COMPUTATIONAL ASSIGNMENTS
(January 9, 2025)

Instructions

Detailed instructions on what to provide in your report can be found in the text below. In general, try to: be concise, answer all questions, if asked provide plots (of reasonable size) with all information on what is plotted and a discussion of your results. For some tasks, you should also provide your code. Do not forget that you can verify your codes on simple examples. You may write your programs in matlab, octave, or python.

This file contains 6 mandatory computational tasks (and one optional task offering a bonus point at the exam). The tasks reflect the content on the lecture. For each computational task, **a report must be sent on Canvas ten days after the corresponding material has been seen in the lecture** (15 days for the last mandatory task). I will try to inform you and give a deadline via Canvas when the corresponding material has been seen in the lecture.

It is possible to work alone or in a group of 2 students (in exceptional cases 3). You can form groups yourself or post a message on piazza if you need a partner. In principle, it is not allowed to change group members once the first lab report is submitted. However, in case of problems in a group, please inform us as soon as possible and we will try to find a solution.

Submission of your lab reports (if you are working in a group, submit only one report) are done via Canvas. If a report is not good enough (or not sent on time), students have the possibility to (re)-submit an improved version of their report. **The final date for submitting all reports is on the 22th of March 2025**. It is not possible to submit a report more than two times.

Plagiarism will not be tolerated, see link. Cases of strong suspicion of cheating or plagiarism will be reported and the whole project could then be considered as a fail.

We are all here to study and learn something new. We thus do not allow the use of chatbots for these computational assignments. Further information on AI systems at Chalmers can be found here.

In order to make each report unique, a small real parameter **epsSTUD** is calculated for each student/group individually according to the formula:

$$\text{epsSTUD} := \begin{cases} A \cdot 10^{-2} & \text{if} \quad A < 10 \\ A \cdot 10^{-3} & \text{if} \quad A \geq 10, \end{cases}$$

where A is the number corresponding to the last two digits of your social security number (ex: 7710091234, then $A = 34$ and thus $\text{epsSTUD} = 34 \cdot 10^{-3}$). If two or three students are working on the same lab, choose one number A. Include this parameter in your report. Only one report needs to be submitted (with the names of all involved students in the group).

A matlab tutorial and a guide can be found here and here (via Chalmers library).

In case of problems in your codes, first read the error messages given by matlab (if you get some). Then, use the commands **doc** <*command*> or **help** <*command*> before contacting a teaching assistant. Use the command **whos** to display the information about your variables. You can display the value of a particular variable by typing its name.

Office hours for Michael Roop: Wednesdays $10-16$ (please send an email in advance). Michael can organise one or two help-sessions if requested.

At this place, I would like to thank Ioanna Motschan-Armen, Michael Roop, and Johan Ulander for helpful suggestions on this document.

Please report any typos and suggestions for improvement to david.cohen@chalmers.se.

1. Polynomial interpolation in 1d

This computer lab is inspired by exercise 3.3 in Scientific Computing with matlab and Octave by A. Quarteroni and F. Saleri and by one example in Scientific Computing by M. Gander, W. Gander, and F. Kwok.

Goals. Application of polynomial interpolation of data.

Report. Your report must contain: a description of the problems in your own words and your solutions (Task 1, Task 2, and Task 3).

In many applications, one is interested in passing a polynomial through a set of data points $(x_i, y_i)_{i=0}^n$ for a given positive integer n and distinct nodes x_i. This polynomial interpolant is denoted by π_n below and therefore satisfies $y_i = \pi_n(x_i)$ for $i = 0, \dots, n$. If the data represent the values of a continuous function f, one denotes the polynomial interpolant by $\pi_n f$.

From the lecture, one knows that

$$\pi_n(x) = \sum_{k=0}^{n} y_k \lambda_k(x),$$

with the Lagrange polynomials λ_k, for $k = 0, \dots, n$. The matlab command **polyfit** can help you to find the polynomial interpolant. For two vectors x and y comprising the set of data, the coefficients of the polynomial interpolant π_n are given by $c = polyfit(x, y, n)$, where $c(1)$ if the coefficient in front of x^n, $c(2)$ the coefficient in front of x^{n-1}, etc.

Task 1. Consider the data set $\{(2,2), (3,6), (4,5), (5,5), (6,6)\}$.

a) Plot the data on the plane using the following

```
x= ... ; % vector of nodes
y= ... ;
figure(),
plot( .. , .. ,'r*','MarkerSize',16)
% MarkerSize increases the size of the markers
% legend (axis label, title)
...
```

b) Plot the polynomial interpolant of degree 4 using the matlab command **polyval** and the following

```
...
c=polyfit( .. );
xx=linspace(min(x),max(x),50); % 50 equidistant points where one plots
yy=polyval(c, .. ); % Output: yy -> values of polynomial
% plot of the polynomial interpolant with the data from a)
...
% legend (axis label, legend, title)
...
```

Include only one plot containing both the data and the polynomial interpolant in your report. For this, you may use the commands **hold on** and **hold off**.

Task 2. The following data are related to the life expectancy of citizens of two European regions:

	1975	1980	1985	1990
Western Europe	72.8	74.2	75.2	76.4
Eastern Europe	70.2	70.2	70.3	71.2

Use the polynomial interpolant of degree 3 to estimate the life expectancy in 1970, 1983 and 1988. Plot your numerical results. It is known that the life expectancy in 1970 was 71.8 years for the citizens of West Europe, and 69.6 for those of East Europe. Compare with your numerical results (don't just display the results but discuss briefly).

Task 3. Write your own matlab function that interpolates with the Lagrange interpolation polynomials. You cannot use built-in matlab functions like **polyval**, etc. Compare the results of your own implementation with the results of **polyval** (that is provide the error between the two quantities), using the examples above for instance. The following may help:

```
function yy=MyLagrangeInterpol(x,y,xx)
% MyLagrangeInterpol interpolates using Lagrange polynomials
% Data set (x,y)
% Form the interpolation polynomial P and interpolates the values yy=P(xx)

n=length(x); nn=length(xx);
for i=1:nn
  yy(i)=0;
  for k=1:n
    yy(i)=yy(i)+y(k)* ... use the fct "prod" to compute Lagrange poly. ... ;
  end
end
end
```

2. Numerical integration in 1d

The last task of this computer lab is inspired by a note of A. Kværnø and M. Grasmair.

Goals. Use basic numerical methods (known as quadrature formulas) to approximate integrals of given functions.

Report. Include your solutions and your codes in your report (Task 1 and Task 2 and Task 3).

Let two real numbers $a < b$ and a continuous function $f\colon [a,b] \to \mathbb{R}$ be given. Consider a (large) positive integer N and a discretisation of the interval $[a,b]$ by $a = x_0 < x_1 < \ldots < x_{N-1} < x_N = b$ with $h_k = x_k - x_{k-1}$ for $k = 1, 2, \ldots, N$. The integral of f can then be decomposed as

$$\int_a^b f(x)\,\mathrm{d}x = \sum_{k=1}^{N} \int_{x_{k-1}}^{x_k} f(x)\,\mathrm{d}x.$$

We want now to find an approximation of the integrals $\int_{x_{k-1}}^{x_k} f(x)\,\mathrm{d}x$ on every subintervals $[x_{k-1}, x_k]$. Using the transformation

$$x = x_{k-1} + t h_k, \quad \mathrm{d}x = h_k\,\mathrm{d}t, \quad \text{for} \quad 0 < t < 1,$$

one gets the relation

$$\int_{x_{k-1}}^{x_k} f(x)\,\mathrm{d}x = h_k \int_0^1 f(x_{k-1} + t h_k)\,\mathrm{d}t.$$

The problem then reduces in finding a numerical approximation of the integral

$$\int_0^1 g(t)\,\mathrm{d}t.$$

We have seen in the lecture how this approximation can be done using, e.g., the midpoint rule, the trapezoidal rule or the Simpson rule.

Task 1. Let us illustrate the above for the composite trapezoidal rule. For ease of presentation, consider a uniform grid with (constant) $h = (b-a)/N$.

On a paper, for yourself, apply the trapezoidal rule to each subintervals $[x_{k-1}, x_k]$ (or to $[0,1]$ as seen in the lecture and transform back). We then obtain an approximation of the integral of f over $[a,b]$ by the *composite trapezoidal rule*

$$T(h) = h\left(\frac{1}{2}f(x_0) + f(x_1) + \ldots + f(x_{N-1}) + \frac{1}{2}f(x_N)\right) \approx \int_a^b f(x)\,\mathrm{d}x.$$

You should now implement the composite trapezoidal rule. The following matlab function could be of help:

```
function t=mytrapezoidalrule(f,a,b,N)
% computes approximation of int_a^b f(x) dx
% using the composite trapezoidal rule
...
h= .. ;
...
t= .. ; %result of the composite trapezoidal rule
end
```

Consider the two cases $N = 10$ and $N = 100$ and test your code on some simple example function f of your choice as well as on the following integral

$$\int_0^1 \frac{xe^x}{(x+1)^2}\,\mathrm{d}x = \frac{e-2}{2}.$$

Task 2. Repeat the above and implement the *composite midpoint rule* seen in the lecture.

Task 3. It is known that the error of the composite trapezoidal rule (and of the composite midpoint rule) verifies

$$|T(h) - \int_a^b f(x)\,\mathrm{d}x| \leq C\frac{1}{N^2}$$

for nice enough f, see for instance wikipedia.org.

We will now illustrate this convergence property by showing that if N is increased by a factor 2, then the error of the composite trapezoidal rule will be reduced by a factor of $1/2^2$. To do so, use your code from Task 1 with a simple choice of f and apply the composite trapezoidal rule with $N = 1, 2, 4, 8, 16, 32, 64, 128$. In a table, display N, the error, and the reduction factor. Observe that this factor is $1/2^2$. The following could be of use

```
...
for n=0:7
  N=2^n;
  T= ... ; % numerical solution
  err = abs( ... ) ; % error
  if n ~= 0
    X=['N=',num2str(N),'   err=',num2str(err),'   reduction factor=',
       num2str(err/err_prev)];
    disp(X);
  end
  err_prev= ... ; % previous error
end
```

3. Initial value problems

Goal. Implement the explicit Euler method for a simple linear IVP and for a system of IVP.

Report. Your report must contain the following: a short text describing the problem in your own words and its exact solution, a plot of the exact solution, a plot of the numerical solutions given by explicit Euler's scheme, a convergence plot and your code for the convergence plot (Task 1). A concise proof for the theoretical part and two plots containing the results of your implementation (Task 2).

Task 1. You don't feel well after eating a kebab downtown ...it seems that you have picked up a parasite that grows exponentially fast until treated. Upon returning home there are 5 parasites in you. The growth parameter for the parasites in your body is $k = 0.1 + \text{epsSTUD}$.

a) Use the Malthusian growth model discussed in the lecture to model the evolution of the parasites in your body. For yourself, on a paper, write down the differential equation corresponding to this model as well as its exact solution. Plot the exact solution of the problem on the time interval $[0, 20]$ using an M-file. The following may be of use

```
clear all
...
p0= .. ; % initial number of parasites
k= .. ; % population growth rate
tExact=[0:0.05:20]; % time interval
pExact= .. ; % exact solution as a function of tExact
figure(),plot(tExact,pExact) % plot solution wrt time
xlabel('Time','FontSize',15) % x-axis with nice fonts
ylabel( .. )
legend( .. ) % legend
title( .. ) % title
% can be used to save the plot in .jpg
% see also the extension .eps which may offer better quality
print -djpeg90 task3a.jpg
```

b) Complete your M-file from the first part with an implementation of the explicit Euler method. You should plot the numerical approximations given by Euler's method with the step sizes $h = 0.5, 0.25$, and $h = 0.1$ on the interval $[0, 20]$ along with the exact solution in the same figure. The following may be of use

```
...
h=0.5; % step size
N= .. ; % compute the number of steps used by Euler's method
 .. ; % initial time and initial value
for n=1:N
   .. ; % compute one step of the method
  tEuler1(n)= .. ; % store the discr. times for plot, see below
  pEuler1(n)= .. ; % Euler approx of pExact(t_n) for step size h
end
% do the same for the next step size
h=0.25;
...
% plot of the exact and numerical sol
```

```
tExact=[0:.1:20]; % time interval
pExact= .. ; % exact solution
figure(),plot(tExact,pExact,tEuler1,pEuler1, .. )
legend( .. ) % legend, etc
...
```

c) It is known that the explicit Euler method is a numerical integrator of order 1, that is, one has

$$|y_N - y(T)| \leq Ch^1,$$

for sufficiently nice problems, see for instance wikipedia.org. Here, h denotes the step size and is given by $h = \frac{T}{N}$, where T is the final time of computation and N is a given positive integer.

We will now illustrate this convergence property at the final time $T = 1$ using the code that you already have implemented in b). The following could be of use

```
% compute pEuler1, pEuler2, pEuler3
% for h=0.5, 0.25, and 0.1 using code in b)
...
hplot=[0.5 0.25 0.1]; % array of step sizes
pExact= ... ; % exact solution at T=1
Errplot=[abs( ... ) ... ]; % array of errors
figure(), loglog(hplot,Errplot,'b-+',hplot,hplot,'--r')
xlabel('Time step sizes','FontSize',15)
ylabel('Errors Euler','FontSize',15)
legend('Euler','Reference line slope 1','Location','NorthWest')
```

Why did we use the **loglog** command above? Connect your reasoning to the reference line of slope 1 and the above error estimate.

Task 2. After an accident at the Chemistry and Chemical Engineering building, 10 zombies escape and attack the building of Mathematical Sciences, where 500 students are peacefully studying. Let us consider the following model for the evolution of humans and zombies at Chalmers:

$$\begin{aligned} H'(t) &= -\beta H(t)Z(t) \\ Z'(t) &= \beta H(t)Z(t) + \zeta R(t) - \alpha H(t)Z(t) \\ R'(t) &= \alpha H(t)Z(t) - \zeta R(t), \end{aligned}$$

where $H(t), Z(t)$, resp. $R(t)$, denotes the levels of humans, zombies, resp. removed ("dead" zombies, which may return as zombies) at time t. Further, the positive parameters

α deals with human-zombie encounters that remove zombies
β deals with human-zombie encounters that convert humans to zombies
ζ deals with removed zombies that revert to zombie status.

Take $\alpha = 0.005 \cdot \text{epsSTUD}$, $\beta = 0.01$ and $\zeta = 0.02$ and $R(0) = 0$.

a) (Theoretical part) Prove that the total population $H(t) + Z(t) + R(t)$ of the exact solution to the above system of differential equations remains constant in time. Include this theoretical argument in your report.
Hint: What is the derivative of a constant?

b) (Implementation) Use the explicit Euler method with step sizes $h = 0.65$ and $h = 0.1$ to approximate the exact solution of the above system of ODEs. Plot the evolution of humans, zombies, and removed

with respect to time until time $T_{\text{end}} = 10$. In another figure, plot the evolution of the total population (along your numerical solutions given by the explicit Euler method). What are your conclusions?

4. FEM for two-point BVP

This computer lab is inspired by the one given by Fardin Saedpanah in 2019 *and by an assignments by M. Asadzadeh in* 2020.

Goals. Approximate solutions to BVP by finite element methods.

Report. Your report must contain: The code for the finite element method (Task 1) as well as the plots for the numerical solutions (Task 1 and Task 2).

Task 1. Problem: Consider the stationary convection-diffusion problem

$$\begin{aligned} -Du''(x) + \frac{1}{2}u'(x) &= 1, \quad 0 < x < \pi, \\ u(0) = u(\pi) &= 0, \end{aligned} \tag{1}$$

where the diffusion constant D is positive. Consider the uniform partition $\mathcal{T}_h$ of the interval $[0, \pi]$ with $m+1$ elements (of length $h = \frac{\pi}{m+1}$). Compute the matrix $\mathbf{A}$ and load vector $\mathbf{b}$ for the cG(1) approximation of the problem (1). Use the computed matrix $\mathbf{A}$ and the load vector $\mathbf{b}$ to find a numerical approximation of this BVP.

Questions:

a) Implement the FEM for problem (1), as described above, in matlab. That is, write an M-file (or a script file) which, given a diffusion coefficient D and a mesh size h (or number of elements m), computes the approximation vector $\boldsymbol{\xi}$ by solving the linear system of equations $\mathbf{A}\boldsymbol{\xi} = \mathbf{b}$.

Hint: The course page in Canvas may offer a template for such matlab file.

b) The Péclet number Pe is defined as the ratio between convective and diffusive transport. In this case, we have the following relation (with the speed of convection $v = 1/2$ and length of interval π)

$$\mathrm{Pe} = \frac{\text{convection}}{\text{diffusion}} = \frac{\frac{1}{2}\pi}{D} \propto \frac{1}{D}.$$

Study, by comparing the FE approximation and the exact solution to problem (1) (calculated by hand), which mesh sizes h is required to get a good FE approximation in the following two cases:

Case 1: $\mathrm{Pe} \approx 1$, that is $D \approx 1$.

Case 2: $\mathrm{Pe} \gg 1$, that is $D \ll 1$ (convection dominated).

Present your results.

Hint: To plot a piecewise linear FE approximation in matlab is not difficult, since $u_h(x_j) = \xi_j$ and matlab draws straight lines between nodes automatically, but don't forget to include the BC: $u_h(0) = u_h(\pi) = 0$!

Task 2. Problem: Write a program that computes the $cG(1)$ and $cG(2)$ finite element approximations of the two-point boundary value problem

$$\begin{cases} -u''(x) = f(x) \quad \text{in} \quad (0,1), \\ u(0) = u(1) = 0, \end{cases}$$

where f is a given function by the user.

Questions:

a) Test your codes for the choice $f(x) = \pi^2 \sin(\pi x)$ where the exact solution can easily be found and is given by $u(x) = \sin(\pi x)$. Present the exact as well as the numerical solutions (with an appropriate choice for the mesh h) in one figure. Hints on the derivation of the $cG(2)$ finite element method can be found on the canvas page of the course and during the exercise sessions.

b) In order to verify numerically the order of convergence of the $cG(1)$ approximation compare the exact solution to the above BVP with the numerical approximations using several meshes $h = (b-a)/(2^\ell + 1)$ (that is $m = 2^\ell$) for $\ell = 1, 2, 3, 4, 5$. The results are then presented in a loglog plot. First, use the function **integral** to compute the load vector. Next, use the function **interp1**, on a fine grid, in order to get a piecewise

interpolation of your finite element vector. Then, evaluate the exact solution on this fine grid. Finally, compute the error between the FE interpolant and the exact solution. You can use the following to start:

```
1
2  .. % boundary, load function, etc.
3  u=@(x) sin(pi*x); % exact solution
4
5  err=zeros(1,5); % initialisation of vect. of errors
6  for l=1:5
7    m=2^l;  % number of subintervals = m+1
8    h=(b-a)/(m+1);
9
10   % fine grid
11   xx=linspace(a,b,10*m);
12
13   % stiffness matrix, load vector,
14   % solution linear system
15   U= .. ; % FE solution
16
17   % FE and exact sol. interpolants
18   UU=interp1(x, ... ,xx);
19   uu=u(...);
20
21   err(l)=norm(UU'-uu,Inf); % error of step l
22 end
23
24 %% plot of errors
25 hl=(b-a)./(2.^(1:5)+1);
26 figure()
27 loglog(hl,err,'b*-'), hold on
28 % reference slopes of 1 and 2
29 loglog(hl,hl.^1,'b.-',hl,hl.^2,'r--'), hold off
30 .. % label etc
```

5. PDE IN 1d

This computer lab is inspired by the one given by Fardin Saedpanah in 2019.

Goal. Discretise numerically the heat equation with finite elements in space and implicit/backward Euler in time.

Report. Your report must contain the codes that you implemented as well as answers to the questions below.

Problem. The aim of this task is to study the FE approximation of the heat equation

$$\begin{cases} u_t(x,t) - u_{xx}(x,t) = f(x,t), & 0<x<1, \quad 0<t<T, \\ u_x(0,t) = u(1,t) = 0, & 0<t<T, \\ u(x,0) = u_0(x), & 0<x<1. \end{cases} \tag{2}$$

Here, the unknown function $u(x,t)$ is the temperature distribution, and $f(x,t)$ and $u_0(x)$ are given source term, resp. initial values. The end time $T>0$ is a given real number.

Observe that we impose homogeneous Neumann BC for $x=0$, that is $u_x(0,t)=0$, and homogeneous Dirichlet BC for $x=1$, that is $u(1,t)=0$.

Variational formulation.

In order to get a numerical approximation of solutions to the PDE (2), we start with deriving a variational formulation. We choose the space of test functions $V = \{w\colon [0,1]\to\mathbb{R} : \|w\|^2_{L^2(0,1)} + \|w'\|^2_{L^2(0,1)} < \infty \text{ and } w(1) = 0\}$. This choice is taken in order to match the BC of the PDE (no condition on the test function at $x=0$ since one has Neumann BC). We then multiply the PDE (2) with a test function $v\in V$ and integrate (with respect to x) over the space domain $(0,1)$. We obtain the following

$$\int_0^1 u_t(x,t)v(x)\,\mathrm{d}x - \int_0^1 u_{xx}(x,t)v(x)\,\mathrm{d}x = \int_0^1 f(x,t)v(x)\,\mathrm{d}x. \tag{3}$$

Performing a partial integration in the second integral in (3) gives us

$$\int_0^1 u_t(x,t)v(x)\,\mathrm{d}x - [u_x(x,t)v(x)]_{x=0}^{x=1} + \int_0^1 u_x(x,t)v_x(x)\,\mathrm{d}x = \int_0^1 f(x,t)v(x)\,\mathrm{d}x. \tag{4}$$

Since $v(1)=0$ and $u_x(0,t)=0$, the above reduces to

$$\int_0^1 u_t(x,t)v(x)\,\mathrm{d}x + \int_0^1 u_x(x,t)v_x(x)\,\mathrm{d}x = \int_0^1 f(x,t)v(x)\,\mathrm{d}x. \tag{5}$$

This equation then gives us the following variational formulation for the PDE (2):

For all fixed $t\in(0,T]$, find $u(\cdot,t)\in V$ such that equation (5) is fulfilled for all $v\in V$.

Observe that the above variational formulation is only in the spatial variable x. The time variable t is considered as a fixed parameter.

Spatial discretisation.

As in one of the computer lab above, we use the above variational formulation to get a FEM discretisation in space of the problem. We consider the partition $\mathcal{T}_h : jh$ for $j=0,1,\dots,m+1$ of the interval $0\le x\le 1$ in $m+1$ subintervals of the same length $h=\frac{1}{m+1}$. We then choose the space V_h to be a subspace of V consisting of continuous functions that are piecewise linear on the partition $\mathcal{T}_h$. The FE formulation then reads: For all fixed $t\in(0,T]$, find $u_h(\cdot,t)\in V_h$ such that

$$\int_0^1 u_{h,t}(x,t)v_h(x)\,\mathrm{d}x + \int_0^1 u_{h,x}(x,t)v_h'(x)\,\mathrm{d}x = \int_0^1 f(x,t)v_h(x)\,\mathrm{d}x, \qquad \forall v_h\in V_h. \tag{6}$$

Let $\{\varphi_j\}_{j=0}^m$ be the standard basis of V_h consisting of the usual hat functions (observe that φ_{m+1} is not included since $u(1,t) = u_h(1,t) = 0$). One can then write the numerical approximation u_h as

$$u_h(x,t) = \xi_0(t)\varphi_0(x) + \xi_2(t)\varphi_2(x) + \ldots + \xi_m(t)\varphi_m(x) = \sum_{j=0}^{m} \xi_j(t)\varphi_j(x). \tag{7}$$

Note that the coefficients $\xi_j(t)$ depend on the time variable but not on the spatial variable.

Inserting equation (7) in equation (6) and choosing as test functions $v_h = \varphi_i$, $i = 0, 1, \ldots, m$, one then obtains

$$\int_0^1 \Big(\sum_{j=0}^{m} \dot{\xi}_j(t)\varphi_j(x)\Big)\varphi_i(x)\,\mathrm{d}x + \int_0^1 \Big(\sum_{j=0}^{m} \xi_j(t)\varphi_j'(x)\Big)\varphi_i'(x)\,\mathrm{d}x = \int_0^1 f(x,t)\varphi_i(x)\,\mathrm{d}x$$

for $i = 0, 1, \ldots, m$.

A rearrangement gives

$$\sum_{j=0}^{m} \dot{\xi}_j(t)\Big(\int_0^1 \varphi_j(x)\varphi_i(x)\,\mathrm{d}x\Big) + \sum_{j=0}^{m} \xi_j(t)\Big(\int_0^1 \varphi_j'(x)\varphi_i'(x)\,\mathrm{d}x\Big) = \int_0^1 f(x,t)\varphi_i(x)\,\mathrm{d}x$$

which is a system of $m+1$ ODEs (one for each $i = 0, \ldots, m$) with $m+1$ unknown functions $\{\xi_j(t)\}_{j=0}^m$. In matrix notation, one gets

$$M\dot{\boldsymbol{\xi}}(t) + S\boldsymbol{\xi}(t) = \mathbf{F}(t). \tag{8}$$

Here, $\boldsymbol{\xi}(t)$ is the vector containing the nodal values $\xi_j(t)$ of the spatial approximation $u_h(x,t)$.

The elements of the mass matrix M and stiffness matrix S are given by

$$m_{ij} = \int_0^1 \varphi_i(x)\varphi_j(x)\,\mathrm{d}x, \quad s_{ij} = \int_0^1 \varphi_i'(x)\varphi_j'(x)\,\mathrm{d}x, \qquad i,j = 0,1,\ldots,m, \tag{9}$$

see for instance the lecture notes or Chapter 5.3 in the course book. The elements of the load vector $\mathbf{F}(t)$ are given by

$$F_i(t) = \int_0^1 f(x,t)\varphi_i(x)\,\mathrm{d}x, \qquad i = 0,1,\ldots,m. \tag{10}$$

Discretisation in time.

The final step of the fully discrete solution (discrete in space and in time) is to solve the *semi-discrete problem* (that is, discrete in space) $M\dot{\boldsymbol{\xi}}(t) + S\boldsymbol{\xi}(t) = \mathbf{F}(t)$. To this end, we introduce a partition $0 = t_0 < t_1 < t_2 < \ldots < t_n = T$ of the time interval $[0,T]$ in n subintervals of same length $k = T/n$ (hence $t_\ell = \ell k$, $\ell = 0, \ldots, n$). We then approximate the time derivative $\dot{\boldsymbol{\xi}}(t)$ with the finite difference quotient

$$M\frac{\boldsymbol{\xi}^{(\ell)} - \boldsymbol{\xi}^{(\ell-1)}}{k} + S\boldsymbol{\xi}^{(\ell)} = \mathbf{F}(t_\ell), \qquad \ell = 1, \ldots, n \tag{11}$$

where $\boldsymbol{\xi}^{(\ell)} \approx \boldsymbol{\xi}(t_\ell)$ for $\ell = 0, \ldots, n$. A reorganisation of the above gives the iterative scheme

$$(M + kS)\boldsymbol{\xi}^{(\ell)} = M\boldsymbol{\xi}^{(\ell-1)} + k\mathbf{F}(t_\ell), \tag{12}$$

which is called the *backward Euler scheme*. There are several possibilities to choose the initial value $\boldsymbol{\xi}^{(0)}$, but the easiest one is to consider a linear interpolation of $u_0(x)$, that is $\xi_j^{(0)} = u_0(x_j)$, $j = 0, \ldots, m$. One can use various quadrature formulas for approximating the integrals in $F_i(t_\ell)$. In other words, one starts with some initial values and some quadrature formula, and then compute successively the approximation $\boldsymbol{\xi}^{(1)}$, $\boldsymbol{\xi}^{(2)}$, …, $\boldsymbol{\xi}^{(n)}$ using the numerical scheme (12).

Questions.

Implement the numerical scheme (12) for an approximation of solutions to the above heat equation. In order to validate your codes, determine the function $f(x)$ and the initial value $u_0(x)$ in order to get the exact solution $u(x,t) = -\frac{epsSTUD}{2}\left(tx^2 - t\right)$ to the above heat equation. For 5 different values of discrete times t_ℓ, plot and

compare the exact and numerical solutions (in 5 different plots). In order to visualize your results you may use the command `plot` at each timestep, or the command `surf`.

Hint: You can use a quadrature formula implemented in matlab (`quad` or `trapz`) to compute the load vector. Or you can use your own `mytrapezoidalrule.m` implementation from a previous computer lab.

Hint: The course page in Canvas may offer a template to start this assignment.

6. PDE IN $2d$

This computer lab is inspired by a project by Martin Gander.

Goal. Cook a chicken in a microwave oven.

Report. Your report must contain the codes that you implemented as well as answers to the questions below.

Problem. The electro-magnetic field in a microwave oven is given by Maxwell's equations:

$$\begin{aligned} \nabla \times \mathbf{E} &= -\mu \mathbf{H}_t \\ \nabla \times \mathbf{H} &= \varepsilon \mathbf{E}_t + \sigma \mathbf{E}, \end{aligned}$$

where the vector $\mathbf{E}$ is the electric field and the vector $\mathbf{H}$ is the magnetic field. The constants ε, σ and μ are the permittivity, the electrical conduction and the permeability.

In the air, one has $\varepsilon_l = 8.85 \cdot 10^{-12}, \sigma_l = 0$ and $\mu_l = 4\pi \cdot 10^{-7}$. In the chicken, one has $\varepsilon_p = 4.43 \cdot 10^{-11}, \sigma_p = 3 \cdot 10^{-11}$ and $\mu_p = 4\pi \cdot 10^{-7}$.

(1) We assume that the solution to the above PDE is time-harmonic:

$$\mathbf{E} = \mathbf{E}(x, y, z)\mathrm{e}^{\mathrm{i}\omega t}, \quad \mathbf{H} = \mathbf{H}(x, y, z)\mathrm{e}^{\mathrm{i}\omega t},$$

with $\omega = 2\pi f := 2\pi 2.45 \cdot 10^9$. In this case, show that Maxwell's equations then read

$$\begin{aligned} \nabla \times \mathbf{E} &= -\mathrm{i}\omega\mu \mathbf{H} \\ \nabla \times \mathbf{H} &= \mathrm{i}\omega\tilde{\varepsilon} \mathbf{E}, \end{aligned}$$

with $\tilde{\varepsilon}$ to be determined.

(2) We assume that $\partial_z \mathbf{E} = \partial_z \mathbf{H} = 0$. Using this, derive an equation for the third component of the electric field E_3. You should obtain the Helmoltz's equation in $2d$:

$$\Delta E_3 + \omega^2 \mu \tilde{\varepsilon} E_3 = 0.$$

We shall now discretize the above equation with the FEM!

(3) First, let us generate a grid:

```
function [N,T,P]=mygrid(G,w)
%% Grid generation
% G -> computational domain (G=0,G=1,G=2, oder G=3)
% w -> frequency of the Helmholtz equation
% N -> list of nodes (with coordinates x and y)
% T -> list of triangles from the list N
%      the first 3 entries are the nodes
%      the next 3 entries are 0 or 1:
%      1 for real boundary and 0 else
% P -> contains the material constant mu*epsilon for every triangle
...
```

This function generates a triangulation (N,T) of the unit square (choice G=0), a triangle (choice G=1), an empty microwave oven, that is a rectangle, (choice G=2), and a chicken in a microwave oven (choice G=3, *the datas are provided on Canvas*).

Hint: For the unit square, one has
N=[0 0;1 0; .. ; ..]. T=[1 2 3 1 0 .. ;2 4 .. ..]. P=[1 1].

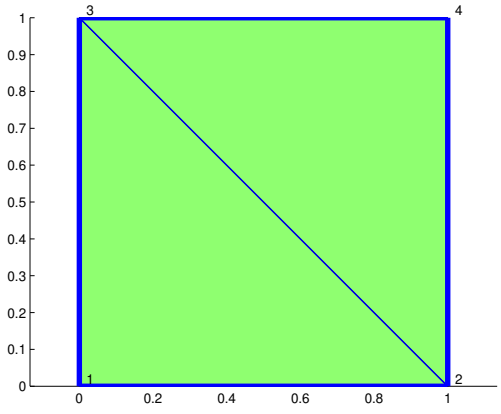
(4) In order to visualize the triangulation, one can use the following function:

```
function plotmygrid(N,T,P)
%% Visualization of the grid
% N -> list of nodes
% T -> list of triangles
% P -> contains the material constant mu*epsilon for each triangle
%      encoded with colors
%      (in patch, use abs(P(i)) as third component)
...
```

All triangle edges that are at the boundary of the computational domain should be bold colored. For a small number of nodes, say < 100, write also the corresponding number of the nodes on it.

Hint: The matlab functions LINE, PATCH, TEXT, NUM2STR *may help.*

For the unit square, one should get the following image:



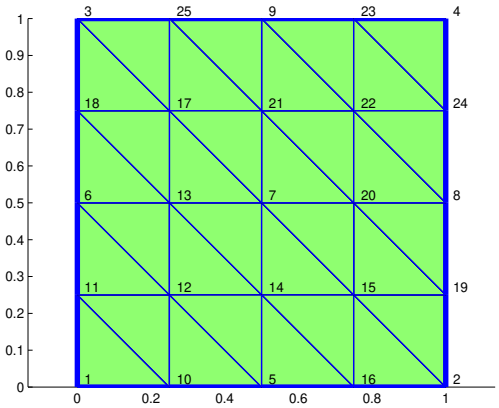
(5) Our next task if to implement a simple procedure for grid-refinement. We just divide all triangles into four smaller ones. To do so, for each triangle, we need to add three nodes (f.ex. n4=0.5*(n1+n2)) and with this we can define four new triangles.

Remark: For each new triangle, one must save the information on the boundary and the material constant.

```
function [Nr,Tr,Pr]=mygridrefinement(N,T,P)
%% simple grid refinement
% Nr, Tr, Pr -> New nodes, triangles, and material constants

% Start the new node list with old once
Nr=N; nn=size(N,1);
% Start triangulation from scratch
Tr=[]; nt=0; Pr=[];

% Construction of 4 new triangles
for j=1:size(T,1),
  % Nodes of triangle j
  i=T(j,1:3);
  % Coordinates of nodes of triangle j
```

```
n= ... ;
% 3 new nodes
n(4,:)=(n(1,:)+n(2,:))/2;
n(5,:)=(n(1,:)+n(3,:))/2;
n(6,:)=(n(2,:)+n(3,:))/2;
% Insert new nodes in Nr if don't have it
for k=4:6,
  % check x-coord.
  l=find(Nr(:,1)==n(k,1));
  % check y-coord.
  m=find(Nr(l,2)==n(k,2));
  if isempty(m), % isempty returns 1 if m is empty
                 % 0 else
    nn=nn+1;
    Nr(nn,:)=n(k,:);
    i(k)= ... ;
  else
    % node found
    i(k)=l(m);
  end;
end;

% Insert 4 new triangles
Tr(nt+1,:)=[i(1) i(4) i(5) T(j,4) 0 T(j,6)];
Tr(nt+2,:)=[ ...  0 0 0];
Tr(nt+3,:)=[ ...  0 T(j,4) T(j,5)];
Tr(nt+4,:)=[ ...  T(j,5) T(j,6) 0];

Pr(nt+1:nt+4)=P(j);
nt=nt+4;
```

After two refinements of the above unit square, one should get the following:



(6) (*The finite element method*) We first compute the element stiffness matrix and the element mass matrix for a triangle with nodes $(x_1, y_1), (x_2, y_2)$ and (x_3, y_3) (the nodes are ordered in the counterclockwise direction):

```
function Se=elementstiffmatrix(t)
%% Computes the element stiffness matrix for
%% the triangle t
% t=[x1 y1;x2 y2;x3 y3] -> triangle
...
```

and

```
function Me=elementmassmatrix(t)
%% Computes the element mass matrix for
%% the triangle t
% t=[x1 y1;x2 y2;x3 y3] -> triangle
% one computes by hand the integrals int_Triangle phi_i phi_j
...
```

(7) Next, we use the solver for the Helmholtz equation with Dirichlet boundary condition $g(x, y)$ on a triangulation N, T, P:

```
function [u,K,M]=FEHelmholtz2D(g,N,T,w,P)
%% FE solver for Helmholtz equation
% g -> Dirichlet BC
% w -> Frequency of the Helmholtz equation
% Assembly + Construction of load vector +
% Solution of the linear system
...
```

(8) In order to visualize your numerical solution, you can use:

```
function PlotSolutionHelmoltz(u,N,T)
%% Plot the numerical solution u
%% on the provided triangulation N,T
...
```

Hint: One can use the commands MESH, PATCH *and* COLORBAR.

(9) Test your implementation on the square with Dirichlet boundary condition $g(x, y) = x+y$ and frequency $\omega = 0$. What is the exact solution? Compare the numerical solution with the exact one.

(10) For the chicken in the microwave oven, take for instance $\omega = 2\pi 2.45 \cdot 10^9$ and consider the Dirichlet boundary condition

```
g=inline('100*(x==0.5 & 0.1<=y & 0.2>=y)','x','y');
```

Play a little with your implementation and describe the obtained numerical solutions.

7. FD for two-point BVP (optional)

This computer lab is inspired by chapter 8 in the book Scientific Computing with matlab and Octave by A. Quarteroni and F. Saleri.

Goals. Approximate solutions to BVP by finite difference methods.

Report. Your report must contain: The code for the finite difference with comments and the plots of the exact and numerical solutions given by the finite difference (Task 1).

Task 1. Let $a, b, \alpha, \beta, \gamma, \delta$ be real parameters and a nice function $f\colon (a,b) \to \mathbb{R}$ be given. Consider the BVP

$$\begin{cases} -u''(x) + \delta u'(x) + \gamma u(x) = f(x) \quad \text{for} \quad x \in (a,b) \\ u(a) = \alpha, u(b) = \beta. \end{cases}$$

In order to find a numerical approximation of the unknown solution u, first consider a (large) positive integer N and a discretisation of the interval $[a,b]$ by $a = x_0 < x_1 < \ldots < x_N < x_{N+1} = b$ with $h = x_k - x_{k-1}$ for $k = 1, 2, \ldots, N+1$. Next, approximate the derivatives by (centered) finite differences at the grid points:

$$u'(x_k) \approx \frac{u_{k+1} - u_{k-1}}{2h} \quad \text{and} \quad u''(x_k) \approx \frac{u_{k+1} - 2u_k + u_{k-1}}{h^2}.$$

The finite difference problem then consists of the linear system of equations

$$\begin{cases} -\frac{u_{k+1}-2u_k+u_{k-1}}{h^2} + \delta \frac{u_{k+1}-u_{k-1}}{2h} + \gamma u_k = f(x_k) \quad \text{for} \quad k = 1, \ldots, N \\ u_0 = \alpha, u_{N+1} = \beta \end{cases}$$

for the unknown vector $u_h = (u_1, \ldots, u_N)^T$. Solving this linear system of equations provides the numerical approximations $u_k \approx u(x_k)$ for $k = 1, \ldots, N$.

You can use the following code in order to implement a function for the approximation of BVPs by finite differences.

```
function [x,uh]= bvpFD(a,b,N,delta,gamma,bvpfun,ua,ub)
% BVPFD Solve two-point boundary value problems
% [X,UH]=BVPFD(A,B,N,DELTA,GAMMA,BVPFUN,UA,UB) solves
% the BVP
% -U'' + DELTA * U' + GAMMA * U = BVPFUN
% on the interval (A,B) with boundary conditions
% U (A)=UA and U(B)=UB.
% BVPFUN can be an inline function.
% with the centered finite difference method

h = .. ; % stepsize
z = linspace(a,b,N+2);
e = ones(N,1);
h2 = 0.5* h * delta ;
A = spdiags([-e-h2 2*e+gamma*h^2 -e+h2],-1:1,N,N); % FD matrix
x = z(2:end-1);
f = h^2*feval(bvpfun,x);
f = f'; f(1)=f(1)+ ... ; f(end)=f(end)+ ... ;
uh = ... \ ... ; % solve linear system
uh =[ ... ;uh; ... ]; % add boundary conditions
x = z;
```

```
22 end
```

Take $N = 50$ and use your code to approximate the solution to the BVP

$$\begin{cases} -u''(x) + u'(x) + u(x) = f(x) \quad \text{for} \quad x \in (0,1) \\ u(0) = 0, u(1) = \sin(1) + 1 + epsSTUD, \end{cases}$$

where $f(x) = 2\sin(x) + \cos(x) + x^2 + (2 + epsSTUD)x + epsSTUD - 2$. In this case, one has the exact solution given by $u(x) = \sin(x) + x^2 + epsSTUD \cdot x$. In your report, present the exact and numerical solutions in the same figure.