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 18, 2013

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

Contents

List of Problems

1 Linear versus nonlinear

In a linear differential equation, the unknown u and its derivatives do not appear in products with themselves. All equations where this property is not fulfilled, are nonlinear. For example, in

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

the unknown $u(t)$ do not appear in products with u or u' and this ODE is therefore linear, while in

$$u'(t) = a(t)u(t)(1 - u(t)/M(t)),$$

there is a product $u(t)^2$ on the right-hand side making this ODE nonlinear. Also,

$$u'' + \frac{\ell}{g}\sin(u) = 0,$$

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

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

Nonlinear differential equations pose no additional difficulty if the product terms involve already computed values (known) values of the unknown and its derivatives. In time-dependent problems, the type of time discretization can therefore eliminate the need for special treatment of nonlinear terms. However, if new unknown quantities of the solution enter product terms, the algebraic equations arising from the discretization will have nonlinear terms, and special methods for nonlinear algebraic equations must be invoked, such as Newton and Picard methods. A series of forthcoming examples will explain when nonlinearities pose difficulties and how these are resolved.

2 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. However, when the time discretization leads to nonlinear algebraic equations, we need to *linearize* these, meaning that we either approximate the nonlinear equation by a linear one, or we construct a sequence of linear equations whose solutions (hopefully) converge to the solution of the nonlinear equation. Examples will demonstrate how this is done.

2.1 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,

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

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

2.2 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 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 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, and 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

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

This equation can easily be solved analytically:

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

Here we encounter a fundamental challenge in nonlinear problems: the equation has 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.

2.3 Linearization by 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 $c = -u_1$. We can introduce $u^2 \approx u_- u$, where u_- is some approximation of u. This approximation makes the equation 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_e of the exact equation $F(u_\mathrm{e}) = 0$, but we can hope that u is closer to u_e than u_-, 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.

2.4 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

$$\frac{u^{n+1} - u^n}{\Delta t} = u^{n+\frac{1}{2}} - (u^{n+\frac{1}{2}})^2 \,. \tag{5}$$

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 (5):

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

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 Crank-Nicolson method in linear problems.

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 (5), we end up with a nonlinear term $(u^{n+1})^2$. The term can be linearized as $u^n u^{n+1}$ and a Picard iteration can then be introduced. Observe that the geometric mean avoids an iteration.

2.5 Newton's method

The Backward Euler scheme (2) for the logistic equation leads to a nonlinear algebraic equation (3) which we now write compactly as

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

Newton's method linearize 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

$$u = u_- + \frac{au_-^2 + bu_- + c}{2au_- + b}, \quad u_- \ \leftarrow u\,. \tag{6}$$

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

$$u^{n,k+1} = u^{n,k} + \frac{\Delta t(u^{n,k})^2 + (1-\Delta t)u^{n,k} - u^{n-1}}{2\Delta t u^{n,k} + 1 - \Delta t}, \quad u^{n,0} = u^{n-1}, \quad k = 0, 1, \ldots \tag{7}$$

The implementation is much closer to (6) than to (7), but the latter is better aligned with the established mathematical notation used in the literature.

2.6 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 an intermediate 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^*:

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

before proceeding with the next iteration. The parameter ω is known as the *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:

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

2.7 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.

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

```
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':

            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 even 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.

2.8 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

$$u' = f(u, t), \tag{8}$$

where f is a nonlinear function of u.

Explicit time discretization. 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 see that the scheme leads to a nonlinear algebraic equation

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

A Picard iteration scheme must in general employ the linearization,

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

while Newton's method can apply the general formula, but we need to derive

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

3 Systems of nonlinear algebraic equations

Now we assume that some time discretization of a system of ODEs, or a PDE, leads to a *system* of nonlinear algebraic equations, written compactly as

$$F(u) = 0,$$

or with some special structure that often appear in real applications, e.g.,

$$A(u)u = b(u) .$$

Here, u is a vector of unknowns $u = (u_0, \ldots, u_N)$, and F is a vector function: $F = (F_0, \ldots, F_N)$. Similarly, $A(u)$ is an $(N+1) \times (N+1)$ matrix function of u and b is 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 convergence properties of these methods, can be found in Kelley [**?**].

3.1 Picard iteration

We cannot apply Picard iteration to nonlinear equations unless there is some special structure. For $A(u)u = b(u)$ 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. $u_- \ \leftarrow \ u_-$

3.2 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 Taylor expand F around a known value u_- and just keep the linear terms. The multi-variate Taylor expansion has its two first terms as

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

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

$$J_{i,j} = \sum_k \frac{\partial A_{i,k}}{\partial u_j} u_k + A_{i,j} - \frac{\partial b_i}{\partial u_j}\,. \tag{9}$$

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.

3.3 Stopping criteria

Let $||\cdot||$ be the standard Eucledian vector norm. Four termination criteria are 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: $||F(u)|| \leq \epsilon_r||F(u_0)||$

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

$$||F(u)|| \leq \epsilon_{rr}||F(u_0)|| + \epsilon_{ra}, \tag{10}$$

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:

$$||\delta u|| \leq \epsilon_{ur}||u_0|| + \epsilon_{ua}, \tag{11}$$

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

$$||F(u)|| \leq \epsilon_{rr}||F(u_0)|| + \epsilon_{ra} \text{ or } ||\delta u|| \leq \epsilon_{ur}||u_0|| + \epsilon_{ua} \text{ or } k > k_{\max}\,. \tag{12}$$

3.4 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

$$S' = -\beta SI, \tag{13}$$
$$I' = \beta SI - \nu I, \tag{14}$$

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}:

$$\frac{S^{n+1} - S^n}{\Delta t} = -\beta[SI]^{n+\frac{1}{2}} \approx -\frac{\beta}{2}(S^n I^n + S^{n+1}I^{n+1}), \tag{15}$$
$$\frac{I^{n+1} - I^n}{\Delta t} = \beta[SI]^{n+\frac{1}{2}} - \nu I^{n+\frac{1}{2}} \approx \frac{\beta}{2}(S^n I^n + S^{n+1}I^{n+1}) - \frac{\nu}{2}(I^n + I^{n+1})\,. \tag{16}$$

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

$$F_S(S,I) = S - S_1 + \frac{1}{2}\Delta t\beta(S_1I_1 + SI) = 0, \tag{17}$$

$$F_I(S,I) = I - I_1 - \frac{1}{2}\Delta t\beta(S_1I_1 + SI) - \frac{1}{2}\Delta t\nu(I_1 + I) = 0\,. \tag{18}$$

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

$$S = \frac{S_1 - \frac{1}{2}\Delta t\beta S_1I_1}{1 + \frac{1}{2}\Delta t\beta I_-},$$

$$I = \frac{I_1 + \frac{1}{2}\Delta t\beta S_1I_1}{1 - \frac{1}{2}\Delta t\beta S_- + \nu}\,.$$

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

Newton's method. The nonlinear system (17)-(18) can be written as $F(u) = 0$ with $F = (F_S, F_I)$ and $u = (S, I)$. The Jacobian becomes

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

The Newton system to be solved in each iteration is then

$$\begin{pmatrix} 1 + \frac{1}{2}\Delta t\beta I_- & \frac{1}{2}\Delta t\beta S_- \\ -\frac{1}{2}\Delta t\beta S_- & 1 - \frac{1}{2}\Delta t\beta I_- - \frac{1}{2}\Delta t\nu \end{pmatrix} \begin{pmatrix} \delta S \\ \delta I \end{pmatrix} = \begin{pmatrix} S_- - S_1 + \frac{1}{2}\Delta t\beta(S_1I_1 + S_-I_-) \\ I_- - I_1 - \frac{1}{2}\Delta t\beta(S_1I_1 + S_-I_-) - \frac{1}{2}\Delta t\nu(I_1 + I_-) \end{pmatrix}$$

4 Linearization at the PDE 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

$$\frac{\partial u}{\partial t} = \nabla\cdot(\alpha(u)\nabla u) + f(u), \qquad \boldsymbol{x}\in\Omega,\ t\in(0,T], \tag{19}$$

$$-\alpha(u)\frac{\partial u}{\partial n} = g, \qquad \boldsymbol{x}\in\partial\Omega_N,\ t\in(0,T], \tag{20}$$

$$u = u_0, \qquad \boldsymbol{x}\in\partial\Omega_D,\ t\in(0,T]\,. \tag{21}$$

4.1 Explicit time integration

The nonlinearities in the PDE are trivial to deal with if we choose an explicit time integration method for (19), 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}:

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

4.2 Picard iteration

A Backward Euler scheme for (19) reads

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

Written out,

$$\frac{u^n - u^{n-1}}{\Delta t} = \nabla \cdot (\alpha(u^n)\nabla u^n) + f(u^n) \tag{22}$$

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

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

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., $u^{n,k+1}$, let u_- be the most recently computed value, $u^{n,k}$, and let u_1 denote the unknown function at the previous time level, u^{n-1}. The PDE to be solved in a Picard iteration then looks like

$$\frac{u - u_1}{\Delta t} = \nabla \cdot (\alpha(u_-)\nabla u) + f(u_-)\,. \tag{24}$$

At the beginning of the iteration we set $u_- = u_1$.

4.3 Newton's method

At time level n we have to solve the stationary PDE (22), 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

$$u^n = u^{n,k} + \delta u\,. \tag{25}$$

The idea is to insert (25) in (22), 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 (25) to find a new approximation $u^{n,k+1} = u^{n,k} + \delta u$ to u^n.

Inserting (25) in (22) gives

$$\frac{u^{n,k} + \delta u - u^{n-1}}{\Delta t} = \nabla\cdot(\alpha(u^{n,k} + \delta u)\nabla(u^{n,k} + \delta u)) + f(u^{n,k} + \delta u) \tag{26}$$

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

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

Inserting the linear approximations of α and f in (26) results in

$$\begin{aligned}\frac{u^{n,k} + \delta u - u^{n-1}}{\Delta t} &= \nabla\cdot(\alpha(u^{n,k})\nabla u^{n,k}) + f(u^{m,k}) + \\ &\quad \nabla\cdot(\alpha(u^{n,k})\nabla\delta u) + \nabla\cdot(\alpha'(u^{n,k})\delta u\nabla u^{n,k}) + \\ &\quad \nabla\cdot(\alpha'(u^{n,k})\delta u\nabla\delta u) + f'(u^{n,k})\delta u\end{aligned} \tag{27}$$

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

$$F(u^{n,k}) = \frac{u^{n,k} - u^{n-1}}{\Delta t} - \nabla\cdot(\alpha(u^{n,k})\nabla u^{n,k}) + f(u^{n,k}), \tag{28}$$

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

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 $u^{n,k+1}$.

$$\frac{u^{n,k+1} - u^{n-1}}{\Delta t} = \nabla \cdot (\alpha(u^{n,k})\nabla u^{n,k+1}) + f(u^{n,k}) + \nabla \cdot (\alpha'(u^{n,k})\delta u \nabla u^{n,k}) + f'(u^{n,k})\delta u\,. \tag{30}$$

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

$$F(u_-) = \frac{u_- - u_1}{\Delta t} - \nabla \cdot (\alpha(u_-)\nabla u_-) + f(u_-), \tag{31}$$

$$\delta F(\delta u; u_-) = -\frac{1}{\Delta t}\delta u + \nabla \cdot (\alpha(u_-)\nabla \delta u) + \nabla \cdot (\alpha'(u_-)\delta u \nabla u_-) + f'(u_-)\delta u\,. \tag{32}$$

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

$$\frac{u - u_1}{\Delta t} = \nabla \cdot (\alpha(u_-)\nabla u) + f(u_-) + \nabla \cdot (\alpha'(u_-)\delta u \nabla u_-) + f'(u_-)\delta u\,. \tag{33}$$

5 Discretization of 1D problems

Section 4 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 treated in Section 3. This latter approach will now be described in detail.

We shall work with the 1D problem

$$-(\alpha(u)u')' + au = f(u), \quad x \in (0, L), \quad \alpha(u(0))u'(0) = C,\ u(L) = 0\,. \tag{34}$$

This problem is of the same nature as those arising from implicit time integration of a nonlinear diffusion PDE as outlined in Section 4.2 (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).

5.1 Finite difference discretizations

The nonlinearity in the differential equation (34) poses no more difficulty than a variable coefficient in $(\alpha(x)u')'$. We can therefore use a standard approach to discretizing the Laplace term with a variable coefficient:

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

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

$$-\frac{1}{\Delta x^2}\left(\alpha_{i+\frac{1}{2}}(u_{i+1} - u_i) - \alpha_{i-\frac{1}{2}}(u_i - u_{i-1})\right) + au_i = f(u_i)\,. \tag{35}$$

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:

$$\alpha_{i+\frac{1}{2}} \approx \alpha(\frac{1}{2}(u_i + u_{i+1}) = [\alpha(\overline{u}^x)]^{i+\frac{1}{2}}, \tag{36}$$

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

Equation (35) with the latter approximation then looks like

$$-\frac{1}{2\Delta x^2}\left((\alpha(u_i) + \alpha(u_{i+1}))(u_{i+1} - u_i) - (\alpha(u_{i-1}) + \alpha(u_i))(u_i - u_{i-1})\right) + au_i = f(u_i)\,. \tag{38}$$

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

$$\alpha(u_0)\frac{u_1 - u_{-1}}{2\Delta x} = C\,. \tag{39}$$

The fictitious value u_{-1} can be eliminated with the aid of (38) for $i = 0$. Formally, (38) should be solved with respect to u_{i-1} and that value ($for i = 0$) should be inserted in (39), 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 (39) after every Picard or Newton iteration. Such an approach means that we use a known u_{-1} value in (38) from the previous iteration.

The nonlinear algebraic equations (38) 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 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 3.

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

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

The explicit expression for nonlinear equation number i, $F_i(u_0, u_1, \ldots)$, arises from moving all terms in (38) 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.

5.2 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

$$\int_0^L \alpha(u)u'v'\,\mathrm{d}x = \int_0^L f(u)v\,\mathrm{d}x - Cv(0), \quad \forall v \in V\,. \tag{40}$$

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

$$\int_0^L \alpha(\sum_{k\in\mathcal{I}_s} c_k\psi_k)\psi_j'\psi_i'\,\mathrm{d}x = \int_0^L f\psi_i\,\mathrm{d}x - Cv(0), \quad i \in \mathcal{I}_s\,. \tag{41}$$

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 $\int_0^L \alpha(\sum_{k\in\mathcal{I}_s} c_k)\psi_k)\psi_i'\psi_j'\,\mathrm{d}x$ and $\int_0^L f(\sum_{k\in\mathcal{I}_s} c_k)\psi_k)\psi_i\,\mathrm{d}x$ when we do not the c_k coefficients in the argument of the α function? We must resort to numerical integration or the group finite element method.

5.3 The group finite element method

Motivation. Let us simplify the model problem for a while and set α, choose $f(u) = u^2$, and have Dirichlet conditions at both ends such that we have a very simple nonlinear problem $-u'' = u^2$. The variational form is then

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

The term with $u'v'$ is well known so the only new feature is the term $\int u^2v\,\mathrm{d}x$. With $v = \psi_i$ and $u = \sum_j c_j\psi_j$, this term becomes

$$\int_0^L (\sum_k c_k\psi_k)^2\psi_i\,\mathrm{d}x\,.$$

Using finite elements, $\psi_i = \varphi_i$, of P1 type, one can show that $\int u^2v\,\mathrm{d}x$ gives rise to the following terms in the algebraic equations:

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

written with $u_i = u(x_i) = c_i$ to better illustrate the difference equation in terms of u values. Obviously, even u^2 gives rise to a complicated term, especially when compared to the finite difference counterpart u_i^2.

Finite element approximation of functions of u. Since we already expand u as $\sum_j c_j\varphi_j$ we may use the same approximation for nonlinearities. More precisely, we have in general

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

where I_b is the index set consisting of the numbers of the nodes subject to a Dirichlet condition, $u(x_j) = U_j$, and $\mathcal{I}_s = \{0, 1, \ldots,\}$ are the indices of the unknowns in the resulting linear system.

We can now approximate f in a similar way,

$$f(u) \approx \sum_{j\in I_b} f(u_j)\varphi_j + \sum_{j\in\mathcal{I}_s} f(u_{\nu(j)})\psi_{\nu(j)}(x),$$

using u_j as a notation for the value of the finite element function u at node x_j. The expression for f is actually a sum of $f(u_j)\varphi_j$ over all nodes so we can avoid distinguishing between Dirichlet nodes and the rest and just write

$$f(u) \approx \sum_{j=0}^{N_n} f(u_j)\varphi_j(x)\,. \tag{42}$$

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

The main advantage of the group finite element method is for deriving difference equations in nonlinear problems. Computer programs will always integrate $\int f(u)\varphi_i\,\mathrm{d}x$ numerically and use an existing approximation of u in $f(u)$ such that the integrand can be sampled at any spatial point.

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

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

We recognize this expression as the mass matrix M ($\int \varphi_i\varphi_j\,\mathrm{d}x$) times the vector $f = (f(u_0), f(u_1), \ldots,)$: Mf. The associated terms in the difference equations are

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

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.

5.4 Numerical integration of nonlinear terms

We can apply numerical integration directly to a term like

$$\int_0^L f(\sum_k c_k\psi_k(x))\psi_i(x)\,\mathrm{d}x,$$

as long as we have some coefficients c_k from some previous iterations such that $\sum_k c_k\psi_k(x)$ can be evaluated for some x.

Choosing finite element basis functions, we have $\psi_i = \varphi_{\nu(i)}$, but in the present model problem $\nu(i) = i$ with a left-to-right numbering. The integral above is the written as

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

where we have used that $c_k = u(x_k) = u_k$ for a node point x_k. Replacing c_k by u_k makes it easier to interpret the resulting algebraic equations as finite difference approximations.

Let us now apply an integration rule that samples the integrand at the node points only. The motivation is that the basis functions φ_i vanish at all such points except at x_i. For example,

$$\sum_k u_k\varphi_k(x_j) = u_j,$$

and $\varphi_i(x_j) = \delta_{ij} = 0$ if $i \neq j$ and unity otherwise. Applying the Trapezoidal rule gives

$$\begin{aligned}\int_0^L f(\sum_k u_k\varphi_k)\varphi_i\,\mathrm{d}x &\approx \frac{h}{2}f(u_0)\varphi_i(0) + \frac{h}{2}f(u_N)\varphi_i(L) + h\sum_{j=1}^{N-1} f(u_j)\varphi_i(x_j)\\ &= hf(u_i),\end{aligned}$$

with a factor one half if $i = 0$ or $i = N_n$.

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.

5.5 Finite element discretization of a variable coefficient Laplace term

Turning back to the model problem (34), 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.

The group finite element method can be used to precompute the integral and also give some analytical insight:

$$\int_0^L \alpha(\sum_k c_k\psi_k)\psi_i'\psi_j'\,\mathrm{d}x \approx \sum_k \left(\underbrace{\int_0^L \psi_k\psi_i'\psi_j'\,\mathrm{d}x}_{L_{i,j,k}}\right)\alpha(u_k) = \sum_k L_{i,j,k}\alpha(u_k)\,.$$

With finite element basis functions and P1 elements the $L_{i,j,k}$ quantity can be computed at the cell level by hand as

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

with $r, s, t = 0, 1$ are indices over local degrees of freedom ($i = q(e, r)$, $j = q(e, s)$, 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

$$\frac{1}{2}(\alpha(\tilde{u}_0) + \alpha(\tilde{u}_1))\frac{1}{h}\begin{pmatrix} 1 & -1 \\ -1 & 1\end{pmatrix}. \tag{43}$$

The assembly of such element matrices with constant h results in

$$\frac{1}{h}\left(\frac{1}{2}(\alpha(u_i) + \alpha(u_{i+1}))(u_{i+1} - u_i) - \frac{1}{2}(\alpha(u_{i-1}) + \alpha(u_i))(u_i - u_{i-1})\right), \tag{44}$$

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 $\int_0^L \alpha(\sum_k u_k\varphi_k)\varphi_i'\varphi_j'\,\mathrm{d}x$ at the cell level:

$$\begin{aligned}\int_{-1}^1 \alpha(\sum_{t=0}^1 \tilde{u}_t\tilde{\varphi}_t)\frac{2}{h}\frac{d\tilde{\varphi}_r}{dX}\frac{2}{h}\frac{d\tilde{\varphi}_s}{dX}\frac{h}{2}dX &= \frac{1}{2h}(-1)^r(-1)^s\int_{-1}^1 \alpha(\sum_{t=0}^1 u_t\tilde{\varphi}_t(X))dX \\ &\approx \frac{1}{2h}(-1)^r(-1)^s(\sum_{t=0}^1 \tilde{\varphi}_t(-1)\tilde{u}_t + \sum_{t=0}^1 \tilde{\varphi}_t(1)\tilde{u}_t) \\ &= \frac{1}{2h}(-1)^r(-1)^s(\alpha(\tilde{u}_0) + \alpha(tildeu_1))\,. \end{aligned} \tag{45}$$

The element matrix in (45) is identical to the one in (43), showing that the group finite element method and Trapezoidal integration are equivalent with the standard finite discretization of a nonlinear Laplace term.

The final term in the variational form is the Neumann condition at the boundary: $Cv(0) = C\varphi_i(0) = C\delta_{i0}$ since only $i = 0$ will give $\varphi_i(0) \neq 0$.

Summary.
For the equation

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

P1 finite elements results in difference equations where

- the term $-(\alpha(u)u')'$ becomes $-h[D_x\overline{\alpha(u)}^x D_x u]_i$ 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.

As we have the nonlinear difference equations available in the finite element, 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.

5.6 Picard iteration defined from the variational form

We address again the problem (34) with variational form (40). Our aim is to define a Picard iteration based on this variational form. The idea 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 variational form to be used for Picard iteration is then

$$\int_0^L \alpha(u_-)u'v'\,\mathrm{d}x = \int_0^L f(u_-)v\,\mathrm{d}x - Cv(0), \quad \forall v \in V\,. \tag{46}$$

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

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

The linear system is computed the standard way from this variational problem.

5.7 Newton's method derived from the variational form

Application of Newton's method to the nonlinear variational form (40) arising from the problem (34) 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 follows from the variational form

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

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, depending on how Dirichlet conditions are to be incorporated. Without a boundary function we get

$$F_i = \int_0^L \left(\alpha(\sum_{k\in\mathcal{I}_s} c_k\psi_k)(\sum_{j\in\mathcal{I}_s} c_j\psi_j')\psi_i' - f(\sum_{k\in\mathcal{I}_s} c_k\psi_k)\psi_i\right)\,\mathrm{d}x - C\psi_i(0), \quad i \in \mathcal{I}_s\,. \tag{47}$$

The associated Jacobian becomes

$$\begin{aligned} J_{i,j} = \frac{\partial F_i}{\partial c_j} &= \int_0^L \frac{\partial}{\partial c_j}(\alpha(\sum_{k\in\mathcal{I}_s} c_k\psi_k)(\sum_{j\in\mathcal{I}_s} c_j\psi_j')\psi_i' - f(\sum_{k\in\mathcal{I}_s} c_k\psi_k)\psi_i)\,\mathrm{d}x \\ &= \int_0^L (\alpha'(\sum_{k\in\mathcal{I}_s} c_k\psi_k)\psi_j(\sum_{j\in\mathcal{I}_s} c_j\psi_j')\psi_i' + \alpha(\sum_{k\in\mathcal{I}_s} c_k\psi_k)\psi_j'\psi_i')\,\mathrm{d}x - \\ &\quad \int_0^L f'(\sum_{k\in\mathcal{I}_s} c_k\psi_k)\psi_j\psi_i\,\mathrm{d}x\,. \end{aligned} \tag{48}$$

In the above derivation we have used the important result

$$\frac{\partial}{\partial c_j}\sum_k c_k\psi_k = \psi_j\,. \tag{49}$$

When forming F_i and $J_{i,j}$ in a program from the variational forms we use a previously computed u, denoted by u_- for the sums $\sum_k c_k\psi_k$ in the above expressions. With this notation we have

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

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

Most computer programs require the user to define F_i and $J_{i,j}$, but many programs have a gallery of models with predefined PDE problems and associated F_i and $J_{i,j}$ available. Some programs, like FEniCS[2], are capable of automatically deriving $J_{i,j}$ if F_i is specified.

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

6 Multi-dimensional PDE problems

6.1 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

$$\int_\Omega (uv + \Delta t \alpha(u)\nabla u \cdot \nabla v - \Delta t f(u)v - u_1 v)\,\mathrm{d}x \,. \tag{52}$$

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

$$F_i = \int_\Omega (u\psi_i + \Delta t \alpha(u)\nabla u \cdot \nabla \psi_i - \Delta t f(u)\psi_i - u_1\psi_i)\,\mathrm{d}x \,. \tag{53}$$

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

$$F_i \approx \hat{F}_i = \int_\Omega (u_-\psi_i + \Delta t \alpha(u_-)\nabla u \cdot \nabla \psi_i - \Delta t f(u_-)\psi_i - u_1\psi_i)\,\mathrm{d}x \,. \tag{54}$$

The equations $\hat{F}_i = 0$ are now linear and we can easily derive a linear system for $\{c_i\}_{i\in\mathcal{I}_s}$ 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:

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

The Jacobian is obtained by differentiating (53) and using $\partial u/\partial c_j = \psi_j$:

$$J_{i,j} = \frac{\partial F_i}{\partial c_j} = \int_\Omega (\psi_j\psi_i + \Delta t \alpha'(u)\psi_j \nabla u\cdot\nabla\psi_i + \Delta t\alpha(u)\nabla\psi_j\cdot\nabla\psi_i - \Delta t f'(u)\psi_j\psi_i - u_1\psi_i)\,\mathrm{d}x \,. \tag{56}$$

The evaluation of $J_{i,j}$ applies the known approximation u_- for u:

$$J_{i,j} = \int_\Omega (\psi_j\psi_i + \Delta t\alpha'(u_-)\psi_j\nabla u_-\cdot\nabla\psi_i + \Delta t\alpha(u_-)\nabla\psi_j\cdot\nabla\psi_i - \Delta t f'(u_-)\psi_j\psi_i - u_1\psi_i)\,\mathrm{d}x\,. \tag{57}$$

Hopefully, these example also show how convenient the notation with u and u_- is: the unknown 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 [**?**].

6.2 Finite difference discretization

6.3 Continuation methods

7 Exercises

Problem 1: Linearize a nonlinear vibration ODE

Consider a nonlinear vibration problem

$$mu'' + bu'|u'| + s(u) = F(t), \tag{58}$$

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 $bu'|u'|$.

Problem 2: Discretize a 1D problem with a nonlinear coefficient

We consider the problem

$$((1+u^2)u')' = 1, \quad x \in (0,1), \quad u(0) = u(1) = 0\,. \tag{59}$$

a) Discretize (59) by a centered finite difference method on a uniform mesh.

b) Discretize (59) 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 3: Linearize a 1D problem with a nonlinear coefficient

We have a two-point boundary value problem

$$((1+u^2)u')' = 1, \quad x \in (0,1), \quad u(0) = u(1) = 0\,. \tag{60}$$

a) Construct a Picard iteration method for (60) without discretizing in space.

b) Apply Newton's method to (60) without discretizing in space.

c) Discretize (60) by a centered finite difference scheme. Construct a Picard method for the resulting system of nonlinear algebraic equations.

d) Discretize (60) 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 4: Finite differences for the 1D Bratu problem

We address the so-called Bratu problem

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

where λ is a given parameter and u is a function of x. This is a widely used model problem for nonlinear differential equations. The problem (61) 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 (61) for $0 < \lambda < \lambda_c$, while for $\lambda > \lambda_c$ there are no solutions. There is one unique solution for $\lambda = \lambda_c$. 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 (61) 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 5: Finite elements for the 1D Bratu problem

We address the same 1D Bratu problem as described in Problem 4.

a) Discretize (5) 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`.

References

Index