PROJECT
(October 26, 2022)

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, 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 matlab code. Do not forget that you can verify your codes on simple examples.

This file contain six computational tasks (only five reports are needed to be handed-in). The tasks reflect the content on the lecture. For each computational task, **a report must be sent ten days after the corresponding material has been seen in the lecture** (15 days for the last task). I will try to inform you via Canvas when the corresponding material has been seen in the lecture.

Submission of your lab reports are done via Canvas. If a report is not good enough, students have the possibility to (re)-submit an improved version of their report. **The final date for submitting reports is on the first of February 2023**. It is not possible to submit a report more than two times.

Do not hesitate to come to the supervised lab sessions to work on your reports and get help if needed.

In order to make each report unique, a small real parameter **epsSTUD** is calculated for each student 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.

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

In case of problems, 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.

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

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

1. Introduction to MATLAB

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

Goal. Recall vector/matrix calculation, $2D$ and $3D$ plots for functions with one and two variables, respectively.

Report. You do not need to report on this first computer lab. It is expected that this material is understood before proceeding with the next computational tasks.

Vector/Matrix calculation. Vectors and matrices can be created in various ways in MATLAB.

A simple way to create a vector is using **colon** (:). You can get help from MATLAB for **colon**, or any other MATLAB functions, by typing **doc colon** or **help colon**. Use **doc** and **help** in order to get a better understanding of what these two MATLAB functions are doing.

There are several different ways of creating vectors in MATLAB. The following code creates three 1×4 vectors denoted by a, b and c:

```
>> a=1:4; b=1:0.5:2.5; c=2*b;
```

What would the output be if you instead of semi-colons (;) you use commas (,) between the commands? That is, if you run the following:

```
>> a=1:4, b=1:0.5:2.5, c=2*b
```

Now, run the following commands to see some simple vector/vector operations. Which one of these commands is not a correct MATLAB command?

```
>> a * c
>> a .* c
>> a^2
>> a .^ 2
>> a' .^ 2
>> a' * c
>> a * c'
>> sum(a)
```

Note that **.*** and **.^** are component-wise operators. Pay particular attention to the results of the operations **a*c** and **a.* c** for instance.

Use the MATLAB functions **length** and **size** to display the size of a given vector:

```
>> length(a), size(a)
```

What are the differences between the commands **length** and **size**?

We shall now create some matrices in MATLAB. Run the following piece of code to understand some ways to create matrices in MATLAB (remember that you can use **doc** or **help** if you need help to understand some command):

```
>> [a c], [a,c]
>> [a;c]
>> diag(a)
>> ones(3), ones(3,2)
>> zeros(3), zeros(3,2)
```

```
>> eye(3), eye(4,3)
>> A=[1 2 3; 4 5 6; 7 8 9; 10 11 12]
>> B=[diag(a) zeros(4,1) ; ones(1,5)]
```

The next example illustrates how one can access specific row(s) or column(s) of a matrix. Run the following code:

```
>> A(1,:), A(2,:), A(:,2), A(2:3, :)
>> B(end,:), B(:,end), B(:, 1:3)
```

It is now time to test some vector/matrix manipulations. Run the following code:

```
>> C=repmat(a,3,1), D=repmat(a,3,2), E=repmat(a,1,2)
>> F=reshape(E,2,4), G=reshape(E,4,2)
```

Don't forget to use the help functions to know what the above is doing.

What happens if you run the following code?

```
>> sort(E), sum(E)
>> sum(G), sum(G,1), sum(G,2)
```

And the last very useful command, related to matrices and vectors, is the command **mldivide** or simply \. Can you guess what is does? Feel free to try this command.

To finish this section, answer the following exercise.

Exercise 1. Create the following matrix in one line command:

$$A = \begin{bmatrix} 1 & 8 & 0 & 0 & 0 & 0 \\ -1 & 2 & 8 & 0 & 0 & 0 \\ 0 & -1 & 3 & 8 & 0 & 0 \\ 0 & 0 & -1 & 4 & 8 & 0 \\ 0 & 0 & 0 & -1 & 5 & 8 \\ 0 & 0 & 0 & 0 & -1 & 6 \end{bmatrix}.$$

Plots in 1D, 2D and 3D. First we recall how to plot the graph of a function of one variable $y = f(x),\ x \in [a, b]$. To this end, we need a partition for the domain $[a, b]$. That is, we divide the interval $[a, b]$ into small sub-domains. This can be done, for instance, by considering a mesh step h, and then use **colon** (:), as follows:

```
>> x=a:h:b;
```

Another possibility is to use **linspace** with some positive integer N:

```
>> x=linspace(a,b,N);
```

Note that, if one chooses $h = \frac{b-a}{N-1}$ then the vectors x in the two examples above would be the same.

Let us try this on a concrete example. We would like to plot the function $y = \sin(x),\ x \in [-\pi, \pi]$. To do so, we proceed as follows

```
>> x=-pi:0.2:pi; y=sin(x); plot(x,y)
```

Another possibility would be to use anonymous functions with the @ operator:

```
>> x=-pi:0.2:pi; yy=@sin; plot(x,yy(x))
```

Observe that the above also works for functions of several variables. Feel free to try it on the following function

```
f = @(x,y,z) x.^2 + y.^2 - z.^2
```

Finally, let us illustrate that one can use more options in the command **plot**:

```
>> x=-pi:0.2:pi; y=sin(x); plot(x,y,'bo-')
>> title('2D-plot')
>> xlabel('x'); ylabel('y')
```

If you want to plot more than one function in one **figure**, you could use the following:

```
>> % plot y1(x)=sin^2(x) and y2(x)=sin(x)+x^2, for x in [-5,5]
>> x=-5:0.2:5; y1=sin(x).^2; y2=sin(x)+x.^2;
>> plot(x,y1,'bo-',x,y2,'r*--');
>> title('2D-plot')
>> xlabel('x'); ylabel('y')
>> legend('y1','y2')
```

See **doc plot** for more examples and options for the command **plot**. You can also find more MATLAB commands related to **plot**, at the bottom of the help page, in the section 'See Also'.

Let us now plot a surface defined by a function of two variables $u = f(x, y)$, $x \in [a, b]$, $y \in [c, d]$. To this end, we have to compute the values of the function at some grid points. As is done in $1D$, we first divide the domain $[a, b] \times [c, d]$ into sub-domains. This is can be done using the function **meshgrid** as follows:

```
>> [x,y]=meshgrid(a:h:b,c:k:d);
```

where h and k are some mesh step sizes.

We can now compute the values of the function $f(x, y)$ at the grid by using vector-vector multiplications. Another, but slower, possibility is to use **for**-loops to compute all the values of the function. Let us look at a concrete example in more detail.

We want to compute and store the values of $f(x, y) = \sin(x)\sin(y)$ for $x \in [0, 5]$, $y \in [0, 10]$. To do this, we could use the following

```
>> [x,y]=meshgrid(0:.2:5,0:.1:10);    % We choose h=0.2 and k=0.1
>> f=sin(x).*sin(y);
```

As written above, another possibility is to use **for**-loops:

```
>> x=0:.2:5; y=0:.1:10;    % We choose h=0.2 and k=0.1
>> for i=1:length(x)
>>     for j=1:length(y)
>>         f(j,i)=sin(x(i))*sin(y(j));
>>     end
>> end
```

We can then plot the surface $u = f(x, y)$ using the above grid as follows:

```
>> surf(x,y,f) % One has u=f(x,y)
>> xlabel('x'); ylabel('y')
```

We can also use MATLAB's function **mesh** to plot a surface. Use it and compare your result with the use of **surf**.

M-files. A good practice, when programming, is to use functions or routines, These objects accept input arguments and return output arguments. You can write a MATAB function (or M-file for short) either in the MATLAB base workspace or in a separate file.

The following example illustrates the basic parts of an M-file (create the file myfactorial.m and write the following in it):

```
function f = myfactorial(n)
% myfactorial(n) returns the factorial of n.
f = prod(1:n);
end
```

To compute 5!, one then input in MATLAB the following

```
>> myfactorial(5)
```

2. 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.

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 and Task 2).

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 $i = 0, \ldots, n$. 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 $k = 0, \ldots, n$. 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. 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.

3. Numerical integration in $1d$

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

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

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.

4. FD and FEM for two-point BVP

This computer lab is inspired by the one given by Fardin Saedpanah in 2019 *as well as 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 and finite element methods.

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

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)+ua ; 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.

Task 2. 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. (See also Welty et al, *Fundamentals of Momentum, Heat and Mass transfer* (6th ed.), equation (23-21) in one dimension, with $v = 1/2$ and $R_A = 1$).

Proposition for a solution: We shall construct a numerical approximation u_h in the finite dimensional space of continuous and piecewise linear functions on the partition $\mathcal{T}_h$.

The goal is to derive the resulting linear system of equations $\mathbf{A}\boldsymbol{\xi} = \mathbf{b}$, for the unknowns $\xi_j = u_h(x_j)$, $j = 1, \ldots, m$, where $x_j = jh$, for $j = 1, \ldots, m$, are the nodes of the partition $\mathcal{T}_h$. Observe that $\xi_0 = \xi_{m+1} = 0$ are the given homogeneous Dirichlet BC.

Both the continuous solution and the test functions belong to the Hilbert space

$$V^0 := H_0^1 = \left\{ w \colon [0,\pi] \to \mathbb{R} \, : \int_0^\pi \left(w(x)^2 + w'(x)^2 \right) \mathrm{d}x < \infty, \quad w(0) = w(\pi) = 0 \right\}.$$

To produce a *variational formulation*, one multiplies the BVP (1) with a test function $v \in V^0$ and integrate over $(0, \pi)$. After a partial integration in the first integral, one gets

$$-Du'(\pi)v(\pi) + Du'(0)v(0) + D\int_0^\pi u'(x)v'(x)\,\mathrm{d}x + \frac{1}{2}\int_0^\pi u'(x)v(x)\,\mathrm{d}x = \int_0^\pi v(x)\,\mathrm{d}x.$$

Since $v(0) = v(\pi) = 0$, one obtains the variational formulation

$$\text{(VF)} \qquad \text{Find } u \in V^0 \text{ such that } D\int_0^\pi u'(x)v'(x)\,\mathrm{d}x + \frac{1}{2}\int_0^\pi u'(x)v(x)\,\mathrm{d}x = \int_0^\pi v(x)\,\mathrm{d}x, \quad \forall v \in V^0.$$

To find a numerical approximation of the solution to the variational formulation with a computer, one looks for an approximation in a finite dimensional space of functions. For the cG(1) method, one looks for a piecewise linear function u_h in the space

$$V_h^0 := \{\varphi \in V^0 : \varphi \text{ is continuous and piecewise linear on the partition } \mathcal{T}_h\}.$$

The space V_h^0 can be described using the basis functions φ_j, for $j = 1, \ldots, m$. That is, all *complete hat functions* (i. e. no half hat functions) φ_j that are non-zero on $[x_{j-1}, x_{j+1}]$, for $j = 1, \ldots, m$, see Figure 4. Observe that, since $V_h^0 \subset V^0$, one has that $u_h(0) = u_h(\pi) = 0$. One then does not need the basis functions at the nodes $x_0 = 0$ and $x_{m+1} = \pi$.

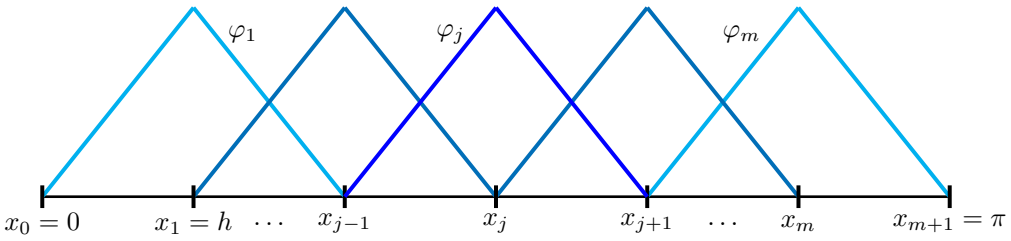

FIGURE 1. Basis functions (hat functions) φ_j on the partition $\mathcal{T}_h$.

The hat functions can be written as

$$\varphi_j(x) = \begin{cases} \frac{1}{h}(x - x_{j-1}), & x \in [x_{j-1}, x_j) \\ \frac{1}{h}(x_{j+1} - x), & x \in [x_j, x_{j+1}) \\ 0, & \text{else.} \end{cases}, \qquad j = 1, \dots, m$$

The finite element formulation (or discrete variational formulation) reads:

(FEM) Find $u_h \in V_h^0$ such that $D \int_0^\pi u_h'(x)\chi'(x)\,\mathrm{d}x + \frac{1}{2}\int_0^\pi u_h'(x)\chi(x)\,\mathrm{d}x = \int_0^\pi \chi(x)\,\mathrm{d}x, \quad \forall \chi \in V_h^0.$

Since $\{\varphi_j\}_{j=1}^m$ is a basis of V_h^0, one can write $u_h(x) = \sum_{j=1}^m \xi_j \varphi_j(x)$, for some (unknown) coefficients ξ_j.

By inserting u_h in equation (FEM), and choosing $\chi = \varphi_i(x)$, $i = 1, \dots, m$ as test functions, one obtains

$$\sum_{j=1}^m \Big(D \int_0^\pi \varphi_j'(x)\varphi_i'(x)\,\mathrm{d}x + \frac{1}{2}\int_0^\pi \varphi_j'(x)\varphi_i(x)\,\mathrm{d}x \Big)\xi_j = \int_0^\pi \varphi_i(x)\,\mathrm{d}x, \quad i = 1, \dots, m,$$

These are m equations (for $i = 1, \dots, m$) with the m unknowns $\{\xi_j\}_{j=1}^m$.

We can write this more compactly in matrix notation $\mathbf{A}\boldsymbol{\xi} = \mathbf{b}$ with $\mathbf{A} = DS + \frac{1}{2}C$, where S is the stiffness matrix, see Chapter 3 of the book for details, given by

$$S = \frac{1}{h}\begin{bmatrix} 2 & -1 & 0 & 0 & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & 0 \\ 0 & -1 & 2 & -1 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & -1 & 2 & -1 \\ 0 & \dots & \dots & \dots & -1 & 2 \end{bmatrix}.$$

To compute the elements of the convection matrix C, we realize, as before, that only c_{ij} with $\|i - j\| \leq 1$ give non-zero contributions (hence one gets a tridiagonal matrix), see Figure 2.

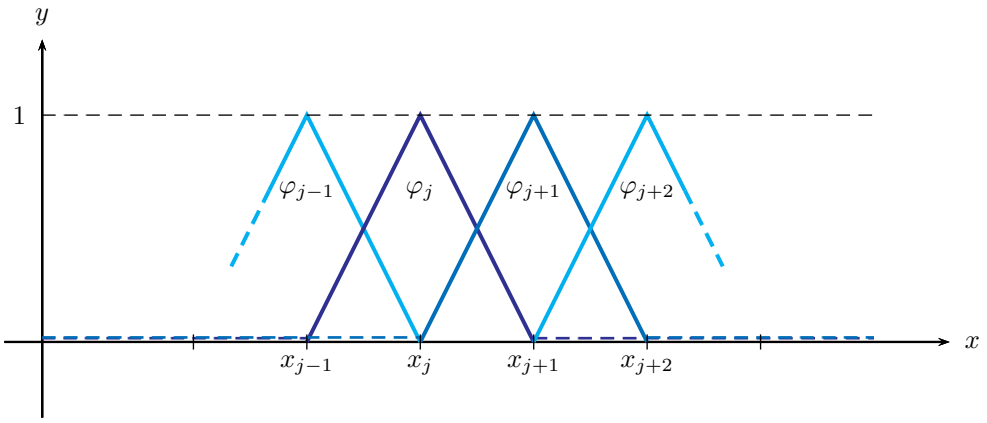

FIGURE 2. φ_j only overlaps with itself, φ_{j-1}, and φ_{j+1}.

The elements of the *skew-symmetric convection matrix* C,

$$c_{ij} = \int_0^\pi \varphi_j'(x)\varphi_i(x)\,\mathrm{d}x,$$

can be computed by evaluating the integrals

$$\begin{cases} c_{ij} = 0, & \text{for} \quad |i-j| > 1 \\ c_{ii} = \int_0^\pi \varphi_i(x)\varphi_i'(x)\,\mathrm{d}x = 0, & \text{for} \quad i = 1,\dots,m \\ c_{i,i+1} = \int_0^\pi \varphi_i(x)\varphi_{i+1}'(x)\,\mathrm{d}x = 1/2, & \text{for} \quad i = 1,\dots,m \\ c_{i+1,i} = \int_0^\pi \varphi_{i+1}(x)\varphi_i'(x)\,\mathrm{d}x = -1/2, & \text{for} \quad i = 1,\dots,m. \end{cases}$$

See Chapter 5.3 in the book for details.

Finally, one computes the load vector $\mathbf{b}$ with elements b_i given by

$$b_i = \int_0^\pi \varphi_i(x)\,\mathrm{d}x = \{\text{area under the basis function } \varphi_i\} = \frac{2h\cdot 1}{2} = h, \quad i = 1,\dots,m.$$

Hence, one has

$$C = \frac{1}{2}\begin{bmatrix} 0 & 1 & 0 & 0 & \dots & 0 \\ -1 & 0 & 1 & 0 & \dots & 0 \\ 0 & -1 & 0 & 1 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & -1 & 0 & 1 \\ 0 & \dots & \dots & \dots & -1 & 0 \end{bmatrix}, \quad \mathbf{b} = h\begin{bmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{bmatrix}.$$

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 offers a template for such Matlab file.

(b) The Péclet number Pe (studied in the course *transportprocesser*) 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: $u_h(0) = u_h(\pi) = 0$!

5. 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 as well as a plot of the exact solution and a plot of the numerical solutions given by explicit Euler's scheme (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 task1a.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 $[0, 20]$. 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
...
```

Task 2. After an accident at the Chemistry and Chemical Engineering building, 10 zombies escape and attack the Biotechnology building, 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?

6. 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. Compare the above PDE with equation (15-17) in Welty et al, *Fundamentals of Momentum, Heat and Mass transfer* (6th ed.).

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 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

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

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 $\chi = \varphi_i,\ i = 0, 1, \ldots, m$, 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*[1]. 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.

a) Implement the numerical scheme (12) for an approximation of solutions to the above heat equation. In order to validate your codes, you can compare the results of your implementation with the plots in Figures F.1

[1] other possibilities of FD approximation are presented in Chapter 6.2 in the book.

and F.7 in the appendix F in Welty et al, *Fundamentals of Momentum, Heat and Mass transfer* (6th ed.). See also equation (17-7) on page 259 and section 17.2 in Welty et al.

Observe that the Neumann BC in the PDE (2) corresponds to solving the problem on a half plate and mirror the solution to the other half. Dirichlet BC would correspond to $h \to \infty$ in Weltys notations. Let $u_0(x) = 1 - x$ and $f(x,t) = 0$ in the above PDE, to turn it into the problem on page 259 in Welty, where $\alpha = 1$, $x_1 = 1$, $T_\infty = 0$, $T_0 = 1$ and $h \to \infty$. Let $m = 0$, $Y = U$, $X = t$ and $n = x$ for comparison with these figures.

Use your code to reproduce the figures F.1 (longtime) and F.7 (short time) for $m = 0$. Observe the logarithmic scale on the y-axis (use the Matlab command `semilogy` to obtain such plots). Test several different discretisation parameters (h and k) and compare the behaviour of the numerical solutions.

Hint 1: Think before you start to implement!

Hint 2: You can download a template of the code and the figure F.1 and F.7 on the Canvas homepage of the course.

b) Let the source term and the initial value be as follows

$$f(x,t) = \frac{10}{\sigma^2} \exp\left(-t - \frac{(x-\bar{x})^2}{\sigma^2}\right),$$
$$u_0(x) = \cos\left(\frac{\pi x}{2}\right).$$

Choose different values of $\bar{x} \in (0,1)$, for instance $\bar{x} = 1/4, 1/2$ and $\bar{x} = 3/4$, and let $\sigma = 0.02$, $T = 2$ and an appropriate choice for the time step size k. Visualise your results (using a `plot` at each timestep, or using `surf`) and assess how reasonable your results are, in terms of the heat conduction, the inital values, the source term, and the BC.

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.