MATH521
Numerical Analysis of Partial Differential Equations

Winter 2017/18, Term 2

Homework Assignment 9

Question 1 | 1 mark We consider the initial boundary value problem for the heat equation

$$\begin{aligned} \partial_t u(t) - a\Delta u(t) &= f(t) && \text{in } Q =]0,T[\times\Omega \\ u(0) &= u_0 && \text{in } \Omega \\ \frac{\partial u}{\partial n} &= 0 && \text{on } \Sigma =]0,T[\times\partial\Omega \end{aligned} \tag{H}$$

where u is a temperature field, u_0 an initial temperature distribution, the diffusion-like parameter $a > 0$ the heat conductivity of the material, f a source term e.g. due to thermal radiation and $T > 0$ a final time. The homogeneous NEUMANN boundary conditions mean that the domain Ω is perfectly insulated so that no thermal energy is radiated into the environment.

The θ-method is a class of RUNGE-KUTTA schemes for integrating ODEs of the form

$$\dot{U} = F(t,U)$$

by using the iteration

$$U_+ = U_\circ + \Delta t\,(\theta F(t_+,U_+) + (1-\theta)F(t_\circ,U_\circ))\,.$$

The parameter $\theta \in [0,1]$ can be interpreted as the 'degree of implicitness', since $\theta = 0$ gives the forward EULER method, $\theta = \frac{1}{2}$ the CRANK-NICOLSON method (aka implicit trapezium rule in the ODE context) and $\theta = 1$ the backward EULER method.

For the discretisation in space, we apply linear finite elements. Both the time step size Δt and the spatial triangulation $\mathcal{T}^h$ are fixed.

Show that in this setting both the method of lines and ROTHE's method lead to the same discrete problems

$$\left(M^h + \theta\Delta t a K^h\right)\vec{u}^h_+ = \left(M^h - (1-\theta)\Delta t a K^h\right)\vec{u}^h_\circ + \Delta t\left(\theta \vec{f}^h_+ + (1-\theta)\vec{f}^h_\circ\right).$$

You don't have to include any details about the components of the discrete vectors and matrices. We all know what they are!

Solution. **Method of lines.** First, we semi-discretise the problem (H) in space using linear finite elements to obtain the weak form of the method of lines

$$\begin{aligned} M^h\partial_t\vec{u}^h + aK^h\vec{u}^h &= \vec{f}^h && \text{in } Q^h =]0,T[\times\Omega^h \\ \vec{u}^h(0) &= \vec{u}^h_0 && \text{in } \Omega^h. \end{aligned} \tag{1}$$

If we now write $U = \vec{u}^h$ and $F(t,\vec{u}^h) = (M^h)^{-1}(\vec{f}^h - aK^h\vec{u}^h)$, we can write the θ-method scheme for the initial value problem (1) as

$$\vec{u}^h_+ = \vec{u}^h_\circ + \Delta t\left(\theta\,(M^h)^{-1}(\vec{f}^h_+ - aK^h\vec{u}^h_+) + (1-\theta)(M^h)^{-1}(\vec{f}^h_\circ - aK^h\vec{u}^h_\circ)\right),$$

or, multiplying through by M^h and rearranging, we have

$$\left(M^h + \theta\Delta t a K^h\right)\vec{u}^h_+ = \left(M^h - (1-\theta)\Delta t a K^h\right)\vec{u}^h_\circ + \Delta t\left(\theta \vec{f}^h_+ + (1-\theta)\vec{f}^h_\circ\right)$$

as desired.

Rothe's method. First, we semi-discretise equation (H) in time using the θ-method by writing $U = u$ and $F(t,u) = f + a\Delta u$. Since the time step Δt is fixed, the resulting scheme is given by

$$u_+ = u_\circ + \Delta t\Big[\theta(f_+ + a\Delta u_+) + (1-\theta)(f_\circ + a\Delta u_\circ)\Big].$$

Rearranging, we have

$$(1 - \theta\Delta t a)\, u_+ = (1 - (1-\theta)\Delta t a)\, u_\circ + \Delta t\, (\theta f_+ + (1-\theta) f_\circ)\,.$$

Now, using the fact that the spatial triangulation $\mathcal{T}^h$ is invariant with time, we can safely multiply both sides through using test functions from the same linear finite element space and integrate in the standard way to obtain

$$\left(M^h + \theta\Delta t a K^h\right) \vec{u}^h_+ = \left(M^h - (1-\theta)\Delta t a K^h\right) \vec{u}^h_\circ + \Delta t \left(\theta \vec{f}^h_+ + (1-\theta)\vec{f}^h_\circ\right)$$

as desired.

$\square$

Question 2 | 4 marks

(a) The FEniCS scridpt `hw9.py` implements the backward Euler method for Problem (H). Starting from room temperature ($u_0 \equiv 20$), the bottom left corner of a metal piece Ω with conductivity parameter $a = 0.1$ is held over a flame for one second, then the flame is extinguished. This is modelled by

$$f(t,x) = \begin{cases} 200e^{-5x_1^2-2x_2^2} & t \leq 1 \\ 0 & t > 1 \end{cases}$$

Complete the missing commands to compute the evolution of the temperature field over the first five seconds using a time step size of $\Delta t = 10^{-2}$.

Save your results as a video, using a frame rate such that the video time is equal to the physical time. You don't have to submit any other files for this part of Question 2.

Hint: Open the PVD-file in ParaView, click the 'Apply'-button, select a reasonable colour map and then re-scale the colour values to the range $[20, 160]$. Use the same range for the following questions, too.

Solution. See attached video. Additionally, the code `hw9.py` in the Appendix A is used for all subsequent parts of this question.

□

(b) Generalise this script to implement the θ-method from Question 1. Check whether setting $\theta = 1$ still gives you the same results. Using the same parameters as in Question 2(a), solve the problem with the CRANK-NICOLSON method and the forward EULER method. What do you observe?

Solution. First, I verified that the difference between the solution computed using the backward EULER method and the solution computed using the θ-method with $\theta = 1$ were numerically identical by taking the norm of their difference, which yielded zero to within machine precision.

Solving the problem with the CRANK-NICOLSON method (i.e. $\theta = 1/2$), the solution appears to be relatively close to the backward EULER solution, although it is certainly somewhat off. At time $t = 5$, the L^2-norm difference between the backward EULER and CRANK-NICOLSON was approximately 10.786.

Solving the problem with the forward EULER method (i.e. $\theta = 0$), however, yielded a completely different solution: the maximum temperature at $t = 5$ nearly doubles compared to the backward EULER solution, and the L^2-norm difference between the two solutions rose to approximately 28.075. My interpretation would be that, since forward EULER time stepping is unstable, that the numerical solution has begun to diverge.

$\square$

(c) Solve the problem with the forward EULER method again up to time $T = 0.1$, once with $\Delta t = 1.25 \times 10^{-4}$ and once with $\Delta t = 10^{-4}$. Explain your observations, using the relevant terminology.

Solution. Firstly, from Example 3.2.5 in the notes (Stability and Dissipation of the forward EULER Method), we know that for the heat equation that for time steps $\Delta t = \mathcal{O}(h^2)$, it is possible for the forward EULER method may in fact be stable.

Now, in our numerical simulations we have that $h \approx 1/50$, and so $h^2 \approx 1/2500 = 4 \times 10^{-4}$. Therefore, we indeed have that $\Delta t = \mathcal{O}(h^2)$ for $\Delta t = 1.25 \times 10^{-4}$ and $\Delta t = 10^{-4}$, and we should expect that the forward EULER solution should be stable, or at least not exceedingly unstable (since we don't know the proportionality constant, after all).

Using $\Delta t = 1.25 \times 10^{-4}$ yielded fairly reasonable looking results. I compared with the backward EULER method for a sanity check, and they agree with within a fraction of a degree across the domain.

After solving with $\Delta t = 10^{-4}$, however, the resulting temperature distributions were much smaller than the corresponding solution for $\Delta t = 1.25 \times 10^{-4}$. The solution heat distribution both didn't spread out as far across the domain and achieved lower values despite being exposed to the heat source for the same amount of time. With that being said, the backward EULER solution exhibited the same behaviour, and so it may be an artifact of the smaller time steps and not faults in the forward EULER method in particular.

In conclusion, although it appears that the forward EULER method is stable for these smaller time steps Δt and is fairly consistent with the backward EULER method, neither solutions seem trustworthy; completely different temperature distributions arise when only changing the time step Δt and not the final time T, and this should not happen for a reasonable method.

□

(d) If you could choose among the time-stepping schemes considered above and other time-discretisation schemes you know of, which one would you choose for this problem? First formulate some reasonable objectives for your simulation, then explain which method attains these to the greatest extent.

Solution. First, we consider reasonable objectives for the method used for the simulation of the heat equation:

- **Dissipation**. The heat equation is naturally dissipative (the negative laplacian has eigenvalues which are bounded from below above zero), and so the solution should be dissipative for all times, increasing in value only do to sources in the domain.
- **Stability**. The continuous heat equation decays all perturbations exponentially in time (Example 3.2.7, Theorem 3.1.7 in the notes), and so the numerical method of choice should match this requirement. For this reason, we require *strongly* A-stable method. This will naturally satisfy the dissipation requirements for the heat equation as well since strongly A-stable methods are dissipative for the heat equation.
- **Consistency**. Since we have used a second order spatial discretization scheme, ideally we would choose a time-discretisation method that achieves the same order of consistency, namely $\mathcal{O}(\Delta t^2)$.
- **Discrete Parabolic Maximum Principle**. The heat equation satisfies the maximum principle, and an ideal time stepping scheme would adhere to this as well.

Taking these requirements into consideration, we make the following conclusions:

- Forward and backward EULER methods are ruled out based on consistency requirements, as we would prefer a second order accurate in time solution.
- The CRANK-NICOLSON method, while second order consistent, is only regularly A-stable and therefore is ruled out.
- Modifications to the CRANK-NICOLSON method, however, are promising. The θ-method can only be made strictly A-stable, but both the $TR\text{-}BDF2$ method and the fractional step θ method are both strongly A-stable and second order accurate in both time and space.

Now, I have not shown that the discrete parabolic maximum principle holds for either of these latter two methods, but since it already does not hold generally for the backward EULER method and only hold for the forward EULER method when using mass lumping, these methods are already the best methods that I know of based on the listed requirements for the heat equation.

As a tie-breaker, I will choose the method with lower computational cost, and use the $TR\text{-}BDF2$ method.

□

Your Learning Progress What is the one most important thing that you have learnt from this assignment?

- I learned about the relative stability of different time stepping schemes can drastically effect the result of even a relatively simple simulation.

Any new discoveries or achievements towards the objectives of your course project?

- I wouldn't say that this in particular helps for my project, although I might have to try an implementation of something using FENICS, as it seems like quite a convenient tool.

What is the most substantial new insight that you have gained from this course this week? Any *aha moment*?

- Again, probably no *aha moment*, but it was a good exercise and it was fun to play around with FENICS.

Appendix A

hw9.py

```
# coding=utf-8
"""
FEniCS program: Solution of the heat equation with homogeneous Neumann boundary
conditions.
"""

from __future__ import print_function
from fenics import *
from mshr import *
import time
import numpy as np

def create_heateqn_problem(t0 = 0.0, T = 5.0, dt = 1e-2):
    # Create a geometry and mesh it
    square = Rectangle(Point(0., 0.), Point(1., 1.))
    diskM = Circle(Point(0.5, 0.5), 0.2)
    diskSW = Circle(Point(0.25, 0.), 0.2)
    diskSE = Circle(Point(0.75, 0.), 0.2)
    diskE = Circle(Point(1, 0.5), 0.2)
    diskNE = Circle(Point(0.75, 1), 0.2)
    diskNW = Circle(Point(0.25, 1), 0.2)
    diskW = Circle(Point(0., 0.5), 0.2)
    domain = square - diskM - diskSW - diskSE - diskE - diskNE - diskNW - diskW
    mesh = generate_mesh(domain, 50) # h ~ 1/50

    # Function space of linear finite elements
    V = FunctionSpace(mesh, 'P', 1)

    # Problem data
    u0 = interpolate(Constant(20.0), V) # initial temperature
    T_source = 1.0 # source f is active for the first 'T_source' seconds

    # Parameters of the time-stepping scheme
    t = t0 # current time
    tsteps = int(round(T/dt)) # number of time steps (round to ensure integer)

    # Refer to Theorem 2.3.25 to decide upon a sensible degree for the interpolation
    # of the source term f below
    #   From Theorem 2.3.25 we have that the error in the B norm, ||e||_B, is O(h)
    #   for conforming linear finite elements if the order of the quadrature formula
    #   is at least r = 2k-1, where k=1 is the order of the finite elements.
    #   So, since order 1 quadrature integrates linear functions exactly and  C
    #   functions O( h  ) in 2D, if we take an order 1 approximation of f, the error
    #   in approximating f will be O( h  ) which is the same order as the error from
    #   integrating the quadratic f*v, which is less than the O(h) error
    f = Expression("t > tstop ? 0 : 200*exp(-5*x[0]*x[0]-2*x[1]*x[1])",
    degree = 1, t = t, tstop = t0 + T_source)

    return u0, f, V, t, tsteps


def heateqn_bwdeuler(t0 = 0.0, T = 5.0, dt = 1e-2, a = 0.1, prnt = True,
                     save = False, fname = 'heat/bwdeuler_theta.pvd'):
    # Create problem geometry
    u0, f, V, t, tsteps = create_heateqn_problem(t0 = 0.0, T = 5.0, dt = 1e-2)

    # Define the variational problem for the Backward Euler scheme
    u = TrialFunction(V)
    v = TestFunction(V)
    B = u*v*dx + dt*a*dot(grad(u), grad(v))*dx

    # Export the initial data
    u = Function(V, name='Temperature')
    u.assign(u0)
    if save:
        results = File('heat/backwardEuler.pvd')
        results << (u, t)

    # Time stepping
    for k in range(tsteps):

        # Current time
        t = t0 + (k+1)*dt
        if prnt:
            print('Step = ', k+1, '/', tsteps , 'Time =', t)

        # Assemble the right hand side
        f.t = t
        L = (u0 + dt*f)*v*dx

        # Compute the solution
        solve(B == L, u)
        if save:
            results << (u, t)

        # Update
        u0.assign(u)

    # Return Solution
    return u

def heateqn_thetamethod(t0 = 0.0, T = 5.0, dt = 1e-2, a = 0.1, theta = 1.0,
                            save = False, fname = 'heat/bwdeuler_theta.pvd',
                            prnt = True, cmp = False):
    # Create problem geometry
    u0, f, V, t, tsteps = create_heateqn_problem(t0 = 0.0, T = 5.0, dt = 1e-2)

    # Define the variational problem
    u = TrialFunction(V)
    v = TestFunction(V)
    B = u*v*dx
    if theta > 0.0:
        B += dt*theta*a*dot(grad(u), grad(v))*dx

    # Export the initial data
    u = Function(V, name='Temperature')
    u.assign(u0)
    if save:
        results = File(fname)
        results << (u, t)

    # Time stepping
    f.t = t0
    for k in range(tsteps):

        t_last = t0 + k*dt # Last time
        t_curr = t0 + (k+1)*dt # Current time
        if prnt:
            print('Step = ', k+1, '/', tsteps , 'Time =', t_curr)

        # Assemble the right hand side
        #   -> Currently, f.t == t_last from the last loop itereration
        L = (1-(1-theta)*dt*a)*u0*v*dx
        if theta < 1.0:
            L += dt*(1-theta)*f*v*dx

        # Update f and add contribution to L
        f.t = t_curr
        if theta > 0.0:
            L += dt*theta*f*v*dx

        # Compute the solution
        solve(B == L, u)
        if save:
            results << (u, t_curr)
        # Update
        u0.assign(u)

    if cmp:
        # Compare the norm of the difference between the solution calculated
        # with the theta-method and with the backward euler method
        u_BE = heateqn_bwdeuler(t0 = t0, T = T, dt = dt, a = a,
                                prnt = prnt, save = save)
        print('\n')
        print('u_th_max = ', np.max(np.abs(u.vector().get_local())))
        print('u_BE_max = ', np.max(np.abs(u_BE.vector().get_local())))
        print('||u-u_BE||_L2 = ', errornorm(u, u_BE, 'L2'))
        print('\n')

# ---------------------------------------------------------------------------- #
# Stricter log level to avoid so much printing
# ---------------------------------------------------------------------------- #
set_log_level(30)

# ---------------------------------------------------------------------------- #
# Q2(b): theta-method - backward euler consistency check with Q2(a)
# ---------------------------------------------------------------------------- #
heateqn_thetamethod(t0 = 0.0, T = 5.0, dt = 1e-2, a = 0.1, theta = 1.0,
                    fname = 'heat/thetaMethodBwdEuler.pvd',
                    save = True, cmp = True, prnt = True)

# ---------------------------------------------------------------------------- #
# Q2(b): theta-method - forward euler and Crank-Nicolson
# ---------------------------------------------------------------------------- #
heateqn_thetamethod(t0 = 0.0, T = 5.0, dt = 1e-2, a = 0.1, theta = 0.0,
                    fname = 'heat/thetaMethodFwdEuler.pvd',
                    save = True, cmp = True, prnt = True)

heateqn_thetamethod(t0 = 0.0, T = 5.0, dt = 1e-2, a = 0.1, theta = 0.5,
                    fname = 'heat/thetaMethodCrankNicolson.pvd',
                    save = True, cmp = True, prnt = True)

# ---------------------------------------------------------------------------- #
# Q2(c): theta-method - forward euler, shorter duration T and timestep dt
# ---------------------------------------------------------------------------- #
heateqn_thetamethod(t0 = 0.0, T = 0.1, dt = 1.25e-4, a = 0.1, theta = 0.0,
                    fname = 'heat/thetaMethodFwdEuler__T_0p1__dt_0p000125.pvd',
                    save = True, cmp = True, prnt = True)

heateqn_thetamethod(t0 = 0.0, T = 0.1, dt = 1.00e-4, a = 0.1, theta = 0.0,
                    fname = 'heat/thetaMethodFwdEuler__T_0p1__dt_0p0001.pvd',
                    save = True, cmp = True, prnt = True)
```