{i,j}) + \alpha(u_{i,j+1}))(u_{i,j+1} - u_{i,j}) - \\ &\frac{1}{2}(\alpha(u_{i,j-1}) + \alpha(u_{i,j}))(u_{i,j} - u_{i-1,j-1})) - \Delta t\, f(u_{i,j}) - u^{(1)}_{i,j} \end{aligned}$$

It is convenient to work with two indices i and j in 2D finite difference di tions, but it complicates the derivation of the Jacobian, which then g indices. (Make sure you really understand the 1D version of this pro treated in Section 4.1.) The left-hand expression of an equation $F_{i,j} = 0$

ifferentiated with respect to each of the unknowns $u_{r,s}$ (recall that this is short otation for $u^n_{r,s}$), $r \in \mathcal{I}_x$, $s \in \mathcal{I}_y$:

$$J_{i,j,r,s} = \frac{\partial F_{i,j}}{\partial u_{r,s}}\,.$$

he Newton system to be solved in each iteration can be written as

$$\sum_{r\in\mathcal{I}_x}\sum_{s\in\mathcal{I}_y} J_{i,j,r,s}\delta u_{r,s} = -F_{i,j}, \quad i \in \mathcal{I}_x,\ j \in \mathcal{I}_y\,.$$

Given i and j, only a few r and s indices give nonzero contribution to the acobian since $F_{i,j}$ contains $u_{i\pm1,j}$, $u_{i,j\pm1}$, and $u_{i,j}$. This means that $J_{i,j,r,s}$ has onzero contributions only if $r = i \pm 1$, $s = j \pm 1$, as well as $r = i$ and $s = j$. The orresponding terms in $J_{i,j,r,s}$ are $J_{i,j,i-1,j}$, $J_{i,j,i+1,j}$, $J_{i,j,i,j-1}$, $J_{i,j,i,j+1}$, and $_{i,j,i,j}$. Therefore, the left-hand side of the Newton system, $\sum_r\sum_s J_{i,j,r,s}\delta u_{r,s}$ ollapses to

$$\begin{aligned}{}_{i,j,r,s}\delta u_{r,s} = J_{i,j,i,j}\delta u_{i,j} + J_{i,j,i-1,j}\delta u_{i-1,j} + J_{i,j,i+1,j}\delta u_{i+1,j} + J_{i,j,i,j-1}\delta u_{i,j-1} \\ + J_{i,j,i,j+1}\delta u_{i,j+1}\end{aligned}$$

he specific derivatives become

$$\begin{aligned}
J_{i,j,i-1,j} &= \frac{\partial F_{i,j}}{\partial u_{i-1,j}} \\
&= \frac{\Delta t}{h^2}(\alpha'(u_{i-1,j})(u_{i,j} - u_{i-1,j}) + \alpha(u_{i-1,j})(-1)), \\
J_{i,j,i+1,j} &= \frac{\partial F_{i,j}}{\partial u_{i+1,j}} \\
&= \frac{\Delta t}{h^2}(-\alpha'(u_{i+1,j})(u_{i+1,j} - u_{i,j}) - \alpha(u_{i-1,j})), \\
J_{i,j,i,j-1} &= \frac{\partial F_{i,j}}{\partial u_{i,j-1}} \\
&= \frac{\Delta t}{h^2}(\alpha'(u_{i,j-1})(u_{i,j} - u_{i,j-1}) + \alpha(u_{i,j-1})(-1)), \\
J_{i,j,i,j+1} &= \frac{\partial F_{i,j}}{\partial u_{i,j+1}} \\
&= \frac{\Delta t}{h^2}(-\alpha'(u_{i,j+1})(u_{i,j+1} - u_{i,j}) - \alpha(u_{i,j-1}))\,.
\end{aligned}$$

he $J_{i,j,i,j}$ entry has a few more terms and is left as an exercise. Inserting ie most recent approximation u^- for u in the J and F formulas and then orming $J\delta u = -F$ gives the linear system to be solved in each Newton iteration. oundary conditions will affect the formulas when any of the indices coincide ith a boundary value of an index.

5.3 Continuation methods

Picard iteration or Newton's method may diverge when solving PDEs wit nonlinearities. Relaxation with $\omega < 1$ may help, but in highly nonlinear p it can be necessary to introduce a *continuation parameter* Λ in the p $\Lambda = 0$ gives a version of the problem that is easy to solve, while $\Lambda = 1$ is th problem. The idea is then to increase Λ in steps, $\Lambda_0 = 0, \Lambda_1 < \cdots < \Lambda_n =$ use the solution from the problem with Λ_{i-1} as initial guess for the itera the problem corresponding to Λ_i.

The continuation method is easiest to understand through an e Suppose we intend to solve

$$-\nabla\cdot(||\nabla u||^q\nabla u) = f,$$

which is an equation modeling the flow of a non-Newtonian fluid th channel or pipe. For $q = 0$ we have the Poisson equation (correspond Newtonian fluid) and the problem is linear. A typical value for pseudo fluids may be $q_n = -0.8$. We can introduce the continuation parameter Λ such that $q = q_n\Lambda$. Let $\{\Lambda_\ell\}_{\ell=0}^n$ be the sequence of Λ values in $[0,$ corresponding q values $\{q_\ell\}_{\ell=0}^n$. We can then solve a sequence of proble

$$-\nabla\cdot\left(||\nabla u^\ell||_\ell^q\nabla u^\ell\right) = f, \quad \ell = 0,\ldots,n,$$

where the initial guess for iterating on u^ℓ is the previously computed $u^{\ell-1}$. If a particular Λ_ℓ leads to convergence problems, one may try a increase in Λ: $\Lambda_* = \frac{1}{2}(\Lambda_{\ell-1} + \Lambda_\ell)$, and repeat halving the step in convergence is reestablished.

6 Exercises

Problem 1: Determine if equations are nonlinear or

Classify each term in the following equations as linear or nonlinear. Assu u is an unknown function and that all other symbols are known quanti

1. $b^2 = 1$
2. $a + b = 1$
3. $mu'' + \beta|u'|u' + cu = F(t)$
4. $u_t = \alpha u_{xx}$
5. $u_{tt} = c^2\nabla^2 u$
6. $u_t = \nabla\cdot(\alpha(u)\nabla u) + f(x,y)$
7. $u_t + f(u)_x = 0$
8. $\boldsymbol{u}_t + \boldsymbol{u}\cdot\nabla\boldsymbol{u} = -\nabla p + r\nabla^2\boldsymbol{u}$, $\nabla\cdot\boldsymbol{u} = 0$ ($\boldsymbol{u}$ is a vector field)

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

10. $\nabla^2 u = \lambda e^u$

ilename: `nonlinear_vs_linear.pdf`.

'roblem 2: Experience the behavior of Newton's method

he program `Newton_demo.py`[3] illustrates graphically each step in Newton's ıethod and is run like

```
erminal> python Newton_demo.py f dfdx x0 xmin xmax
```

se this program to investigate potential problems with Newton's method when olving $e^{-0.5x^2}\cos(\pi x) = 0$. Try a starting point $x_0 = 0.8$ and $x_0 = 0.85$ and atch the different behavior. Just run

```
erminal>  python Newton_demo.py '0.2 + exp(-0.5*x**2)*cos(pi*x)' \
          '-x*exp(-x**2)*cos(pi*x) - pi*exp(-x**2)*sin(pi*x)' \
          0.85 -3 3
```

nd repeat with 0.85 replaced by 0.8.

'roblem 3: Compute the Jacobian of a 2×2 system

/rite up the system (18)-(19) in the form $F(u) = 0$, $F = (F_0, F_1)$, $u = (u_0, u_1)$, nd compute the Jacobian $J_{i,j} = \partial F_i/\partial u_j$.

'roblem 4: Solve nonlinear equations arising from a vibration ODE

onsider a nonlinear vibration problem

$$mu'' + bu'|u'| + s(u) = F(t), \tag{91}$$

here $m > 0$ is a constant, $b \geq 0$ is a constant, $s(u)$ a possibly nonlinear function f u, and $F(t)$ is a prescribed function. Such models arise from Newton's second w of motion in mechanical vibration problems where $s(u)$ is a spring or restoring rce, mu'' is mass times acceleration, and $bu'|u'|$ models water or air drag.

) Rewrite the equation for u as a system of two first-order ODEs, and discretize is system by a Crank-Nicolson (centered difference) method. With $v = u'$, we et a nonlinear term $v^{n+\frac{1}{2}}|v^{n+\frac{1}{2}}|$. Use a geometric average for $v^{n+\frac{1}{2}}$.

[3] `http://tinyurl.com/nm5587k/nonlin/Newton_demo.py`

b) Formulate a Picard iteration method to solve the system of nonlinear a equations.

c) Explain how to apply Newton's method to solve the nonlinear equa each time level. Derive expressions for the Jacobian and the right-hanc each Newton iteration.
Filename: `nonlin_vib.pdf`.

Exercise 5: Find the truncation error of arithmetic of products

In Section 3.4 we introduce alternative arithmetic means of a product. product is $P(t)Q(t)$ evaluated at $t = t_{n+\frac{1}{2}}$. The exact value is

$$[PQ]^{n+\frac{1}{2}} = P^{n+\frac{1}{2}}Q^{n+\frac{1}{2}}$$

There are two obvious candidates for evaluating $[PQ]^{n+\frac{1}{2}}$ as a mean of v P and Q at t_n and t_{n+1}. Either we can take the arithmetic mean of eac P and Q,

$$[PQ]^{n+\frac{1}{2}} \approx \frac{1}{2}(P^n + P^{n+1})\frac{1}{2}(Q^n + Q^{n+1}),$$

or we can take the arithmetic mean of the product PQ:

$$[PQ]^{n+\frac{1}{2}} \approx \frac{1}{2}(P^nQ^n + P^{n+1}Q^{n+1})\,.$$

The arithmetic average of $P(t_{n+\frac{1}{2}})$ is $\mathcal{O}(\Delta t^2)$:

$$P(t_{n+\frac{1}{2}}) = \frac{1}{2}(P^n + P^{n+1}) + \mathcal{O}(\Delta t^2)\,.$$

A fundamental question is whether (92) and (93) have different orders of a in $\Delta t = t_{n+1} - t_n$. To investigate this question, expand quantities at $t_$ t_n in Taylor series around $t_{n+\frac{1}{2}}$, and subtract the true value $[PQ]^{n+\frac{1}{2}}$ f approximations (92) and (93) to see what the order of the error terms a

Hint. You may explore `sympy` for carrying out the tedious calculat general Taylor series expansion of $P(t + \frac{1}{2}\Delta t)$ around t involving just a function $P(t)$ can be created as follows:

```
>>> from sympy import *
>>> t, dt = symbols('t dt')
>>> P = symbols('P', cls=Function)
>>> P(t).series(t, 0, 4)
P(0) + t*Subs(Derivative(P(_x), _x), (_x,), (0,)) +
t**2*Subs(Derivative(P(_x), _x, _x), (_x,), (0,))/2 +
t**3*Subs(Derivative(P(_x), _x, _x, _x), (_x,), (0,))/6 + O(t**4)
>>> P_p = P(t).series(t, 0, 4).subs(t, dt/2)
>>> P_p
P(0) + dt*Subs(Derivative(P(_x), _x), (_x,), (0,))/2 +
```

```
dt**2*Subs(Derivative(P(_x), _x, _x), (_x,), (0,))/8 +
dt**3*Subs(Derivative(P(_x), _x, _x, _x), (_x,), (0,))/48 + O(dt**4)
```

he error of the arithmetic mean, $\frac{1}{2}(P(-\frac{1}{2}\Delta t) + P(-\frac{1}{2}\Delta t))$ for $t = 0$ is then

```
>>> P_m = P(t).series(t, 0, 4).subs(t, -dt/2)
>>> mean = Rational(1,2)*(P_m + P_p)
>>> error = simplify(expand(mean) - P(0))
>>> error
dt**2*Subs(Derivative(P(_x), _x, _x), (_x,), (0,))/8 + O(dt**4)
```

se these examples to investigate the error of (92) and (93) for $n = 0$. (Choosing $= 0$ is necessary for making the expressions too complicated for `sympy`, but here is of course no lack of generality by using $n = 0$ rather than an arbitrary n the main point is the product and addition of Taylor series.)
ilename: `product_arith_mean.py`.

Problem 6: Newton's method for linear problems

uppose we have a linear system $F(u) = Au - b = 0$. Apply Newton's method o this system, and show that the method converges in one iteration. Filename: `ewton_linear.pdf`.

Exercise 7: Discretize a 1D problem with a nonlinear coefficient

We consider the problem

$$((1 + u^2)u')' = 1, \quad x \in (0, 1), \quad u(0) = u(1) = 0\,. \tag{94}$$

) Discretize (94) by a centered finite difference method on a uniform mesh.

) Discretize (94) by a finite element method with P1 of equal length. Use the rapezoidal method to compute all integrals. Set up the resulting matrix system n symbolic form such that the equations can be compared with those in a).
ilename: `nonlin_1D_coeff_discretize.pdf`.

Exercise 8: Linearize a 1D problem with a nonlinear coefficient

We have a two-point boundary value problem

$$((1 + u^2)u')' = 1, \quad x \in (0, 1), \quad u(0) = u(1) = 0\,. \tag{95}$$

) Construct a Picard iteration method for (95) without discretizing in space.

) Apply Newton's method to (95) without discretizing in space.

) Discretize (95) by a centered finite difference scheme. Construct a Picard ethod for the resulting system of nonlinear algebraic equations.

d) Discretize (95) by a centered finite difference scheme. Define the of nonlinear algebraic equations, calculate the Jacobian, and set up N method for solving the system.
Filename: `nonlin_1D_coeff_linearize.pdf`.

Problem 9: Finite differences for the 1D Bratu probl

We address the so-called Bratu problem

$$u'' + \lambda e^u = 0, \quad x \in (0, 1), \quad u(0) = u(1) = 0,$$

where λ is a given parameter and u is a function of x. This is a widely use problem for studying numerical methods for nonlinear differential eq The problem (96) has an exact solution

$$u_{\mathrm{e}}(x) = -2\ln\left(\frac{\cosh((x - \frac{1}{2})\theta/2)}{\cosh(\theta/4)}\right),$$

where θ solves

$$\theta = \sqrt{2\lambda}\cosh(\theta/4)\,.$$

There are two solutions of (96) for $0 < \lambda < \lambda_c$ and no solution for $\lambda > \lambda = \lambda_c$ there is one unique solution. The critical value λ_c solves

$$1 = \sqrt{2\lambda_c}\frac{1}{4}\sinh(\theta(\lambda_c)/4)\,.$$

A numerical value is $\lambda_c = 3.513830719$.

a) Discretize (96) by a centered finite difference method.

b) Set up the nonlinear equations $F_i(u_0, u_1, \ldots, u_{N_x}) = 0$ from a). C the associated Jacobian.

c) Implement a solver that can compute $u(x)$ using Newton's method. error as a function of x in each iteration.

d) Investigate whether Newton's method gives second-order converg computing $||u_{\mathrm{e}} - u||/||u_{\mathrm{e}} - u^-||^2$ in each iteration, where u is solutio current iteration and u^- is the solution in the previous iteration.
Filenames: `nonlin_1D_Bratu_fd.pdf`, `nonlin_1D_Bratu_fd.py`.

Problem 10: Integrate functions of finite element e sions

We shall investigate integrals on the form

$$\int_0^L f(\sum_k u_k\varphi_k(x))\varphi_i(x)\,\mathrm{d}x,$$

here $\varphi_i(x)$ are P1 finite element basis functions and u_k are unknown coefficients, ıore precisely the values of the unknown function u at nodes x_k. We introduce a ɔde numbering that goes from left to right and also that all cells have the same ɛngth h. Given i, the integral only gets contributions from $[x_{i-1}, x_{i+1}]$. On this ıterval $\varphi_k(x) = 0$ for $k < i-1$ and $k > i+1$, so only three basis functions will ɔntribute:

$$\sum_k u_k\varphi_k(x) = u_{i-1}\varphi_{i-1}(x) + u_i\varphi_i(x) + u_{i+1}\varphi_{i+1}(x)\,.$$

he integral (97) now takes the simplified form

$$\int_{x_{i-1}}^{x_{i+1}} f(u_{i-1}\varphi_{i-1}(x) + u_i\varphi_i(x) + u_{i+1}\varphi_{i+1}(x))\varphi_i(x)\,\mathrm{d}x\,.$$

plit this integral in two integrals over cell L (left), $[x_{i-1}, x_i]$, and cell R (right), $:_i, x_{i+1}]$. Over cell L, u simplifies to $u_{i-1}\varphi_{i-1} + u_i\varphi_i$ (since $\varphi_{i+1} = 0$ on this ell), and over cell R, u simplifies to $u_i\varphi_i + u_{i+1}\varphi_{i+1}$. Make a `sympy` program ıat can compute the integral and write it out as a difference equation. Give ıe $f(u)$ formula on the command line. Try out $f(u) = u^2, \sin u, \exp u$.

Hint. Introduce symbols `u_i`, `u_im1`, and `u_ip1` for u_i, u_{i-1}, and u_{i+1}, re- ɔectively, and similar symbols for x_i, x_{i-1}, and x_{i+1}. Find formulas for the ıasis functions on each of the two cells, make expressions for u on the two cells, ıtegrate over each cell, expand the answer and simplify. You can ask `sympy` for ᵀEX code and render it either by creating a LaTeX document and compiling it ɔ a PDF document or by using `http://latex.codecogs.com` to display LaTeX ɔrmulas in a web page. Here are some appropriate Python statements for the ıtter purpose:

```
rom sympy import *
..
# expr_i holdes the integral as a sympy expression
latex_code = latex(expr_i, mode='plain')
# Replace u_im1 sympy symbol name by latex symbol u_{i-1}
latex_code = latex_code.replace('im1', '{i-1}')
# Replace u_ip1 sympy symbol name by latex symbol u_{i+1}
latex_code = latex_code.replace('ip1', '{i+1}')
# Escape (quote) latex_code so it can be sent as HTML text
import cgi
ıtml_code = cgi.escape(latex_code)
# Make a file with HTML code for displaying the LaTeX formula
 = open('tmp.html', 'w')
# Include an image that can be clicked on to yield a new
# page with an interactive editor and display area where the
# formula can be further edited
ext = """
<a href="http://www.codecogs.com/eqnedit.php?latex=%(html_code)s"
 target="_blank">
<img src="http://latex.codecogs.com/gif.latex?%(html_code)s"
 title="%(latex_code)s"/>
</a>
 """ % vars()
```



```
f.write(text)
f.close()
```

The formula is displayed by loading `tmp.html` into a web browser.
Filename: `fu_fem_int.py`.

Problem 11: Finite elements for the 1D Bratu probl

We address the same 1D Bratu problem as described in Problem 9.

a) Discretize (11) by a finite element method using a uniform mesh elements. Use a group finite element method for the e^u term.

b) Set up the nonlinear equations $F_i(u_0, u_1, \ldots, u_{N_x}) = 0$ from a). C the associated Jacobian.
Filename: `nonlin_1D_Bratu_fe.pdf`.

Exercise 12: Discretize a nonlinear 1D heat condu PDE by finite differences

We address the 1D heat conduction PDE

$$\varrho c(T)T_t = (k(T)T_x)_x,$$

for $x \in [0, L]$, where ϱ is the density of the solid material, $c(T)$ is t capacity, T is the temperature, and $k(T)$ is the heat conduction co $T(x, 0) = I(x)$, and ends are subject to a cooling law:

$$k(T)T_x|_{x=0} = h(T)(T - T_s), \quad -k(T)T_x|_{x=L} = h(T)(T - T_s),$$

where $h(T)$ is a heat transfer coefficient and T_s is the given surrounding ature.

a) Discretize this PDE in time using either a Backward Euler or Crank-N scheme.

b) Formulate a Picard iteration method for the time-discrete problem iteration method before discretizing in space).

c) Formulate a Newton method for the time-discrete problem in b).

d) Discretize the PDE by a finite difference method in space. Derive the and right-hand side of a Picard iteration method applied to the spa discretized PDE.

e) Derive the matrix and right-hand side of a Newton method applie discretized PDE in d).
Filename: `nonlin_1D_heat_FD.pdf`.

Exercise 13: Use different symbols for different approximations of the solution

The symbol u has several meanings, depending on the context, as briefly mentioned in Section 5.1. Go through the derivation of the Picard iteration method in that section and use different symbols for all the different approximations of :

- $u_{\mathrm{e}}(\boldsymbol{x},t)$ for the exact solution of the PDE problem
- $u_{\mathrm{e}}(\boldsymbol{x})^n$ for the exact solution after time discretization
- $u^n(\boldsymbol{x})$ for the spatially discrete solution $\sum_j c_j\psi_j$
- $u^{n,k}$ for approximation in Picard/Newton iteration no k to $u^n(\boldsymbol{x})$

Filename: `nonlin_heat_FE_usymbols.pdf`.

Exercise 14: Derive Picard and Newton systems from a variational form

We study the multi-dimensional heat conduction PDE

$$\varrho c(T)T_t = \nabla\cdot(k(T)\nabla T)$$

in a spatial domain Ω, with a nonlinear Robin boundary condition

$$-k(T)\frac{\partial T}{\partial n} = h(T)(T - T_s(t)),$$

at the boundary $\partial\Omega$. The primary unknown is the temperature T, ϱ is the density of the solid material, $c(T)$ is the heat capacity, $k(T)$ is the heat conduction, $h(T)$ is a heat transfer coefficient, and $T_s(T)$ is a possibly time-dependent temperature of the surroundings.

) Use a Backward Euler or Crank-Nicolson time discretization and derive the variational form for the spatial problem to be solved at each time level.

) Define a Picard iteration method from the variational form at a time level.

) Derive expressions for the matrix and the right-hand side of the equation system that arises from applying Newton's method to the variational form at a time level.

) Apply the Backward Euler or Crank-Nicolson scheme in time first. Derive Newton method at the PDE level. Make a variational form of the resulting PDE at a time level.
Filename: `nonlin_heat_FE.pdf`.

Exercise 15: Derive algebraic equations for nonlinear heat conduction

We consider the same problem as in Exercise 14, but restricted to one spatial dimension: $\Omega = [0, L]$. Simplify the boundary condition to $T_x = 0$ (i.e., $h($ Use a uniform finite element mesh of P1 elements, the group finite method, and the Trapezoidal rule for integration at the nodes to derive algebraic expressions for the algebraic equations arising from this diffusion problem.
Filename: `nonlin_1D_heat_FE.pdf`.

Exercise 16: Differentiate a highly nonlinear term

The operator $\nabla\cdot(\alpha(u)\nabla u)$ with $\alpha(u) = |\nabla u|^q$ appears in several problems, especially flow of Non-Newtonian fluids. The expression is defined as the Eucledian norm of a vector: $|\nabla u|^2 = \nabla u\cdot\nabla u$. In a method one has to carry out the differentiation $\partial\alpha(u)/\partial c_j$, for $u = \sum$ Show that

$$\frac{\partial}{\partial u_j}|\nabla u|^q = q|\nabla u|^{q-2}\nabla u\cdot\nabla\psi_j\,.$$

Filename: `nonlin_differentiate.pdf`.

Exercise 17: Crank-Nicolson for a nonlinear 3D diffusion equation

Redo Section 5.2 when a Crank-Nicolson scheme is used to discretize the equation in time and the problem is formulated for three spatial dimensions.

Hint. Express the Jacobian as $J_{i,j,k,r,s,t} = \partial F_{i,j,k}/\partial u_{r,s,t}$ and observe the 2D case, that $J_{i,j,k,r,s,t}$ is very sparse: $J_{i,j,k,r,s,t} \neq 0$ only for r $s = j\pm 1$, and $t = k\pm 1$ as well as $r = i$, $s = j$, and $t = k$.
Filename: `nonlin_heat_FD_CN_2D.pdf`.

Exercise 18: Find the sparsity of the Jacobian

Consider a typical nonlinear Laplace term like $\nabla\cdot\alpha(u)\nabla u$ discretized by finite differences. Explain why the Jacobian corresponding to this term has the same sparsity pattern as the matrix associated with the corresponding term $\alpha\nabla^2 u$.

Hint. Set up the unknowns that enter the difference equation at a point in 2D or (i, j, k) in 3D, and identify the nonzero entries of the Jacobian that arise from such a type of difference equation.
Filename: `nonlin_sparsity_Jacobian.pdf`.

Problem 19: Investigate a 1D problem with a continuation method

Flow of a pseudo-plastic power-law fluid between two flat plates can be modeled by

$$\frac{d}{dx}\left(\mu_0\left|\frac{du}{dx}\right|^{n-1}\frac{du}{dx}\right) = -\beta, \quad u'(0) = 0,\ u(H) = 0,$$

where $\beta > 0$ and $\mu_0 > 0$ are constants. A target value of n may be $n = 0.2$.

a) Formulate a Picard iteration method directly for the differential equation problem.

b) Perform a finite difference discretization of the problem in each Picard iteration. Implement a solver that can compute u on a mesh. Verify that the solver gives an exact solution for $n = 1$ on a uniform mesh regardless of the cell size.

c) Given a sequence of decreasing n values, solve the problem for each n using the solution for the previous n as initial guess for the Picard iteration. This is called a continuation method. Experiment with $n = (1, 0.6, 0.2)$ and $n = (1, 0.9, 0.8, \ldots, 0.2)$ and make a table of the number of Picard iterations versus n.

d) Derive a Newton method at the differential equation level and discretize the resulting linear equations in each Newton iteration with the finite difference method.

e) Investigate if Newton's method has better convergence properties than Picard iteration, both in combination with a continuation method.

References

[1] C. T. Kelley. *Iterative Methods for Linear and Nonlinear Equations*. SIAM, 1995.

[2] M. Mortensen, H. P. Langtangen, and G. N. Wells. A FEniCS-based programming framework for modeling turbulent flow by the Reynolds-averaged Navier-Stokes equations. *Advances in Water Resources*, 34(9), 2011.

Index