Nonlinear differential equation problems

Hans Petter Langtangen[1,2]

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

Nov 29, 2013

Note: **VERY PRELIMINARY VERSION** (expect typos and mathematical errors)

Contents

List of Exercises and Problems

In a linear differential equation all terms involving the unknown functions are linear in the unknown functions or their derivatives. Linear here means that the unknown function or a derivative of it is multiplied by a number or a known function. All other differential equations are non-linear. The easiest way to see if an equation is nonlinear is to spot nonlinear terms where the unknown functions or their derivatives are multiplied by each other. For example, in

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

the terms involving the unknown function u are linear: u' contains the derivative of the unknown function multiplied by unity, and au contains the unknown 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 multiplied 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 appears in a product with itself or one if its derivatives. Another example of a nonlinear equation is

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

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

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

A series of forthcoming examples will explain who to tackle nonlinear differential equations with various techniques.

1 Basic examples using the logistic equation

Consider the (scaled) logistic equation

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

This is a nonlinear differential equation which will be solved by different strategies in the following. A time discretization of (1) will either lead to a linear algebraic equation or a nonlinear algebraic equation at each time level. In the former case, the time discretization method transforms the nonlinear ODE into linear subproblems at each time level, and the solution is straightforward to find. However, when the time discretization leads to nonlinear algebraic equations, we cannot (except in very rare cases) solve these without turning to approximate, iterative solution methods

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}. Therefore, the nonlinearity in the original equation poses no difficulty in the discrete algebraic equation. Any other explicit scheme in time will also give only linear algebraic equations to solve. For example, a typical 2nd-order Runge-Kutta method for (1) reads,

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

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

1.2 Exact solution of nonlinear equations

Switching to a Backward Euler scheme for (1),

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

results in a nonlinear algebraic equation for the unknown value u^n. The equation 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 nonlinear algebraic equation that appear at a given time level. The notation gets rid of the superscript that indicates the time level and is motivated by how we will program the solution method for the algebraic equation, especially in more advanced partial differential equation problems. The unknown in the algebraic equation is denoted by u, while u_1 is the value of the unknown at the previous time level (in general u_ℓ is the value of the unknown ℓ levels back in time). The quadratic equation for the unknown u^n in (2) can then be written

$$F(u) = \Delta t u^2 + (1 - \Delta t)u - u_1 = 0, \tag{3}$$

and the solution is

$$u = \frac{1}{2\Delta t}\left(-1 - \Delta t \pm \sqrt{(1 - \Delta t)^2 - 4\Delta t u_1}\right) \,. \tag{4}$$

Here we encounter a fundamental challenge with nonlinear algebraic equations: the equation may have more than one solution. How do we pick the right solution? In the present simple case we can expand the square root in a series in Δt and truncate after the linear term since the Backward Euler scheme will introduce an error proportional to Δt anyway. Using `sympy` we find the following Taylor series 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 root in (4), behaves as $1/\Delta t$ and will therefore blow up as $\Delta t \rightarrow 0$! Only the `r2` root is of relevance in this case.

1.3 Linearization

When the time integration of an ODE results in a nonlinear algebraic equation, we must normally find its solution by defining a sequence of linear equations and hope that the solutions of these linear equations converge to the desired solution of the nonlinear algebraic equation. Usually this means solving the linear equation repeatedly in an iterative fashion. Sometimes the nonlinear equation is just approximated by a linear equation and no iteration is carried out.

Constructing a linear equation from a nonlinear one requires *linearization* of each nonlinear term. This can be done manually as in Picard iteration, or fully algorithmically as in Newton's method. Examples will best illustrate how 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_- an available approximation of the unknown u. Then we can linearize the term u^2 by writing $u_- u$. The resulting equation, $\hat{F}(u) = 0$, is 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 equal the exact solution $u_{\rm e}$ of the exact equation $F(u_{\rm e}) = 0$, but we can hope that u is closer to $u_{\rm e}$ than u_- is, and hence it makes sense to repeat the procedure, i.e., set $u_- = u$ and solve $\hat{F}(u) = 0$ again.

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

Picard iteration for solving the nonlinear equation arising from the Backward 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 time level: $u_- = u_1$.

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

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

However, we will normally apply a mathematical notation in our final formulas that is as close as possible to what we aim to write in a computer code and then we want to omit the k superscript in u.

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

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

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

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

With $\epsilon_r = 10^{-7}$ we seldom need more than about 5 iterations when solving this logistic equation.

A single Picard iteration. Instead of iterating until a stopping criterion is fulfilled, one may iterate a specific number of times. Just one Picard iteration is popular as this corresponds to the intuitive idea of approximating a nonlinear term like $(u^n)^2$ by $u^{n-1}u^n$. That is, one just applies a known value for the unknown at the previous time level in nonlinear terms. The corresponding time discretization reads

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

This is obviously an approximation and does not correspond to a "pure" finite difference method where the equation is sampled at a point and derivatives replaced by differences. The best interpretation of the scheme (5) is a Backward Euler difference combined with a single Picard iteration at each time level, using the value at the previous time level as start for the Picard iteration.

1.5 Linearization by a geometric mean

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

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

written 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 levels where we actually compute it. The same arithmetic mean applied to the nonlinear 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 mean* for $(u^{n+\frac{1}{2}})^2$ is a way of linearizing the nonlinear term in (6):

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

The linearized scheme for u^{n+1} now reads

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

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 algebraic equation to solve by methods for those kind of problems we skip the simplified notation (u for u^{n+1} and u_1 for u^n).

The geometric mean approximation is often very effective to deal with quadratic nonlinearities. Both the arithmetic and geometric mean approximations have truncation errors of order Δt^2 and are therefore compatible with the truncation error of the centered difference approximation for U' in the Crank-Nicolson method.

Applying the operator notation for the means, the linearized Crank-Nicolson scheme for the logistic equation can be compactly 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 nonlinear term in (6), we end up with a nonlinear term $(u^{n+1})^2$. The term can be linearized as $u^n u^{n+1}$ in a Picard iteration approach. Observe that the geometric mean avoids any iteration.

1.6 Newton's method

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

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

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

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

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

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

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

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

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

$$F(u) = au^2 + bu + c, \quad a = \Delta t,\ b = 1 - \Delta t,\ c = -u_1,$$

and then

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

The iteration method becomes

$$u = u_- + \frac{au_-^2 + bu_- + c}{2au_- + b}, \quad u_- \leftarrow u\,. \tag{7}$$

At each time level, we start the iteration by setting $u_- = u_1$. Stopping criteria 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, and use a time level counter n and an iteration counter k, takes the form

$$u^{n,k+1} = u^{n,k} + \frac{\Delta t(u^{n,k})^2 + (1-\Delta t)u^{n,k} - u^{n-1}}{2\Delta t u^{n,k} + 1 - \Delta t}, \quad u^{n,0} = u^{n-1}, \quad k = 0, 1, \ldots \tag{8}$$

The implementation is much closer to (7) than to (8), but the latter is better aligned with the established mathematical notation used in the literature.

1.7 Relaxation

One iteration in Newton's method or Picard iteration consists of solving a linear problem $\hat{F}(u) = 0$. Sometimes convergence problems arise because the new solution u of $\hat{F}(u) = 0$ is "too far away" from the previously computed solution u_-. A remedy is to introduce a relaxation, meaning that we first solve $\hat{F}(u^*) = 0$ for a suggested value u^* and then we take u as a weighted mean of what we had, 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 $\omega < 1$ may prevent divergent iterations.

Relaxation in Newton's method can be directly incorporated in the basic 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 methods described above. Below is an extract of the file showing how the Picard and Newton methods are implemented for a Backward Euler discretization of the logistic equation.

```
def BE_logistic(u0, dt, Nt, choice='Picard', eps_r=1E-3, omega=1):
    u = np.zeros(Nt+1)
    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:
                u_ = omega*(-c/(a*u_ + b)) + (1-omega)*u_
                k += 1
            u[n] = u_
        elif choice == 'Newton':
```

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

```
            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:
                u_ = u_ - F(u_)/dF(u_)
                k += 1
            u[n] = u_
    return u
```

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

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

Experiments with this program reveal the relative performance of the methods as summarized in the table below. The Picard and Newton columns reflect the typical number of iterations with these methods before the curve starts to flatten out and the number of iterations is significantly reduced since the solution of the nonlinear algebraic equation is very close to the starting value for the iterations (the solution at the previous time level). Increasing Δt moves the starting value further away from the solution of the nonlinear equation and one expects an increase in the number of iterations. Picard iteration is very much more sensitive to the size of Δt than Newton's method. The tolerance ϵ_r in residual-based stopping criterion takes on a low and high value in the experiments.

Δt	ϵ_r	Picard	Newton
0.2	10^{-7}	5	2
0.2	10^{-3}	2	1
0.4	10^{-7}	12	3
0.4	10^{-3}	4	2
0.8	10^{-7}	58	3
0.8	10^{-3}	4	2

Remark. The simple Crank-Nicolson method with a geometric mean for the quadratic nonlinearity gives visually more accurate solutions than the Backward Euler discretization. Even with a tolerance of $\epsilon_r = 10^{-3}$, all the methods for treating the nonlinearities in the Backward Euler discretization gives graphs that cannot be distinguished. So for accuracy in this problem, the time discretization is much more crucial than ϵ_r. Ideally, one should estimate the error in the time 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 applied to the more generic model

$$u' = f(u,t), \tag{9}$$

where f is a nonlinear function of u.

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

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

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

or alternatively

$$F(u) = u - \Delta t f(u, t_n) - u_1 = 0\,.$$

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

$$\hat{F}(u) = u - \Delta t f(u_-, t_n) - u_1\,.$$

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.

Newton's method 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

$$u = u_- - \omega \frac{F(u_-)}{F'(u_-)} = u_- \omega \frac{u_1 + \Delta t f(u,t_n)}{1 - \Delta t \frac{\partial}{\partial u} f(u,t_n)}\,.$$

The geometric mean trick cannot be used unless we know that f has a special structure with quadratic expressions in u.

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

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

Introducing u for the unknown u^{n+1} and u_1 for u^n, we can write the scheme as a nonlinear algebraic equation

$$F(u) = u - u_1 - \Delta t\frac{1}{2}f(u, t_{n+1}) - \Delta t\frac{1}{2}f(u_1, t_n) = 0\,.$$

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, but we need to derive

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

2 Systems of nonlinear algebraic equations

Implicit time discretization methods for a system of ODEs, or a PDE, lead to *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 function: $F = (F_0, \ldots, F_N)$. Sometimes the equation system has a special structure because of the underlying 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 function: $b = (b_0, \ldots, b_N)$.

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

2.1 Picard iteration

We cannot apply Picard iteration to nonlinear equations unless there is some special structure. For the commonly arising case $A(u)u = b(u)$ we can linearize the product $A(u)u$ to $A(u_-)u$ and $b(u)$ as $b(u_-)$. That is, we use the most previously computed approximation in A and b to arrive at a *linear system* for 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 sequence of linear systems.

Algorithm for relaxed Picard iteration.
Given $A(u)u = b(u)$ and an initial guess u_-, iterate until convergence:

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

2.2 Newton's method

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

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

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

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

So, the original nonlinear system is approximated by

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

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

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

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

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

and

$$J_{i,j} = \sum_k \frac{\partial A_{i,k}}{\partial u_j} u_k + A_{i,j} - \frac{\partial b_i}{\partial u_j}\,. \qquad (10)$$

We realize that the Jacobian needed in Newton's method consists of $A(u_-)$ as in the Picard iteration plus two additional terms arising from the differentiation. Using the notation $A'(u)$ for $\partial A/\partial u$ (a quantity with three indices: $\partial A_{i,k}/\partial u_j$), and $b'(u)$ for $\partial b/\partial u$ (a quantity with two indices: $\partial b_i/\partial u_j$), we can write the linear system to be solved as

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

or

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

Rearranging the terms demonstrates the difference from the system solved in each Picard iteration:

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

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

2.3 Stopping criteria

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

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

To prevent divergent iterations to run forever, one terminates the iterations when 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 the initial start value for the iteration is very close to the solution, since an unnecessary reduction in the error measure is enforced. In such cases the absolute criteria work better. It is common to combine the absolute and relative measures of the size of the residual, as in

$$||F(u)|| \leq \epsilon_{rr}||F(u_0)|| + \epsilon_{ra}, \tag{11}$$

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

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

$$||\delta u|| \leq \epsilon_{ur}||u_0|| + \epsilon_{ua}, \tag{12}$$

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

$$||F(u)|| \leq \epsilon_{rr}||F(u_0)|| + \epsilon_{ra} \text{ or } ||\delta u|| \leq \epsilon_{ur}||u_0|| + \epsilon_{ua} \text{ or } k > k_{\max}\,. \tag{13}$$

2.4 Example: A nonlinear ODE model from epidemiology

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

$$S' = -\beta SI, \tag{14}$$

$$I' = \beta SI - \nu I, \tag{15}$$

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

Implicit time discretization. A Crank-Nicolson scheme leads to a 2×2 system of nonlinear algebraic equations in the unknowns S^{n+1} and I^{n+1}:

$$\frac{S^{n+1} - S^n}{\Delta t} = -\beta[SI]^{n+\frac{1}{2}} \approx -\frac{\beta}{2}(S^nI^n + S^{n+1}I^{n+1}), \tag{16}$$

$$\frac{I^{n+1} - I^n}{\Delta t} = \beta[SI]^{n+\frac{1}{2}} - \nu I^{n+\frac{1}{2}} \approx \frac{\beta}{2}(S^nI^n + S^{n+1}I^{n+1}) - \frac{\nu}{2}(I^n + I^{n+1})\,. \tag{17}$$

Introducing S for S^{n+1}, S_1 for S^n, I for I^{n+1}, I_1 for I^n, we can rewrite the system as

$$F_S(S,I) = S - S_1 + \frac{1}{2}\Delta t\beta(S_1 I_1 + SI) = 0, \tag{18}$$

$$F_I(S,I) = I - I_1 - \frac{1}{2}\Delta t\beta(S_1 I_1 + SI) - \frac{1}{2}\Delta t\nu(I_1 + I) = 0\,. \tag{19}$$

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

$$S = \frac{S_1 - \frac{1}{2}\Delta t\beta S_1 I_1}{1 + \frac{1}{2}\Delta t\beta I_-},$$

$$I = \frac{I_1 + \frac{1}{2}\Delta t\beta S_1 I_1}{1 - \frac{1}{2}\Delta t\beta S_- + \nu}\,.$$

The solutions S and I are stored in S_- and I_- and a new iteration is carried out.

Newton's method. The nonlinear system (18)-(19) can be written as $F(u) = 0$ with $F = (F_S, F_I)$ and $u = (S, I)$. The Jacobian becomes

$$J = \left(\begin{array}{cc} \frac{\partial}{\partial S}F_S & \frac{\partial}{\partial I}F_S \\ \frac{\partial}{\partial S}F_I & \frac{\partial}{\partial I}F_I \end{array}\right) = \left(\begin{array}{cc} 1 + \frac{1}{2}\Delta t\beta I & \frac{1}{2}\Delta t\beta \\ -\frac{1}{2}\Delta t\beta S & 1 - \frac{1}{2}\Delta t\beta I - \frac{1}{2}\Delta t\nu \end{array}\right).$$

The Newton system to be solved in each iteration is then

$$\left(\begin{array}{cc} 1 + \frac{1}{2}\Delta t\beta I_- & \frac{1}{2}\Delta t\beta S_- \\ -\frac{1}{2}\Delta t\beta S_- & 1 - \frac{1}{2}\Delta t\beta I_- - \frac{1}{2}\Delta t\nu \end{array}\right)\left(\begin{array}{c} \delta S \\ \delta I \end{array}\right) =$$
$$\left(\begin{array}{c} S_- - S_1 + \frac{1}{2}\Delta t\beta(S_1 I_1 + S_- I_-) \\ I_- - I_1 - \frac{1}{2}\Delta t\beta(S_1 I_1 + S_- I_-) - \frac{1}{2}\Delta t\nu(I_1 + I_-) \end{array}\right)$$

Remark. For this particular system explicit time integration methods work very well. The 4-th order Runge-Kutta method is an excellent balance between high accuracy, high efficiency, and simplicity.

3 Linearization at the differential equation level

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

$$\frac{\partial u}{\partial t} = \nabla\cdot(\alpha(u)\nabla u) + f(u), \qquad \boldsymbol{x}\in\Omega,\ t\in(0,T], \tag{20}$$

$$-\alpha(u)\frac{\partial u}{\partial n} = g, \qquad \boldsymbol{x}\in\partial\Omega_N,\ t\in(0,T], \tag{21}$$

$$u = u_0, \qquad \boldsymbol{x}\in\partial\Omega_D,\ t\in(0,T]\,. \tag{22}$$

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

3.1 Explicit time integration

The nonlinearities in the PDE are trivial to deal with if we choose an explicit time integration method for (20), such as the Forward Euler method:

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

which leads to a linear equation in the unknown u^{n+1}:

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

3.2 Picard iteration

A Backward Euler scheme for (20) reads

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

Written out,

$$\frac{u^{n}-u^{n-1}}{\Delta t} = \nabla\cdot(\alpha(u^n)\nabla u^n) + f(u^n) \tag{23}$$

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

$$\frac{u^{n,k+1}-u^{n-1}}{\Delta t} = \nabla\cdot(\alpha(u^{n,k})\nabla u^{n,k+1}) + f(u^{n,k})\,. \tag{24}$$

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

We can alternatively apply the notation where u corresponds to the unknown we want to solve for, i.e., $u^{n,k+1}$ above, let u_- be the most recently computed value, $u^{n,k}$ above, and let u_1 denote the unknown function at the previous time level, u^{n-1} above. The PDE to be solved in a Picard iteration then looks like

$$\frac{u - u_1}{\Delta t} = \nabla \cdot (\alpha(u_-)\nabla u) + f(u_-)\,. \tag{25}$$

At the beginning of the iteration we start with the value from the previous time level: $u_- = u_1$.

3.3 Newton's method

At time level n we have to solve the stationary PDE (23), this time with Newton's method. Normally, Newton's method is defined for systems of *algebraic equations*, but the idea of the method can be applied at the PDE level too.

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

$$u^n = u^{n,k} + \delta u\,. \tag{26}$$

The idea is to insert (26) in (23), Taylor expand the nonlinearities and only keep the terms that are linear in δu. Then we can solve a linear PDE for the correction δu and use (26) to find a new approximation $u^{n,k+1} = u^{n,k} + \delta u$ to u^n.

Inserting (26) in (23) gives

$$\frac{u^{n,k} + \delta u - u^{n-1}}{\Delta t} = \nabla \cdot (\alpha(u^{n,k} + \delta u)\nabla(u^{n,k} + \delta u)) + f(u^{n,k} + \delta u) \tag{27}$$

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

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

Inserting the linear approximations of α and f in (27) results in

$$\begin{aligned}
\frac{u^{n,k} + \delta u - u^{n-1}}{\Delta t} &= \nabla \cdot (\alpha(u^{n,k})\nabla u^{n,k}) + f(u^{m,k}) +\\
&\quad \nabla \cdot (\alpha(u^{n,k})\nabla \delta u) + \nabla \cdot (\alpha'(u^{n,k})\delta u \nabla u^{n,k}) +\\
&\quad \nabla \cdot (\alpha'(u^{n,k})\delta u \nabla \delta u) + f'(u^{n,k})\delta u
\end{aligned} \tag{28}$$

The term $\alpha'(u^{n,k})\delta u\nabla\delta u$ is $\mathcal{O}(\delta u^2)$ and therefore omitted. Reorganizing the equation 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

$$\begin{aligned} F(u^{n,k}) &= \frac{u^{n,k} - u^{n-1}}{\Delta t} - \nabla\cdot(\alpha(u^{n,k})\nabla u^{n,k}) + f(u^{n,k}), \qquad (29)\\ \delta F(\delta u; u^{n,k}) &= -\frac{1}{\Delta t}\delta u + \nabla\cdot(\alpha(u^{n,k})\nabla\delta u) + \\ &\quad \nabla\cdot(\alpha'(u^{n,k})\delta u\nabla u^{n,k}) + f'(u^{n,k})\delta u\,. \qquad (30)\end{aligned}$$

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

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

We can rewrite the PDE for δu in a slightly different way too if we define $u^{n,k} + \delta u$ as $u^{n,k+1}$.

$$\begin{aligned} \frac{u^{n,k+1} - u^{n-1}}{\Delta t} &= \nabla\cdot(\alpha(u^{n,k})\nabla u^{n,k+1}) + f(u^{n,k}) + \\ &\quad \nabla\cdot(\alpha'(u^{n,k})\delta u\nabla u^{n,k}) + f'(u^{n,k})\delta u\,. \qquad (31)\end{aligned}$$

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

For coding we want to introduce u_- for $u^{n,k}$ and u_1 for u^{n-1}. The formulas for F and δF are then

$$\begin{aligned} F(u_-) &= \frac{u_- - u_1}{\Delta t} - \nabla\cdot(\alpha(u_-)\nabla u_-) + f(u_-), \qquad (32)\\ \delta F(\delta u; u_-) &= -\frac{1}{\Delta t}\delta u + \nabla\cdot(\alpha(u_-)\nabla\delta u) + \\ &\quad \nabla\cdot(\alpha'(u_-)\delta u\nabla u_-) + f'(u_-)\delta u\,. \qquad (33)\end{aligned}$$

The form that orders the PDE as the Picard iteration terms plus the Newton method's derivative terms becomes

$$\begin{aligned} \frac{u - u_1}{\Delta t} &= \nabla\cdot(\alpha(u_-)\nabla u) + f(u_-) + \\ &\quad \nabla\cdot(\alpha'(u_-)\delta u\nabla u_-) + f'(u_-)\delta u\,. \qquad (34)\end{aligned}$$

4 Discretization of nonlinear differential equations

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

We shall work with the 1D problem

$$-(\alpha(u)u')' + au = f(u), \quad x \in (0,L), \quad \alpha(u(0))u'(0) = C,\ u(L) = 0\,. \tag{35}$$

This problem is of the same nature as those arising from implicit time integration of a nonlinear diffusion PDE as outlined in Section 3.2 (set $a = 1/\Delta t$ and let $f(u)$ incorporate the nonlinear source term as well as known terms with the time-dependent unknown function at the previous time level).

4.1 Finite difference discretizations

Discretization. The nonlinearity in the differential equation (35) poses no more difficulty than a variable coefficient, as in $(\alpha(x)u')'$. We can therefore use a standard approach to discretizing the Laplace term with a variable coefficient:

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

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

$$-\frac{1}{\Delta x^2}\left(\alpha_{i+\frac{1}{2}}(u_{i+1} - u_i) - \alpha_{i-\frac{1}{2}}(u_i - u_{i-1})\right) + au_i = f(u_i)\,. \tag{36}$$

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

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

$$\alpha_{i+\frac{1}{2}} \approx \alpha(\frac{1}{2}(u_i + u_{i+1}) = [\alpha(\overline{u}^x)]^{i+\frac{1}{2}}, \tag{37}$$

$$\alpha_{i+\frac{1}{2}} \approx \frac{1}{2}(\alpha(u_i) + \alpha(u_{i+1})) = [\overline{\alpha(u)}^x]^{i+\frac{1}{2}} \tag{38}$$

Equation (36) with the latter approximation then looks like

$$
\begin{aligned}
-\frac{1}{2\Delta x^2}&\left((\alpha(u_i)+\alpha(u_{i+1}))(u_{i+1}-u_i)-(\alpha(u_{i-1})+\alpha(u_i))(u_i-u_{i-1})\right)\\
&+au_i = f(u_i),
\end{aligned}
\tag{39}
$$

or 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 discretized by

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

meaning

$$
\alpha(u_0)\frac{u_1-u_{-1}}{2\Delta x} = C\,. \tag{40}
$$

The fictitious value u_{-1} can be eliminated with the aid of (39) for $i=0$. Formally, (39) should be solved with respect to u_{i-1} and that value (for $i=0$) should be inserted in (40), but it is algebraically much easier to do it the other way around. Alternatively, one can use a ghost cell $[-\Delta x, 0]$ and update the u_{-1} value in the ghost cell according to (40) after every Picard or Newton iteration. Such an approach means that we use a known u_{-1} value in (39) from the previous iteration.

Solution of algebraic equations. The nonlinear algebraic equations (39) 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 obvious Picard iteration scheme is to use previously computed values of u_i in $A(u)$ and $b(u)$, as described more in detail in Section 2.

Newton's method requires computation of the Jacobian. Here it means that we need to differentiate $F(u) = A(u)u - b(u)$ with respect to $u_0, u_1, \ldots, u_{N_x-1}$. Nonlinear equation number i has the structure

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

The Jacobian becomes

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

The explicit expression for nonlinear equation number i, $F_i(u_0, u_1, \ldots)$, arises from moving all terms in (39) to the left-hand side. Then we have $J_{i,j}$ and F_i (modulo the boundary conditions) and can implement Newton's method.

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

4.2 Finite element discretizations

For the finite element discretization we first need to derive the variational problem. Let V be an appropriate function space with basis functions $\{\psi_i\}_{i\in\mathcal{I}_s}$. Because of the Dirichlet condition at $x = L$ we require $\psi_i(L) = 0$, $i \in \mathcal{I}_s$. Using Galerkin's method, we multiply the differential equation by any $v \in V$, integrate terms with second-order derivatives by parts, and insert the Neumann condition at $x = 0$. The variational problem is then: find $u \in V$ such that

$$
\int_0^L \alpha(u)u'v'\,\mathrm{d}x + \int_0^L auv\,\mathrm{d}x = \int_0^L f(u)v\,\mathrm{d}x - Cv(0), \quad \forall v \in V\,. \tag{41}
$$

To derive the algebraic equations we also demand the above equations to hold for $v = \psi_i$, $i \in \mathcal{I}_s$, and we set $u = \sum_{j\in\mathcal{I}_s} c_j\psi_j$. The result is

$$
\sum_{j\in\mathcal{I}_s}\left(\int_0^L \alpha(\sum_{k\in\mathcal{I}_s} c_k\psi_k)\psi_j'\psi_i'\,\mathrm{d}x\right)c_j = \int_0^L f(\sum_{k\in\mathcal{I}_s} c_k\psi_k)\psi_i\,\mathrm{d}x - C\psi_i(0), \quad i \in \mathcal{I}_s\,. \tag{42}
$$

Remark. Fundamental integration problem Methods that use the Galerkin or weighted residual principle face a fundamental difficulty in nonlinear problems: 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 $\int_0^L f(\sum_k c_k\psi_k)\psi_i\,\mathrm{d}x$ when we do not know the c_k coefficients in the argument of the α function? We can resort to numerical integration, provided an approximate $\sum_k c_k\psi_k$ can be used for the argument u in f and α. If we want to derive the structure of the nonlinear algebraic equations, we need to apply numerical integration based on the nodes only and/or the group finite element method.

4.3 The group finite element method

Finite element notation. Let us simplify the model problem for a while and set $a = 0$, $\alpha = 1$, $f(u) = u^2$, and have Dirichlet conditions at both ends such that we get a very simple nonlinear problem $-u'' = u^2$. The variational form is then

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

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

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 $\nu(j)$ in the mesh corresponds to unknown number j. When the degrees of freedom are just the function values at nodes, we have that $c_j = u(x_{\nu(j)}) = u_{\nu(j)}$, i.e., the value of u at node number $\nu(j)$. The finite element expansion for u is now

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

with the U_j quantities being prescribed Dirichlet values at some nodes with numbers in the index I_b. Instead of the $\nu(j)$ indices in the sum $\sum_{j\in\mathcal{I}_s}\varphi_{\nu(j)}u_{\nu(j)}$, we just write $\sum_j \varphi_j u_j$. This is possible by saying that j runs over a transformed index set: $\{\nu(0), \nu(1), \ldots, \nu(N)\}$. In the following we drop the boundary term $\sum_j U_j\varphi_j$ and write $u = \sum_j \varphi_j u_j$. The replacement of c_j by u_j as explained is motivated by simpler interpretation of the nonlinear algebraic equations as a finite difference scheme.

Integrating nonlinear functions. Consider the term $\int u^2 v\,\mathrm{d}x$ in the variational 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 9) results in the expression

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

to be compared with the simple value u_i^2 that would arise in a finite difference discretization. More complicated $f(u)$ functions give rise to much more lengthy expressions, if it is possible to carry out the integral symbolically.

Finite element approximation of functions of u. Since we already expand u as $\sum_j \varphi_j u_j$ we may use the same approximation for nonlinearities. That is, any function can be expanded as a sum of basis functions times the function values. In particular,

$$f(u) \approx \sum_{j\in I_b} \varphi_j f(u_j) + \sum_j \psi_j(x) f(u_j),$$

where the first sum contain f values at the boundary where u has Dirichlet conditions and the other sum is over the node values j where u is unknown. However, for f there is no reason two have two summations as we do not need to distinguish between the nodes where u are known or unknown. Therefore, we can collapse the two sums into one (over all nodes, $j = 0, \ldots, N_n$) and write

$$f(u) \approx \sum_{j=0}^{N_n} \varphi_j(x) f(u_j)\,. \tag{43}$$

This approximation is known as the *group finite element method* or the *product approximation* technique.

The principal advantage of the group finite element method is for deriving the *symbolic form* of difference equations in nonlinear problems. The symbolic form is useful for comparing finite element and finite difference equations of non-linear differential equation problems. Computer programs will always integrate $\int f(u)\varphi_i \,\mathrm{d}x$ numerically by using an existing approximation of u in $f(u)$ such that the integrand can be sampled at any spatial point.

Application. Let use the group finite element method to derive the terms in the difference equation corresponding to $f(u)$ 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 $\int \varphi_i\varphi_j \,\mathrm{d}x$, times the vector $f = (f(u_0), f(u_1), \ldots,)$: Mf. The associated terms in the difference equations are

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

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

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

We may lump the mass matrix through integration with the Trapezoidal rule. In that case the $f(u)$ term in the differential equation gives rise to a single term $hf(u_i)$, just as in the finite difference method.

4.4 Numerical integration of nonlinear terms

Let us reconsider a term $\int f(u)v\,\mathrm{d}x$ as treated in the previous section, but now we want to integrate this term numerically. Such an approach can lead to easy-to-interpret formulas if we apply a numerical integration rule that samples the integrand at the node points.

The term in question takes the form

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

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

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

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

where 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, we have

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

The term $\mathcal{C}$ contains the evaluations of the integrand at the ends with weight $\frac{1}{2}$, needed to make a true Trapezoidal rule. The answer $hf(u_i)$ must therefore be multiplied by $\frac{1}{2}$ if $i=0$ or $i=N_n$. ($\mathcal{C} = \frac{h}{2}f(u_0)\varphi_i(0) + \frac{h}{2}f(u_{N_n})\varphi_i(L)$.)

One can easily use the Trapezoidal rule on the reference cell and assemble the contributions. It is a bit more work in this context, but working on the reference cell is safer as that approach is guaranteed to handle discontinuous derivatives of finite element functions correctly.

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

4.5 Finite element discretization of a variable coefficient Laplace term

Turning back to the model problem (35), it remains to calculate the contribution of the $(\alpha u')'$ and boundary terms to the difference equations. The integral in 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 iteration must in general be used to compute the integral. Now our aim is to integrate symbolically, as much as we can, to obtain some insight into how the finite element method approximates this term.

To be able to derive symbolic expressions, we either turn to the group finite element method or numerical integration in the node points. Finite element basis functions φ_i are used, we set $\alpha(u) \approx \sum_k \alpha(u_k)\varphi_k$, and then we write

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

Further calculations are now easiest to carry out in the reference cell. With P1 elements we can compute $L_{i,j,k}$ for the two k values that are relevant on the reference cell. Turning to local indices, one gets

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

where $r,s,t=0,1$ are indices over local degrees of freedom in the reference cell ($i=q(e,r)$, $j=q(e,s)$, and $k=q(e,t)$). The sum $\sum_k L_{i,j,k}\alpha(u_k)$ at the cell 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 at local node number t in cell number e. The element matrix becomes

$$\frac{1}{2}(\alpha(\tilde u_0)+\alpha(\tilde u_1))\frac{1}{h}\left(\begin{array}{rr} 1 & -1 \\ -1 & 1 \end{array}\right). \tag{44}$$

As usual, we employ a left-to-right numbering of cells and nodes. Row number i in the global matrix gets contributions from the first row of the element matrix in cell $i-1$ and the last row of the element matrix in cell i. In cell number $i-1$ the sum $\alpha(\tilde u_0)+\alpha(\tilde u_1)$ corresponds to $\alpha(u_{i-1})+\alpha(u_i)$. The same becomes $\alpha(u_i)+\alpha(u_{i+1})$ in cell number i. We can with this insight assemble the contributions to row number i in the global matrix:

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

Multiplying by the vector of unknowns u_i results in

$$-\frac{1}{h}(\frac{1}{2}(\alpha(u_i)+\alpha(u_{i+1}))(u_{i+1}-u_i) - \frac{1}{2}(\alpha(u_{i-1})+\alpha(u_i))(u_i-u_{i-1})), \tag{45}$$

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

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:

$$\begin{aligned}
\int_{-1}^{1} \alpha(\sum_{t=0}^{1}\tilde u_t\tilde\varphi_t)\frac{2}{h}\frac{d\tilde\varphi_r}{dX}\frac{2}{h}\frac{d\tilde\varphi_s}{dX}\frac{h}{2}dX &= \frac{1}{2h}(-1)^r(-1)^s\int_{-1}^{1}\alpha(\sum_{t=0}^{1}u_t\tilde\varphi_t(X))dX\\
&\approx \frac{1}{2h}(-1)^r(-1)^s(\sum_{t=0}^{1}\tilde\varphi_t(-1)\tilde u_t + \sum_{t=0}^{1}\tilde\varphi_t(1)\tilde u_t)\\
&= \frac{1}{2h}(-1)^r(-1)^s(\alpha(\tilde u_0)+\alpha(\tilde u_1))\,. \qquad (46)
\end{aligned}$$

The element matrix in (46) is identical to the one in (44), 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: $[D_x\overline{x}D_xu]_i$.

We might comment on integration in the physical coordinate system too. The common Trapezoidal rule in Section 4.4 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 boundary: $Cv(0)=C\varphi_i(0)$. With a left-to-right numbering only $i=0$ will give a 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 $-h[D_x\overline{\alpha(u)}^x D_xu]_i$ if the group finite element method or Trapezoidal integration is applied,

- $f(u)$ becomes $hf(u_i)$ with Trapezoidal integration or the "mass matrix" representation $h[f(u) - \frac{h}{6}D_xD_xf(u)]_i$ if computed by a group 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 equations also in the finite element method, a Picard or Newton method can be defined as shown for the finite difference method. Nevertheless, the general situation is that we have not assembled finite difference-style equations by hand and the linear system in the Picard or Newton method must therefore be defined directly through the variational form, as shown next.

4.6 Picard iteration defined from the variational form

We address again the problem (35) with the corresponding variational form (41). Our aim is to define a Picard iteration based on this variational form without any attempt to compute integrals symbolically as in the previous three sections. The idea in Picard iteration is to use a previously computed u value in the nonlinear functions $\alpha(u)$ and $f(u)$. Let u_- be the available approximation to u from the previous iteration. The linearized variational form for Picard iteration is then

$$\int_0^L (\alpha(u_-)u'v' + auv)\,\mathrm{d}x = \int_0^L f(u_-)v\,\mathrm{d}x - Cv(0), \quad \forall v \in V\,. \tag{47}$$

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

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

The associated linear system is computed the standard way. Technically, we are back to solving $-(\alpha(x)u')' + au = f(x)$.

4.7 Newton's method defined from the variational form

Application of Newton's method to the nonlinear variational form (41) arising from the problem (35) requires identification of the nonlinear algebraic equations $F_i(c_0,\ldots,c_N) = 0$, $i \in \mathcal{I}_s$, and the Jacobian $J_{i,j} = \partial F_i/\partial c_j$ for $i,j \in \mathcal{I}_s$.

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

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

by 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 function to incorporate Dirichlet conditions.

With $v = \psi_i$ we have

$$F_i = \int_0^L (\alpha(u)u'\psi_i' + au\psi_i - f(u)\psi_i)\,\mathrm{d}x + C\psi_i(0) = 0, \quad i \in \mathcal{I}_s\,. \tag{48}$$

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

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

The derivation of the Jacobian goes as

$$\begin{aligned}
J_{i,j} = \frac{\partial F_i}{\partial c_j} &= \int_0^L \frac{\partial}{\partial c_j}(\alpha(u)u'\psi_i' + au\psi_i - f(u)\psi_i)\,\mathrm{d}x\\
&= \int_0^L ((\alpha'(u)\frac{\partial u}{\partial c_j}u' + \alpha(u)\frac{\partial u'}{\partial c_j})\psi_i' + a\frac{\partial u}{\partial c_j}\psi_i - f'(u)\frac{\partial u}{\partial c_j}\psi_i)\,\mathrm{d}x\\
&= \int_0^L ((\alpha'(u)\psi_j u' + \alpha(u)\psi_j'\psi_i' + a\psi_j\psi_i - f'(u)\psi_j\psi_i)\,\mathrm{d}x\\
&= \int_0^L (\alpha'(u)u'\psi_i'\psi_j + \alpha(u)\psi_i'\psi_j' + (a - f(u))\psi_i\psi_j)\,\mathrm{d}x
\end{aligned} \tag{49}$$

When calculating the right-hand side vector F_i and the coefficient matrix $J_{i,j}$ in the linear system to be solved in each Newton iteration, one must use a previously computed u, denoted by u_-, for the u in (48) and (51). With this notation we have

$$F_i = \int_0^L \left(\alpha(u_-)u_-'\psi_i' + (a - f(u_-))\psi_i\right)\,\mathrm{d}x - C\psi_i(0), \quad i \in \mathcal{I}_s, \tag{50}$$

$$J_{i,j} = \int_0^L (\alpha'(u_-)u_-'\psi_i'\psi_j + \alpha(u_-)\psi_i'\psi_j' + (a - f(u_-))\psi_i\psi_j)\,\mathrm{d}x, \quad i,j \in \mathcal{I}_s\,. \tag{51}$$

These expressions can be used for any basis $\{\psi_i\}_{i\in\mathcal{I}_s}$. Choosing finite element functions for ψ_i, one will normally want to compute the integral contribution cell by cell, working in a reference cell. To this end, we restrict the integration to one cell and transform the cell to $[-1, 1]$. The formulas (50) and (51) then change to

$$\tilde{F}_r^{(e)} = \int_{-1}^1 \left(\alpha(\tilde{u}_-)\tilde{u}_-'\tilde{\varphi}_r' + (a - f(\tilde{u}_-))\tilde{\varphi}_r\right)\det J\,\mathrm{d}X - C\tilde{\varphi}_r(0), \tag{52}$$

$$\tilde{J}_{r,s}^{(e)} = \int_{-1}^1 (\alpha'(\tilde{u}_-)\tilde{u}_-'\tilde{\varphi}_r'\tilde{\varphi}_s + \alpha(\tilde{u}_-)\tilde{\varphi}_r'\tilde{\varphi}_s' + (a - f(\tilde{u}_-))\tilde{\varphi}_r\tilde{\varphi}_s)\det J\,\mathrm{d}X, \tag{53}$$

with $r, s \in I_d$ runs over the local degrees of freedom. In the above formulas, $\tilde{u}_-(X) = \sum_r \tilde{c}_{-r}\tilde{\varphi}_r(X)$ is the finite element expansion of u_- over the current cell.

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

Dirichlet conditions. Incorporation of the Dirichlet values by assembling contributions from all degrees of freedom and then modifying the linear system can be obviously be applied to Picard iteration as that method involves a standard linear system. In the Newton system, however, the unknown is a correction δu to the solution. Dirichlet conditions are implemented by inserting them in the initial guess u_- for the Newton iteration and implementing $\delta u_i = 0$ for all known degrees of freedom. The manipulation of the linear system follows exactly the algorithm in the linear problems, the only difference being that the known values are zero.

5 Multi-dimensional PDE problems

5.1 Finite element discretization

The derivation of F_i and $J_{i,j}$ in the 1D model problem is easily generalized to multi-dimensional problems. For example, Backward Euler discretization of the PDE

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

gives the nonlinear time-discrete PDEs

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

or with u^n simply as u and u^{n-1} as u_1,

$$u - \Delta t\nabla\cdot(\alpha(u^n)\nabla u) - \Delta t f(u) = u_1\,.$$

The variational form, assuming homogeneous Neumann conditions for simplicity, becomes

$$\int_\Omega (uv + \Delta t\alpha(u)\nabla u\cdot\nabla v - \Delta t f(u)v - u_1 v)\,\mathrm{d}x\,. \tag{54}$$

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

$$F_i = \int_\Omega (u\psi_i + \Delta t\alpha(u)\nabla u\cdot\nabla\psi_i - \Delta t f(u)\psi_i - u_1\psi_i)\,\mathrm{d}x\,. \tag{55}$$

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

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

$$F_i \approx \hat{F}_i = \int_\Omega (u_-\psi_i + \Delta t\alpha(u_-)\nabla u\cdot\nabla\psi_i - \Delta t f(u_-)\psi_i - u_1\psi_i)\,\mathrm{d}x\,. \tag{56}$$

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

In Newton's method we need to evaluate F_i with the known value u_- for u:

$$F_i \approx \hat{F}_i = \int_\Omega (u_-\psi_i + \Delta t\alpha(u_-)\nabla u_-\cdot\nabla\psi_i - \Delta t f(u_-)\psi_i - u_1\psi_i)\,\mathrm{d}x\,. \tag{57}$$

The Jacobian is obtained by differentiating (55) and using $\partial u/\partial c_j = \psi_j$:

$$\begin{aligned} J_{i,j} = \frac{\partial F_i}{\partial c_j} = \int_\Omega & (\psi_j\psi_i + \Delta t\alpha'(u)\psi_j\nabla u\cdot\nabla\psi_i + \Delta t\alpha(u)\nabla\psi_j\cdot\nabla\psi_i - \\ & \Delta t f'(u)\psi_j\psi_i - u_1\psi_i)\,\mathrm{d}x\,. \end{aligned} \tag{58}$$

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

$$\begin{aligned} J_{i,j} = \int_\Omega & (\psi_j\psi_i + \Delta t\alpha'(u_-)\psi_j\nabla u_-\cdot\nabla\psi_i + \Delta t\alpha(u_-)\nabla\psi_j\cdot\nabla\psi_i - \\ & \Delta t f'(u_-)\psi_j\psi_i - u_1\psi_i)\,\mathrm{d}x\,. \end{aligned} \tag{59}$$

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

5.2 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

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

We do not dive into details of boundary conditions now. Dirichlet and Neumann conditions are handled as in linear diffusion problems.

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

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

This defines a nonlinear algebraic system $A(u)u = b(u)$. A Picard iteration applies old values u_- in α and f, or equivalently, old values for u in $A(u)$ and $b(u)$. The result is a linear system of the same type as those arising from $u_t = \nabla \cdot (\alpha(\boldsymbol{x})\nabla u) + f(\boldsymbol{x}, t)$.

Newton's method is as usual more involved. We first define the nonlinear algebraic equations to be solved, drop the superscript n, and introduce u_1 for u^{n-1}:

$$
\begin{aligned}
F_{i,j} = u^n_{i,j} - \frac{\Delta t}{h^2}(& \\
&\frac{1}{2}(\alpha(u^n_{i,j}) + \alpha(u^n_{i+1,j}))(u^n_{i+1,j} - u^n_{i,j}) - \frac{1}{2}(\alpha(u^n_{i-1,j}) + \alpha(u^n_{i,j}))(u^n_{i,j} - u^n_{i-1,j}) + \\
&\frac{1}{2}(\alpha(u^n_{i,j}) + \alpha(u^n_{i,j+1}))(u^n_{i,j+1} - u^n_{i,j}) - \frac{1}{2}(\alpha(u^n_{i,j-1}) + \alpha(u^n_{i,j}))(u^n_{i,j} - u^n_{i-1,j-1})) - \\
&\Delta t f(u^n_{i,j}) - u^{n-1}_{i,j} = 0\,.
\end{aligned}
$$

It is convenient to work with two indices i and j in 2D finite difference discretizations, but it complicates the derivation of the Jacobian, which then gets four indices. The left-hand expression of an equation $F_{i,j} = 0$ is to be differentiated with respect to each of the unknowns $u_{r,s}$ (short for $u^n_{r,s}$), $r \in \mathcal{I}_x$, $s \in \mathcal{I}_y$,

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

Given i and j, only a few r and s indices give nonzero contribution since $F_{i,j}$ contains $u_{i\pm 1,j}$, $u_{i,j\pm 1}$, and $u_{i,j}$. Therefore, $J_{i,j,r,s}$ is very sparse and we can set up the left-hand side of the Newton system as

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

The specific derivatives become

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

The $J_{i,j,i,j}$ entry has a few more terms. Inserting u_- for u in the J formula and then forming $J\delta u = -F$ gives the linear system to be solved in each Newton iteration.

5.3 Continuation methods

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

The continuation method is easiest to understand through an example. 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 through i channel or pipe. For $q = 0$ we have the Poisson equation (corresponding to a Newtonian fluid) and the problem is linear. A typical value for pseudo-plastic fluids may be $q_n = -0.8$. We can introduce the continuation parameter $\Lambda \in [0,1]$ such that $q = q_n\Lambda$. Let $\{\Lambda_\ell\}_{\ell=0}^n$ be the sequence of Λ values in $[0,1]$, with corresponding q values $\{q_\ell\}_{\ell=0}^n$. We can then solve a sequence of problems

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

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

6 Exercises

Problem 1: Determine if equations are nonlinear or not

Classify each term in the following equations as linear or nonlinear. Assume that a and b are unknown numbers and that u and v are unknown functions. All other symbols are known quantities.

1. $b^2 = 1$
2. $a + b = 1$, $a - 2b = 0$
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. $\nabla^2 u = \lambda e^u$

Problem 2: Linearize a nonlinear vibration ODE

Consider a nonlinear vibration problem

$$mu'' + bu'|u'| + s(u) = F(t), \tag{60}$$

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

Approximate u'' by a centered finite difference D_tD_tu, and use a centered difference D_tu for u' as well. Observe then that $s(u)$ does not contribute to making the resulting algebraic equation at a time level nonlinear. Use a geometric mean to linearize the quadratic nonlinearity arising from the term $bu'|u'|$.

Exercise 3: Find the sparsity of the Jacobian

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

Hint. Set up the unknowns that enter the difference stencil and find the sparsity of the Jacobian that arise from the stencil.

Filename: `nonlin_sparsity_Jacobian.pdf`.

Exercise 4: Newton's method for linear problems

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

Exercise 5: Differentiate a highly nonlinear term

The operator $\nabla \cdot (\alpha(u)\nabla u)$ with $\alpha(u) = ||\nabla u||^q$ appears in several physical problems, especially flow of Non-Newtonian fluids. In a Newton method one has to carry out the differentiation $\partial\alpha(u)/\partial c_j$, for $u = \sum_k c_k\psi_k$. 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`.

Problem 6: Discretize a 1D problem with a nonlinear coefficient

We consider the problem

$$((1+u^2)u')' = 1, \quad x \in (0,1), \quad u(0) = u(1) = 0\,. \tag{61}$$

a) Discretize (61) by a centered finite difference method on a uniform mesh.

b) Discretize (61) by a finite element method with P1 of equal length. Use the Trapezoidal method to compute all integrals. Set up the resulting matrix system.

Filename: `nonlin_1D_coeff_discretize.pdf`.

Problem 7: Linearize a 1D problem with a nonlinear coefficient

We have a two-point boundary value problem

$$((1+u^2)u')' = 1, \quad x \in (0,1), \quad u(0) = u(1) = 0\,. \tag{62}$$

a) Construct a Picard iteration method for (62) without discretizing in space.

b) Apply Newton's method to (62) without discretizing in space.

c) Discretize (62) by a centered finite difference scheme. Construct a Picard method for the resulting system of nonlinear algebraic equations.

d) Discretize (62) by a centered finite difference scheme. Define the system of nonlinear algebraic equations, calculate the Jacobian, and set up Newton's method for solving the system.

Filename: `nonlin_1D_coeff_linearize.pdf`.

Problem 8: Finite differences for the 1D Bratu problem

We address the so-called Bratu problem

$$u'' + \lambda e^u = 0, \quad x \in (0,1), \quad u(0) = u(1) = 0, \tag{63}$$

where λ is a given parameter and u is a function of x. This is a widely used model problem for studying numerical methods for nonlinear differential equations. The problem (63) has an exact solution

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

where θ solves

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

There are two solutions of (63) for $0 < \lambda < \lambda_c$ and no solution for $\lambda > \lambda_c$. For $\lambda = \lambda_c$ 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 (63) 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). Calculate the associated Jacobian.

Filename: `nonlin_1D_Bratu_fd.pdf`.

Problem 9: Integrate functions of finite element expansions

We shall investigate integrals on the form

$$\int_0^L f(\sum_k u_k\varphi_k(x))\varphi_i(x)\,\mathrm{d}x, \tag{64}$$

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

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

The integral (64) now takes the simplified form

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

Split this integral in two integrals over cell L (left), $[x_{i-1}, x_i]$, and cell R (right), $[x_i, x_{i+1}]$. Over cell L, u simplifies to $u_{i-1}\varphi_{i-1} + u_i\varphi_i$ (since $\varphi_{i+1} = 0$ on this cell), and over cell R, u simplifies to $u_i\varphi_i + u_{i+1}\varphi_{i+1}$. Make a `sympy` program that can compute the integral and write it out as a difference equation. Give the $f(u)$ 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}, respectively, and similar symbols for x_i, x_{i-1}, and x_{i+1}. Find formulas for the basis functions on each of the two cells, make expressions for u on the two cells, integrate over each cell, expand the answer and simplify. You can make LaTeX code and render it via http://latex.codecogs.com. Here are some appropriate Python statements for the latter purpose:

```
from 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
html_code = cgi.escape(latex_code)
# Make a file with HTML code for displaying the LaTeX formula
f = 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
text = """
<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 10: Finite elements for the 1D Bratu problem

We address the same 1D Bratu problem as described in Problem 8.

a) Discretize (10) by a finite element method using a uniform mesh with P1 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). Calculate the associated Jacobian.

Filename: `nonlin_1D_Bratu_fe.pdf`.

Problem 11: Derive the Newton system 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.

a) 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.

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

c) 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.

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

Filename: `nonlin_heat_Newton.pdf`.

Problem 12: Derive algebraic equations for nonlinear 1D heat conduction

Consider a 1D heat conduction PDE

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

where ϱ is the density of the solid material, $c(T)$ is the heat capacity, T is the temperature, and $k(T)$ is the heat conduction coefficient.

Use a uniform finite element mesh, P1 elements, and the group finite element method to derive the algebraic equations arising from the heat conduction PDE

a) Discretize the PDE by a finite difference method. Use either a Backward Euler or Crank-Nicolson scheme in time.

b) Derive the matrix and right-hand side of a Newton method applied to the discretized PDE.

Filename: `nonlin_1D_heat_PDE.pdf`.

Problem 13: 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