Programming tutorial 4

In this final programming tutorial, we deal with ***isogeometric analysis***. We start by ***setting up geometries*** using ***B-splines and NURBS*** and then solve PDEs on those geometries again using ***splines as ansatz functions***. Basically we can do everything from the lecture also with isogeometric elements as we did it with standard elements so far, i.e. solve elliptic, parabolic and hyperbolic problems static or dynamically, linear or nonlinear, etc. However, for computational simplicity (but on the other hand a deeper insight in what is going on, than FEniCS gave us for example) we will restrict ourselves to simpler examples than for example Navier-Stokes equation and hence in the following will implement ***Poisson's equation***, the ***Heat-equation*** as well as ***(nonlinear) Wave equations***, each with various boundary conditions and parameters.

Exercise 16 (Getting started with the NURBS-toolbox)

Goal:

Installing the NURBS-toolbox and GeoPDEs for MATLAB,
setting pathes and getting ready to work with it.

– – –

GeoPDEs is a MATLAB-toolbox for isogeomtric analysis. It contains a lot of prebuilt functions to deal with the usual tasks of finite elements like assembling matrices or imposing boundary conditions. It also contains the so called NURBS-toolbox, that provides all the necessary functionality to create and manipulate NURBS-geometries that we will use as domains for our PDEs.

a) Download the GeoPDEs-package from `http://rafavzqz.github.io/geopdes/download/` and unpack it. You will find a handbook and some more archive-files in it.

b) Also unpack those archive-files and copy the resulting folders all into one folder on your computer called `GeoPDEs3` or whatever you like.

c) Open MATLAB and navigate into that folder. Then create a new MATLAB-script called `GeoPDEs3_setpath.m`, which contains the following lines:

   -) `my_path = '/home/user/GeoPDEs3';`

   -) `addpath (genpath (fullfile (my_path)));`

   where the variable `my_path` is of course replaced with your individual folder structure on your computer. ***Whenever starting MATLAB, execute this script*** so MATLAB can find all the functions saved in the folder specified by `my_path`, i.e. so far all the prebuilt functions of GeoPDEs.

d) Within the folder `GeoPDEs3`, create a subfolder called `My_programms` or something like that, where all your programms and functions written for this course should be saved, so they can also find each other. The subfolder structure of `My_programms` you can design as you wish.

Exercise 17 (NURBS origami - generating simple geometries)

Goal:

Creating simple NURBS-structures like curves, surfaces and volumes and learn how they can be influences by control points, knots and degrees. Also learn about prebuilt functions of the NURBS-toolbox to set up geometries.

– – –

Linear NURBS-curves

a) Use the command `crv = nrbline(..,..)` to generate a NURBS-curve connecting the points $[0,0]^T$ and $[2,1]^T$ with a straight line. If you need help with the syntax, type `doc nrbline`. What information does the NURBS-structure `crv` you just created give you? Especially examine `crv.coefs` and `crv.knots`.

b) Now create a different NURBS curve, connecting the points $[0,0]^T, [1,3]^T$ and $[2,1]^T$ with straight lines in the given order. Use the command `crv2 = nrbmak(..,..)`. Plot the two curves with the command `nrbctrlplot(..)`.

Hint: Remember that in an open knot-vector the first $p+1$ knots are repeated as are the last $p+1$, where p is the degree of the NURBS.

c) Create a NURBS-curve connecting the points $[0,0]^T$ and $[1,0]^T$ but with a total of 5 (equidistant) control points. You can either use `nrbmak(..,..)` again or (and much simpler) you first create a line connecting the end points via `nrbline(..)` and then apply `nrbkntins(..,..)` to that line to insert three more control points in between.

d) Starting from the curve created in c), move the control points in y-direction such that their coordinates $[c_{i,x}, c_{i,y}]^T$ satisfy $c_{i,y} = c_{i,x}^2 \ \forall i = 1, ..., 5$. Plot the curve and you should get a polygonal chain approximating a parabola.

e) Write a function `nrbrefine (crv, nctrlptsins)` that takes a linear NURBS-curve created by `nrbline(..,..)`, i.e. the pure connection between two points and inserts a number of control points specified by `nctrlptsins` with uniform knot distance. Test your function by plotting its output.

Hint: The function may use the prebuilt function `nrbkntins(..,..)`, you just have to tell, on which positions knots should be inserted in the knot-vector.

Higher order NURBS-curves

f) In order to create a NURBS-curve of second order, one can again use the `nrbmak(..,..)` command. Use the same control points as in exercise b) but adapt the knot-vector such that you end up with a quadratic curve. Plot the curve!

Hint: Remember the formula $K = n + p + 1$, where K is the number of knots in the knot-vector, n the number of control points and p the degree of the NURBS.

g) An other possibility to create higher order NURBS is to start with a linear one (created by `nrbline(..,..)`) and then to elevate its degree. This can be done by the command `nrbdegelev(..,..)`. Start with a straight line connecting start and end point, elevate its degree by one and adjust the middle control point to recreate the curve from f).

h) Of course one can also mix degree-elevation and knot-insertion to create higher order curves with more control points. Try this and insert two additional control points into the quadratic curve from g). Plot the result!

i) Move the second (from left) control points of the curve from h) and observe the behaviour of the curve. Does the right end change at all? Why not? Instead of `nrbctrlplot(..)` plot the modified curve with `nrbkntplot(..)`. What do the intersection points tell you?

j) Eumel wants to create a smooth curve by a quadratic NURBS-curve with 5 control points. Why does the following code still give wrong results, i.e. why is there still a kink? Can you correct it?

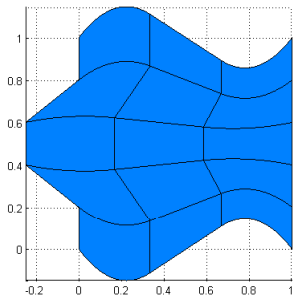
```
eumel = nrbline ([0,0],[1,0]);
eumel = nrbkntins (eumel, [1/2]);
eumel = nrbdegelev (eumel, 1);
eumel.coefs(2,:)  = eumel.coefs(1,:).^4;
figure(5)
nrbctrlplot(eumel);
```

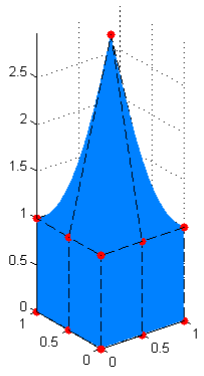
NURBS-surfaces and volumes

k) Like curves, also NURBS-surfaces can be created with the `nrbmak(..,..)` command. Use this to create the (linear) unit square with control points at its four vertices. Again plot the structure via `nrbctrlplot(..)`.

l) Since the method from k) is probably a bit tedious, one can also use the so called *Coons-patch* to create surfaces by specifying its four boundaries as curves. Try this again for the unit square by using `nrbcoons(..,..,..,..)`.

m) Use the Coons-patch again, but this time you modify the four boundary curves by i.e. elevating their degree, adding control points or shifting the existing control points or even all together. You just have to take care that opposing edges of the "square" have the same degree and number of control points! Also be aware of self-intersection! Again visualize the supports of ansatzfunctions by `nrbkntplot(..)`. (Picture below shows an example).



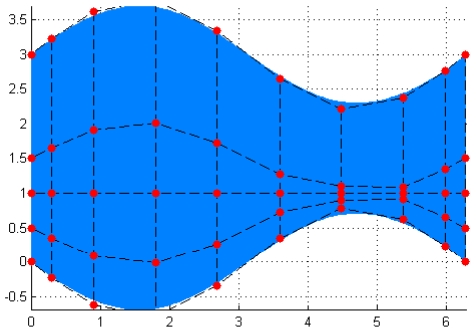


n) For a 3D-NURBS-volume there is unfortunately no Coons-patch available, hence we again have to use `nrbmak(..,..)` just with three components instead of two. Create the three dimensional unit cube. Plot it again with `nrbctrlplot(..)`. Then insert an additional control-point in x and y direction at $\xi = \eta = 1/2$ and move one of them to create a "dwarf-hat" for the cube (see picture above).

Master-Task

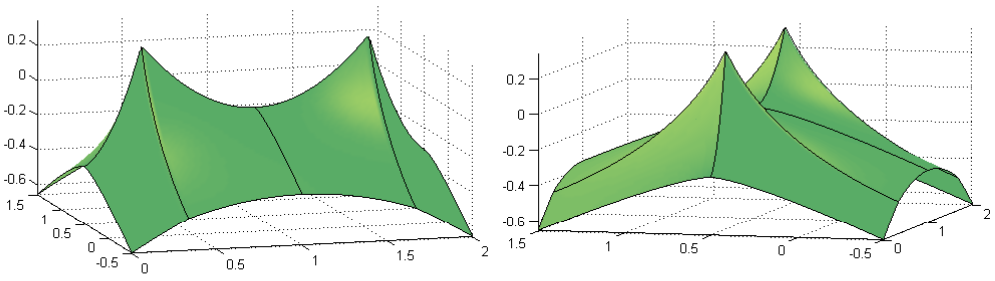
Now, that you know most of the important commands to create NURBS-geometries, lets try to apply this and recreate the following a bit more complex geometries.

M1) Create a smooth, plane NURBS-surface, that is cubic in x-direction and quadratic in y-direction. The number of control points should be 10 in x and 5 in y direction. The control points of the upper y-boundary should lie on the graph line $3+4/5\sin([0,2\pi])$, the ones of the lower y-boundary on the graph line of $-4/5\sin([0,2\pi])$ while the control points of the x-boundaries are on the $x=0$, resp. $x=2\pi$ axis, but concentrated in the bottom half as the following picture shows, i.e. on the boundaries having y-coordinate `[0,0.5,1,1.5,3]`. Be careful with mesh-tangling, you possibly have to manually correct something!



Note: It is *not* said, that this mesh is a useful one in context of a finite element discretization, its purpose is just to see how one can increase the mesh density in some regions of the domain and that one has to take care with the Coons-patch!

M2) Recreate the following (quadratic in x and y) NURBS-model of the Munich Olympic stadium's roof. Hereby it is not necessary to match every single control point since they are not even visible in the picture. The goal is to correctly represent the two kinks where the "tent"-like structure is (in reality) supported by metal beams and hence to find the correct number of control points and especially the knot vectors in x and y.



Exercise 18 (A homogeneous Dirichlet problem)

Goal:

Solving Poisson's equation $-\Delta u = f$ on the unit square with homogeneous Dirichlet boundary conditions using isogeometric finite elements.

Getting used to the GeoPDEs-structures
It is recommended to create an empty MATLAB-file where you can write all the following commands one after the other and in between you can always run the script and check what the different structures contain.

a) The first step is to specify the geometry Ω over which we want to solve the PDE. Here it will be the unit square. Use the commands of the NURBS-toolbox to create the unit square with linear splines and (for now) only 4 control points at the vertices of the square. In this setting the domain is equivalent to one single (classical) element with bilinear ansatz-functions. This will make it simpler to understand and compare the following structures to classical finite elements. Later we will of course refine the mesh to really calculate a PDE solution.

b) Beside the domain Ω the only other data needed to specify the problem is the right hand side function f. This should be given by a function handle. Make sure that your function is vectorized, i.e. that it can be applied to vectors or matrices componentwise. This is especially tricky (not difficult but usually one forgets this) for the most simple choice of $f(x, y) \equiv 1$ since also here the output should have the same dimension as the input!

From here on it is the goal to set up the stiffness-matrix K and the right hand side vector b to solve $K \cdot u = b$ for the solution vector u. To do so, GeoPDEs uses a sequence of commands that subsequently create all the necessary tools like quadrature nodes, boundary specifications or Jacobians etc.

c) At first, the NURBS-surface describing the geometry has to be enhanced by some more information. We will later need for example the Jacobian of the NURBS mapping or some information on how the boundaries are numbered. In GeoPDEs this information is created by loading the NURBS-surface with the command `geo_load (..)`. We will

call the resulting structure `geometry`. Play around with `geometry`. See what its different entries are and what values they take.

d) The next thing to specify is the numerical quadrature rule we want to use. Here we restrict ourselves to Gaussian quadrature with a given number of nodes in each parametric direction. We implement this by specifying a variable `rule` with the function `msh_gauss_nodes(..)`, then pass `rule` to the function `msh_set_quad_nodes` to set up quadrature nodes (`qn`) and quadrature weights (`qw`). Again check the values, dimensions etc. for the created variables.

e) The following two commands are probably a bit more intransparent. They generate - just by rearranging and evaluating the data we have already created - special data structures suitable to compute the stiffness-matrix with.

```
msh   = msh_cartesian (geometry.nurbs.knots, qn, qw, geometry);
space = sp_nurbs (geometry.nurbs, msh);
```

Of course it is worth again to check the different field entries of `msh` and `space`. You can get an overview over them by calling `fieldnames(..)`. However, there are more than just a few, so it should also be enough to just look at those entries that we will need at all in the following. At the respective positions we will always mention, where to find these variables.

f) Now all necessary data structures are available to create the right hand side vector `rhs` and the stiffness matrix `K`. The later one is created with the command `op_gradu_gradv_tp(..,..,..,..)`, where we use `@(x,y) ones(size(x))` as the last argument, since $\Delta u = \nabla \cdot (\nabla u)$. For the general case $\nabla \cdot (a(x,y) \cdot \nabla u)$ it is possible to specify the diffusion coefficient $a(x,y)$ here.

For the right hand side vector we use `op_f_v_tp (..,.., @(x,y) f(x,y))` where the function handle `f(x,y)` is the one specified in exercise b).

Control: For Ω being just one bilinear element and $f(x,y) \equiv 1$, K and b are given by:

$$K = \begin{pmatrix} 0.6667 & -0.1667 & -0.1667 & -0.3333 \\ -0.1667 & 0.6667 & -0.3333 & -0.1667 \\ -0.1667 & -0.3333 & 0.6667 & -0.1667 \\ -0.3333 & -0.1667 & -0.1667 & 0.6667 \end{pmatrix} \qquad b = \begin{pmatrix} 0.2500 \\ 0.2500 \\ 0.2500 \\ 0.2500 \end{pmatrix}$$

g) The last task to do before we can solve the linear system is to bring in the boundary conditions. Since we have homogeneous Dirichlet conditions on the whole boundary, we can simply set all degrees of freedom contributing to the boundary to zero. Therefore we somehow have to know, which degrees of freedom resp. which ansatzfunctions *do* contribute to the boundary. In the simple case of just one element it is clear that all DOFs are Dirichlet DOFs, however in general we have to find them first.

The `space` structure from exercise e) has the fields `space.boundary(i)`, i=1,...,4 which themselves have the subfield `space.boundary(i).dofs` giving the numbers of DOFs contributing to boundary i. By looping over all four boundaries an collecting those DOF

numbers one gets the array `drchlt_dofs` containing all numbers of Dirichlet DOFs. Its complement is called `int_dofs` collecting the numbers of interior DOFs.

h) Just as in classical finite elements one proceeds by setting the `drchlt_dofs` entries of the solution vector `u` to zero, brings them to the right hand side of the system (for homogeneous conditions this can be skipped, just don't forget it for non-homogeneous ones) and solves only for the remaining interior ones $u = K^{-1}b$.

i) In order to plot the solution you may copy the following lines assuming that you indeed used the same variable names as proposed here.

```
figure(1);
samp = {linspace(0, 1, 20), linspace(0, 1, 20)};
[u_eval, F_geo] = sp_eval (u, space, geometry, samp);
[X, Y] = deal (squeeze(F_geo(1,:,:)), squeeze(F_geo(2,:,:)));
surf (X, Y, u_eval);

shading interp
colormap jet
view([-45,45]);
axis equal;
```

Annotation: Don't wonder, if you just get the zero function as a plot. If we only use one element all four nodes are boundary nodes and due to the homogeneous Dirichlet condition this results in a zero solution.

j) Refine the geometry (Exercise part a) by inserting at least 3 more nodes per parametric direction. Again run the whole script now in the refined geometry for the following right hand sides:

- $f(x, y) \equiv 10$
- $f(x, y) = 20 \cdot \sin(2\pi x) \cdot \sin(2\pi y)$

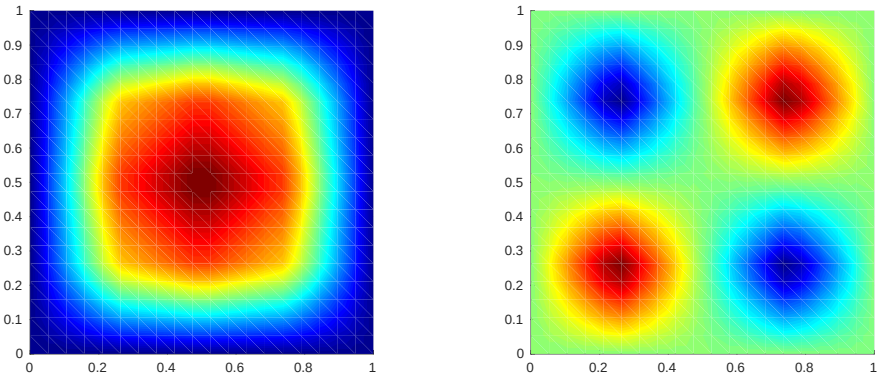


Fig. 1: left: solution for constant f, **right:** solution for the other f

~~**Master-Task**~~ For now only necessary if you

M1) Again for $\Omega = [0,1]^2$ solve the following Helmholtz-type equation: $-\Delta u + a(x,y)u = 1$ with homogeneous Dirichlet conditions and

a) $a(x,y) \equiv -100$

b) $a(x,y) = -100 \cdot x \cdot y$

with quadratic elements and at least 400 degrees of freedom in total.

Hint: Similar to the stiffness matrix, the command `op_u_v_tp (..,..,..,..)` assembles the mass matrix. The factor $a(x,y)$ you can include like a reaction coefficient (compare to exercise f). Also do not forget to elevate the NURBS degree of the domain at the correct position!

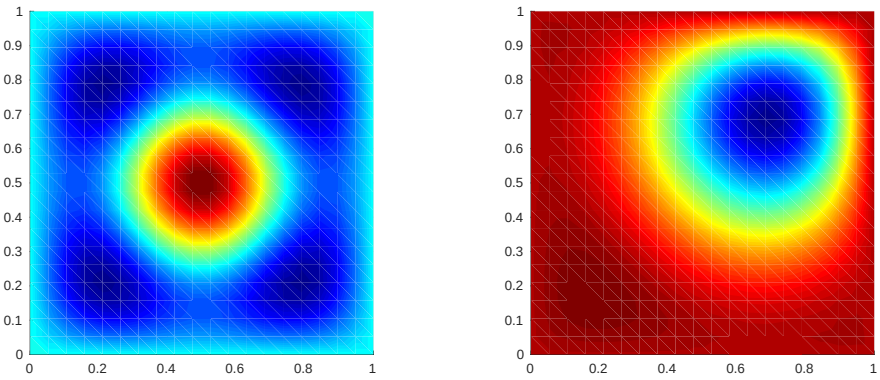


Fig. 2: left: solution for a), **right:** solution for b)

Exercise 19 (Adding Neumann data)

Goal:

Again solving Poisson's equation $-\Delta u = f$ but this time a Neumann boundary is included. How can this be treated in the isogeometric framework?

Just a few changes

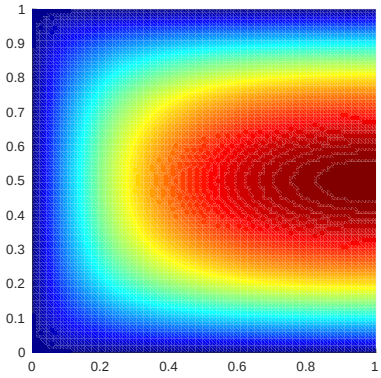
a) Copy your code from Exercise 3, since we only have to add a few things. In order to get nice pictures you should probably also increase the resolution to incorporate at least 400 DOFs in total.

b) Lets assume the Neumann-boundary is the right edge of the unit square, i.e. the problem we want to solve reads:

$$\begin{aligned} -\Delta u &= f && \text{in } \Omega := (0,1)^2 \\ \partial_n u &= g && \text{at } \Gamma_N := \{(x,y) \in [0,1]^2 \mid x = 1\} \\ u &= 0 && \text{at } \Gamma_D := \partial\Omega \backslash \Gamma_N \end{aligned}$$

Find out, which of the boundaries `space.boundary(i)`, $i = 1, ..., 4$ corresponds to the (right) Neumann-boundary and exclude its DOFs from the `drchlt_dofs` array in the code.

c) Run the code! Explain why this already works even though the right boundary has (currently) no boundary condition prescribed. Which boundary condition is (implicitly) given anyway?



d) Now we also want to incorporate non-homogeneous Neumann-data. Theory tells us that we can do this by adding the following vector to the right hand side of the discrete system:

$$\underline{g} := \left(\int_{\Gamma_N} g \cdot N_i \, \mathrm{d}S \right)_{i=1}^{\mathrm{ndof}}$$

where $g = g(x, y)$ is the Neumann-data function and N_i are the NURBS ansatz functions from the finite element space. Of course this integral will vanish for most of the N_is, since ansatz functions contributing to the left boundary of the domain for example or somewhere in the interior of Ω are zero at the right (Neumann) boundary over which we integrate to form $\underline{g}$.

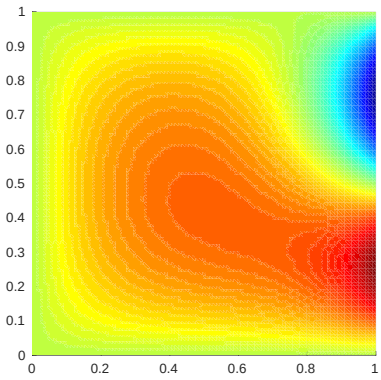
In GeoPDEs we get the Neumann-vector very similar to how we got the right hand side vector b

$$\underline{b} := \left(\int_{\Omega} f \cdot N_i \, \mathrm{d}\Omega \right)_{i=1}^{\mathrm{ndof}}$$

we just have to replace f by a function handle g stating the Neumann-data and instead of Ω we want to integrate over Γ_N. To do so, you can again use the command `op_f_v_tp(..,..,..)` but as arguments you do not pass the whole `space` and `msh` but rather the respective boundary sub-fields `space.boundary(i)` and `msh.boundary(i)`, where i has to be replaced by the index of the right (Neumann) boundary.

Attention: GeoPDEs checks which ansatzfunctions N_i do contribute to the boundary over which we integrate and gives back the resulting entry of $\underline{g}$ only for those degrees of freedom. You have to consider this and write these results at the right positions of the global Neumann-vector to avoid a dimension mismatch!

e) As in the last exercise, bring the Dirichlet DOFs to the right hand side and solve $\underline{u} = K^{-1} \cdot (\underline{b} + \underline{g})$ for the internal DOFs. Use $f \equiv 1$ and $g(x, y) = \sin(2\pi y) - \frac{x}{3}$ to compare with the following picture.



~~**Master-Task**~~ Only if you are interest

M1) Recreate (i.e. copy) the example from Exercise 4) but this time not only the right boundary carries Neumann-data but:

a) right and bottom boundary

b) All boundaries except the point $(0, 0)$, where a homogeneous Dirichlet datum is given, i.e. $u(0, 0) = 0$. (it is not said, that it makes sense to prescribe a Dirichlet datum only on one single point of the boundary, however at this moment, it is just a convenient exercise to deal with the boundary conditions.)

Use the same functions for the right hand side and the Neumann datum as in exercise 4,e).

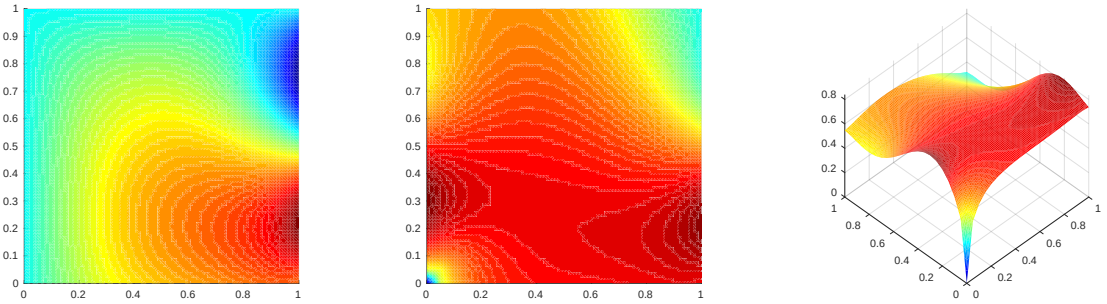


Fig. 3: left: solution for a), **middle and right:** solution for b)

Important for Domain decomposition

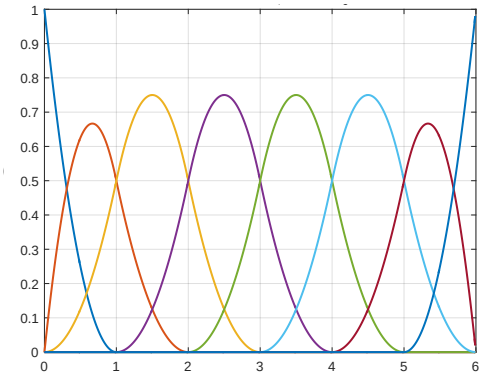
Exercise 20 (Non-homogeneous Dirichlet data in the IGA setting)

Goal:

Solve Poisson's equation one last time, but this time with inhomogeneous Dirichlet-conditions. A first "real" difference to classical finite elements.

Why are inhomogeneous Dirichlet conditions a "problem" in the IGA setting?

The problem is that the spline ansatz functions used in IGA are not interpolatory. This means that compared to classical FEM, where for an arbitrary degree of freedom $\underline{x}_i$ there was exactly one ansatzfunction N_i with $N_i(\underline{x}_i) = 1$ and $N_j(\underline{x}_i) = 0$ for all $j \neq i$, this does not hold for B-spline or NURBS basis functions as one can see in the following picture of quadratic ansatzfunctions.



Assuming that the quadratic ansatzfunction from the picture constitute a Dirichlet boundary of a two-dimensional domain with Dirichlet values not being constant but for example $u_D(x) = x^2$. In classical FEM one would prescribe the values for the Dirichlet DOFs as follows:

$$x_i^2 = u_D(x_i) = u(x_i) = \sum_{j=1}^{\text{ndof}} \underbrace{N_j(x_i)}_{\delta_{ij}} \underline{u}_j = \underline{u}_i$$

where x_i is the location of an arbitrary Dirichlet DOF. But since the interpolatory property gets lost in the IGA setting, more than just one ansatzfunction contribute to every single point x_i on the boundary. Imagine again in the upper picture you have a degree of freedom at $x_i = 2.5$. Somehow you would have to combine the purple, yellow and green ansatzfunctions (those which are not zero at that point) such that their sum matches the prescribed value. In addition you can not just do this for your degree of freedom at $x_i = 2.5$ independently from the other DOFs, since changing the coefficient of the lets say green ansatz function also affects for example the DOF at $x_j = 4.5$, where the ansatzfunction is also not zero! Somehow one has to find a global combination/solution for all ansatzfunctions contributing to the boundary, such that their sum satisfies the prescribed data in the best possible way.

The L^2-projection

On some domain D (it can be the boundary of a 2D-domain but also its interior) we want to find the best possible NURBS-basis representation of a prescribed function $p \in L^2(D)$ in the

sense of the L^2 norm. This means we want to minimize the functional $F(u_h) = \frac{1}{2}\,||u_h - p||^2_{L^2(D)}$, w.r.t. the finite element (in our case NURBS) space $V_h(D) \subset L^2(D)$. The optimality condition is that all directional derivatives of F vanish, i.e. $\langle u_h - p, v_h\rangle_{L^2(D)} = 0 \quad \forall v_h \in V_h$. Discretizing $u_h = \sum_{i=1}^{\text{ndof}} N_i u_i$ and $v_h = \sum_{j=1}^{\text{ndof}} N_j v_j$ finally leads to the following system to be solved for u_i.

$$\underbrace{\left(\int_D N_j N_i \,\mathrm{d}D\right)_{j,i=1}^{\text{ndof}}}_{=:M} \cdot (\underline{u}_i)_{i=1}^{\text{ndof}} = \left(\int_D N_j \cdot p \,\mathrm{d}D\right)_{j=1}^{\text{ndof}}$$

The right hand side one recognizes as the usual right hand side that also appears when solving for example $-\Delta u = f$ just with p in place of f and (maybe) a different domain $D \neq \Omega$. The matrix M is also referred to as *mass-matrix* of the domain D.

~~Implementing the L^2-projection - full domain~~ Only if interested because there is a function which does

Stepping one step back, we forget about PDEs for the moment and just want to implement the L^2 projection for arbitrary domains / spaces / functions etc.

a) Again set up the unit square as a NURBS domain but this time on a very coarse mesh (on purpose), i.e. linear NURBS in both directions with knots at `[0 0 0.5 1 1]` each.

b) Also set up all the other necessary data structures we worked so far with like `msh`, `space` etc.

c) Implement a function handle `p` for the function $p(x, y) = (x - 0.5)^2 + (y - 0.5)^2$ that we want to project.

d) Write a function `L2projection(p,space,msh)` that returns the coefficient vector $\underline{u}$ of the L^2-projection of p into the NURBS-space `space`.

Hint: Remember the Master-task from exercise 3), you already implemented a mass-matrix there. The full function `L2projection` can be implemented in three lines, just think about what you need (matrix an right hand side) and of course the solution of the linear system.

e) Test your projection function on the given function p and the given (coarse) mesh. Plot the result with the usual plotting routines (see last exercises).

f) Compute the L^2-error $\|u_h - p\|_{L^2(\Omega)}$ of the coarse L^2-projection. Use the command `sp_l2_error(..,..,..,..)` for the error-computation.

Attention: GeoPDEs computes the error with a numerical quadrature rule that uses the quadrature nodes specified by the variables `rule, qn` and `qw`. So far we have always used 2 Gauss points in each parametric direction per element to integrate the splines. Since we never went higher than order 2 this was justified as 2 Gauss-points integrate exactly up to degree 3. Now for the L^2 error-computation involving a quadratic we have to integrate polynomials up to fourth order. Hence to avoid additional errors from the mere computation of the error, we increase the number of Gauss-points to 3 per parametric dimension, so we integrate again exactly up to 5th order. In the code, just replace the line:

```
rule = msh_gauss_nodes (geometry.nurbs.order);
```

by
`rule = msh_gauss_nodes ([3 3]);`

g) Now use quadratic NURBS in both directions again just on a very coarse mesh with knot-vectors `[0 0 0 1 1 1]` in both directions and again project p into that space and compute the L^2-error. What do you expect the error to be?

Annotation: You can play around with the L^2-projection. For example you can return to a linear ansatz-space but increase the number of DOFs, which will of course also increase the quality of the projection, even though a quadratic is still a quadratic and will never be exactly represented in a space of linear functions. You can of course also change the function to project!

Applying the L^2-projection to the Dirichlet-data - boundary space

Use function "sp_drchlt_l2_proj.m" instead of loops and function L2projection

Back to the actual PDE business. You can copy your code from exercise 3, we will now enhance it to also incorporate non-homogeneous Dirichlet-conditions by first projecting the Dirichlet-data into the boundary-NURBS space and then (as usual) just solve for internal DOFs.

a) Define a function handle `d` for the Dirichlet-data on the boundaries, i.e. for the following problem

$$\begin{aligned} -\Delta u &= f \qquad \text{in } \Omega := (0,1)^2 \\ u &= d \qquad \text{at } \partial\Omega \end{aligned}$$

b) For the sake of simplicity we treat each of the four boundaries individually, i.e. at first we project the Dirichlet-datum into the NURBS space of boundary 1 to determine the values of all DOFs contributing to boundary 1. Then we project to the second boundary and set these DOFs and so on. All you have to do is to replace the line `u(drchlt_dofs) = 0;` by a loop over the four boundaries, each time setting the respective DOFs to the values the function `L2projection(..,..,..)` gives you, if you call it not for the whole space and mesh `space` and `msh` but for the individual boundary structures `space.boundary(i)` and `msh.boundary.(i)`.

Annotation: The approach to L^2-project each boundary individually is strictly speaking not correct, since the boundaries meet at the vertices of the unit square and hence the coefficient for a vertex ansatz-function from one edge could differ from the coefficient from the other edge and one would have to unify the four boundaries first, then project for the whole boundary and identify backwards. However, this would be too much of an index-war distracting from the actual goal at this moment.

c) If you have not yet done it in the exercises before (where it was actually not necessary due to homogeneous Dirichlet conditions) you now *have to* bring the Dirichlet DOFs to the right hand side of the system and then only solve for the internal ones.

d) Use the code to compute the numerical solution u_h of:

$$\begin{aligned} -\Delta u &= -(2+y^2)\exp(x) \qquad \text{in } \Omega := (0,1)^2 \\ u &= y^2 \exp(x) \qquad \text{at } \partial\Omega \end{aligned}$$

The analytical solution is given by $u_{\text{ex}} = y^2 \exp(x)$. Also compute the L^2-error $\|u_h - u_{\text{ex}}\|_{L^2(\Omega)}$ and the H^1-error $\|u_h - u_{\text{ex}}\|_{H^1(\Omega)}$.

Hint: For the L^2 error remember the command `sp_l2_error(..,..,..)`, for the H^1 error use `sp_h1_error(..,..,..,..)` (you have to compute the gradient of the exact solution by hand and pass it as a vectorial function handle to the fourth argument in the following way: If $u(x, y) = x^2 + y^2$ we would have $\nabla u = (2x, 2y)^T$. The function handle passed to the H^1 error function would then read:
`@(x, y) cat (1, reshape (2*x, [1, size(x)]), reshape (2*y, [1, size(x)]))`.

Master-Task

M1) Conduct a convergence study for the numerical solution of the following problem:

$$\begin{aligned} -\Delta u + 2u &= 4\left(1 - x^2\right) y \exp\left(-x^2\right) && \text{in } \Omega := (0,1)^2 \\ u &= y \exp\left(-x^2\right) && \text{at } \partial\Omega \end{aligned}$$

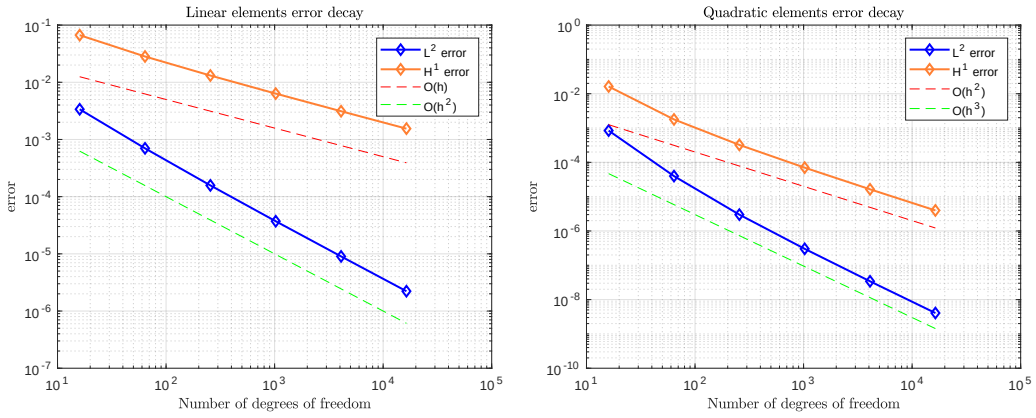
i.e. solve the equation on several - each finer than the last - grids and plot the L^2, resp. H^1 error w.r.t. the total number of degrees of freedom. Do this once for linear and once for quadratic elements. Which error decay rates do you observe for each of them? The exact solution is again given by the boundary data.

To do this as efficient as possible, you can proceed as follows:

a) Write a function `h_refine(..,..)` that takes a given NURBS domain - in the simplest case the coarsest one with only 4 nodes - and refines it to a specified number of degrees of freedom. Be careful that your function does also work for NURBS of arbitrary (or at least also for quadratic) order, not just linear.

b) Write an other function `solve_equation(..)` that takes a (refined) NURBS domain and on this domain solves the given equation, i.e. it conducts all the steps you already implemented in Exercise 5 or before that are necessary. The function should give back the L^2 and H^1 error of the current domain as well as the total number of degrees of freedom.

c) In some script file, you collect all the returned arguments of b) an plot them in a logarithmic scale together with the reference lines $\mathcal{O}(h^n)$ for some suitable n. The result should look as in the following pictures.



~~Exercise 21 (Save the cold beer - a dynamical heat problem)~~

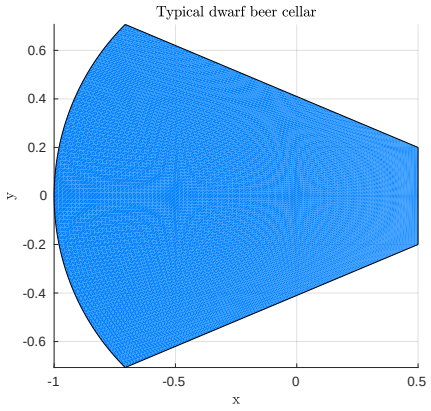
Goal:

Solve the parabolic heat equation by applying an isogeomtric finite element discretization in space and a suitable ODE integration scheme in time.

After an important victory against a hostile tribe, the dwarfs of mount Erebor have a huge party. One of them - Eumel - was responsible for serving beer. Unfortunately he had a few beers too much himself and forgot to close the door of the beer cellar at the end of the party causing warm air to stream into the cellar and eventually warming up all the dwarfs beer supply. In the middle of the night, at $t = 100$ he wakes up and remembers his fault. Can he still save the beer from warming up by closing the door again? Use the following parameters / conditions to simulate the situation:

- The room temperature at $t = 0$ was $u = 5$.
- The bottom wall of the cellar is permanently cooled by a river flowing behind that wall to $u = 5$.
- The outside temperature, which also applies to the (right) boundary of the cellar representing the open door is $u = 45$ (the dwarf city is close to a lava flow, hence the outside air is quite hot).
- The upper and left wall of the cellar are perfect isolators, i.e. there is no heat flux $\kappa \partial u / \partial \underline{n}$ across them.
- The heat conductivity of air is given by $\kappa = 0.025$.
- The beer may never reach a temperature of more than $u = 11.5$ otherwise it degenerates and can not be drunk again. This is what he wants to avoid once he wakes up.

The cellar looks as follows:



The left boundary is part of the unit circle ranging from $\theta = \frac{3}{4}\pi$ to $\theta = \frac{5}{4}\pi$, the other boundaries are straight edges. The door is on the right hand side with vertices at $(0.5, \pm 0.2)^T$. The beer is supposed to be stored directly on the left (curved) wall, hence the excise reduces to determine

whether the temperature will exceed its limit at any point of the left boundary.

Setting up the geometry

a) To set up the curved boundary you may use the command `nrbcirc(..,..,..,..)`. In case the left boundary curve is oriented in the wrong way, use `nrbreverse(..)` to fix it. Now also place the remaining edges each with `nrbline(..,..)`.

b) You will realize that the left boundary is quadratic, hence also elevate the degree of the other three boundaries to 2 and form the NURBS surface via the Coons-patch.

c) Use your function `h_refine(..,..,..)` from exercise 5 to refine the domain to at least 1600 DOFs in total.

Spatial discretization

Recall the heat-equation without heat source:

$$\frac{\partial u}{\partial t} - \kappa \Delta u = 0$$

The weak (in space) form reads:

$$\int_\Omega \frac{\partial u}{\partial t} \cdot v \, \mathrm{d}\Omega + \kappa \int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}\Omega = \int_{\Gamma_N} \kappa \frac{\partial u}{\partial \underline{n}} \cdot v \, \mathrm{d}S \quad \forall v$$

where Γ_N consists of the left and upper boundary where homogeneous Neumann-conditions ($\kappa \partial u / \partial \underline{n} = 0$) are prescribed and hence even the Neumann-boundary integral vanishes. The other two boundaries are Dirichlet-boundaries.

After discretization in space with isogeometric finite elements we get the following semi-discrete formulation (semi-discrete, since it is still continuous in time).

$$\left(\int_\Omega N_j \cdot N_i \, \mathrm{d}\Omega \right)_{i,j=1}^{\mathrm{ndof}} \cdot \left(\underline{\dot{u}}_j \right)_{j=1}^{\mathrm{ndof}} + \kappa \left(\int_\Omega \nabla N_j \cdot \nabla N_i \, \mathrm{d}\Omega \right)_{i,j=1}^{\mathrm{ndof}} \cdot \left(\underline{u}_j \right)_{j=1}^{\mathrm{ndof}} = \underline{0}$$

which is a system of ndof-many ODEs, that can also be written in the following form using mass- and stiffness matrices:

$$M \cdot \underline{\dot{u}} + \kappa K \underline{u} = \underline{0}$$

d) Set up the necessary quantities as usual for the spatial discretization. Especially mass- and stiffness matrices M and K are needed.

Time discretization

For the time-discretization we use the implicit Euler-Scheme with steplength Δt. With the notation $\underline{u}^k := \underline{u}(k \Delta t)$ it reads:

$$M \cdot \frac{\underline{u}^{k+1} - \underline{u}^k}{\Delta t} + \kappa K \cdot \underline{u}^{k+1} = \underline{0}$$

Including the Dirichlet-data into the formulation leads to:

$$M_{\text{int,int}} \cdot \left(\underline{u}_{\text{int}}^{k+1} - \underline{u}_{\text{int}}^{k}\right) + \Delta t \kappa K_{\text{int,int}} \cdot \underline{u}_{\text{int}}^{k+1} = -M_{\text{int,drchlt}} \cdot \underbrace{\left(\underline{u}_{\text{drchlt}}^{k+1} - \underline{u}_{\text{drchlt}}^{k}\right)}_{=0} - \Delta t \kappa K_{\text{int,drchlt}} \cdot \underline{u}_{\text{drchlt}}^{k+1}$$

$$\Leftrightarrow \underline{u}_{\text{int}}^{k+1} = [M + \Delta t \kappa K]_{\text{int,int}}^{-1} \cdot \left[M_{\text{int,int}} \cdot \underline{u}_{\text{int}}^{k} - \Delta t \kappa K_{\text{int,drchlt}} \cdot \underline{u}_{\text{drchlt}}^{k+1}\right]$$

where the indices int and drchlt stand for internal or Dirichlet DOFs.

e) Initialize the solution vector $\underline{u}^0$ according to the initial datum $u(0, x, y) \equiv 5$ in Ω.

Hint: You could do this by L^2-projecting the initial datum $u(0, x, y) \equiv 5$ into the NURBS space of the full domain, however you can also exploit the following "trick". Thanks to the partition of unity property of the NURBS-basis functions such a spatial constant function can simply be expressed by the coefficient vector $(5, 5, ..., 5)^T$.

f) Also initialize the boundary conditions. Again: You do not have to use the boundary L^2-projection either, since also the boundary conditions are spatially constant and hence can be directly set.

g) Implement a function `step_in_time` which takes the solution vector $\underline{u}^k$ of timestep k and computes the solution vector $\underline{u}^{k+1}$ of the new timestep. For the internal DOFs the update rule from above can be used. For the Dirichlet DOFs you could of course proceed as before but you can also exploit a "trick" such that not a single L^2 projection will be necessary!

Trick: Since the Dirichlet data are also constant in time, you can always copy them from the solution vector of the last timestep, i.e. no update is needed at all.

Hint: To speed up the computation, see if you can precompute some parts, especially concerning the matrix inversion e.g. by a L-U-decomposition!

h) Put e),f) and g) together and implement a time-step loop calling `step_in_time(..)` successively in order to compute the solution of the semi-discrete system. Also plot the temperature-field in every time-step. Use at least 100 timesteps.

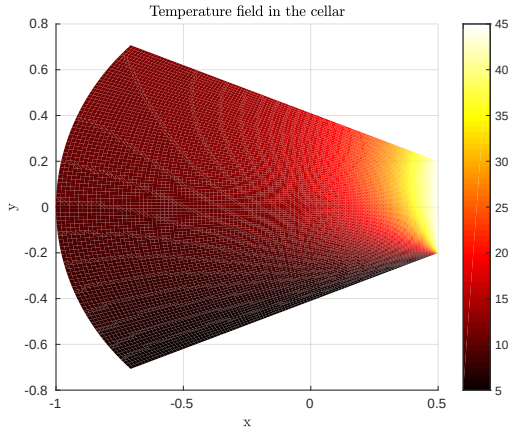


Fig. 4: Temperature distribution at final time $t = 100$.

Post-processing

i) Plot the temperature distribution at the left wall to determine whether it exceeds the threshold of 11.5 degrees at the time Eumel wakes up.

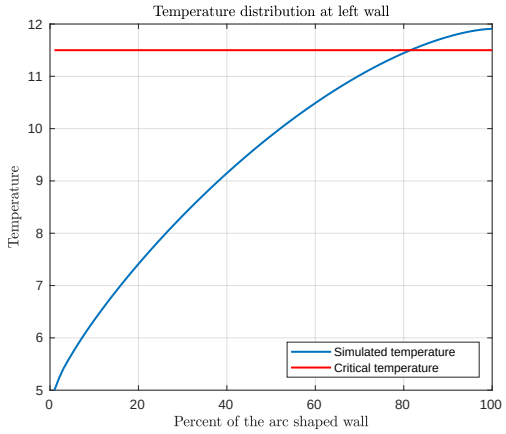


Fig. 5: Temperature distribution at the left wall

j) After this disaster the dwarf council decides to relocate the beer barrels in the cellar, such that in the (quite probable) case Eumel will again forget to close the door no beer will be harmed. Plot the regions of the cellar where even after $t = 300$ (the next morning) the temperature is still below 11.5.

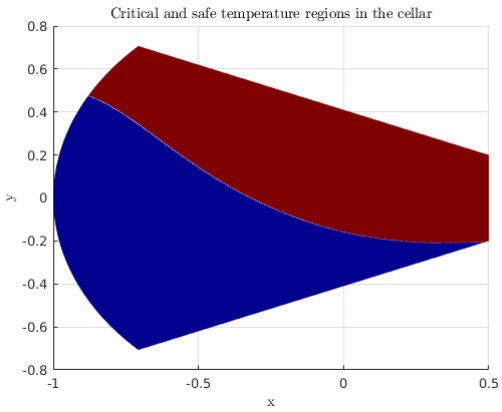


Fig. 6: Save (blue) and unsafe (red) regions of the cellar with open door after $t = 300$. The barrels should be relocated into the blue region to avoid overheating.

Master-task

M1) Quite similar to the heat problem in the cellar we now want to exploit the advantages of isogeometric analysis a bit more and really consider a curved object, like a "Weißbierglas". You can download from the lecture homepage the file `Beer_glas.m` which creates a NURBS geometry resembling a (2D) "Weißbierglas". As in exercise 7 repeat all the necessary steps to solve the heat equation $\partial_t u - \kappa \Delta u = 0$ under the following conditions:

- The initial temperature of the beer within the glass is $T_0 = 2$ (directly from a very cold fridge).
- At $t = 0$ the glass is placed on a table with (assumed to be constant) temperature $T_{\text{table}} = 8$.
- The ambient air has a maximal temperature of $T_{\text{air,max}} = 20$. However due to some cooling wind, the actual temperature oscillates around 3/4 of the maximal temperature in the following way: $T_{\text{air}} = \frac{T_{\text{air,max}}}{4}\left(3 + \sin(\frac{t}{10})\right)$. This temperature is prescribed as a (time-dependent) Dirichlet boundary condition at the top of the beer glass (where there is a direct "contact" beer-air).
- For the remaining three boundaries we somehow want to include the effect of the cold glas itself (it was also in the fridge and hence starts with temperature T_0) but without simulating a temperature-contact problem between glas and fluid. Hence we simply assume the glas temperature increases linearly from T_0 to $3/4 \cdot T_{\text{air,max}}$ on the left and right boundary and linearly from T_0 to T_{table} at the bottom boundary, where the goal-temperatures are reached after some time $t_{\text{saturation}} = 600$ sec. (10 min.) Once it reaches the goal, the temperature stays constant. Mathematically this means that at left and right boundary we prescribe the following (time-dependent) Dirichlet-conditions:

  $$u = \min\left\{T_0 + \frac{3/4 \cdot T_{\text{air,max}} - T_0}{t_{\text{saturation}}} \cdot t, 3/4 \cdot T_{\text{air,max}}\right\}$$

  and at the bottom we have:

  $$u = \min\left\{T_0 + \frac{T_{\text{table}} - T_0}{t_{\text{saturation}}} \cdot t, T_{\text{table}}\right\}$$

- Use $\kappa = 0.591$ as the heat conductivity constant of beer (actually water) and $\Delta t = 1$ sec. as time step size.

How long does it take for the mean-temperature to reach 11.5 °C?

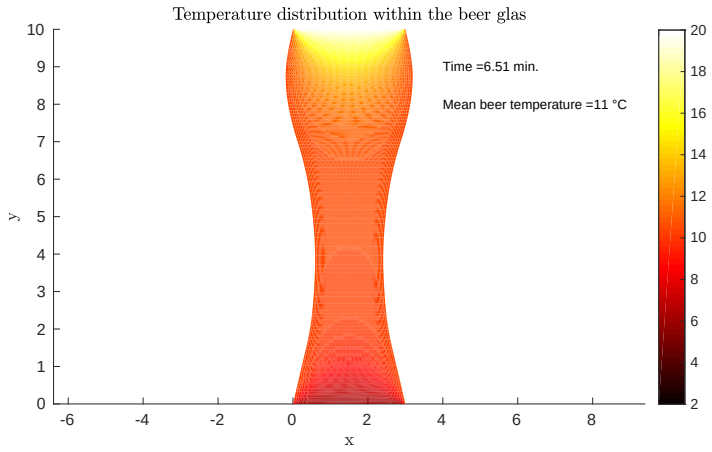


Fig. 7: Temperature distribution after 6.5 min. The mean temperature is still below 11.5 degree.

~~Exercise 22 (From linear to nonlinear PDEs: Fisher's/KPP equation)~~

Goal:

Solve Fisher's/KPP equation as a first example of a nonlinear PDE. To solve the nonlinear system resulting from discretization a fixpoint approach as well as the Newton method will be used.

We want to solve the following equation

$$\begin{aligned} \frac{\partial u}{\partial t} - D\Delta u - ru(1-u) &= 0 && \text{in } (0,3) \\ u &= 1 && \text{at } x = 0 \\ u &= 0 && \text{at } x = 3 \\ u(0,x) &= u_0(x) \end{aligned}$$

where $D > 0$ and $r \geq 0$ are parameters.

Stationary version

a) For the sake of simplicity (and computational speed) the spatial dimension of the PDE is chosen to be 1. In order to represent this with our GeoPDEs toolbox, you create the rectangle $[0,3] \times [0,1]$ where in y-direction only two DOFs (one at upper one at lower boundary) are used. Furthermore one prescribes homogeneous Neumann conditions at the boundaries $y = 0$ and $y = 1$. By doing so, the y-direction will only be resolved by one element and the solution will be constant w.r.t. y, while in x-direction you can refine to have at least 512 elements.

b) In the stationary case, $\partial_t u = 0$ and also the initial condition vanishes, hence we remain with $-D\Delta u - ru + ru^2 = 0$ and the two Dirichlet conditions. The usual variational formulation and integration by parts approach leads to:

$$D \int_0^3 \nabla u \cdot \nabla v \, \mathrm{d}x - r \int_0^3 uv \, \mathrm{d}x + r \int_0^3 uuv \, \mathrm{d}x = 0$$

where the first integral will again be discretized using the stiffness matrix $(K_{ij})_{i,j=1}^n$, the second integral via the mass matrix $(M_{ij})_{i,j=1}^n$ and for the third integral we have to introduce a three dimensional "mass-tensor" $(N_{ijk})_{i,j,k=1}^n$, where N_{ijk} is defined as

$$N_{ijk} = \int_0^3 N_i N_j N_k \, \mathrm{d}x$$

and (for the sake of notation) we will write $(N[\underline{x}, \cdot, \cdot])_{jk} := \sum_{i=1}^n N_{ijk} \underline{x}_i$ and likewise for the other entries.

- To assemble the tensor N you can download the functions `op_u_v_w_tp(..)` and `op_u_v_w(..)` from the lecture homepage. They are just simple extensions by one more index of the mass matrix assembly functions.

- To perform the tensor multiplications like $N[x,\cdot,\cdot]$ you can use the command `ttv` from the tensor toolbox available at http://www.sandia.gov/~tgkolda/TensorToolbox/index-2.6.html. The syntax is `ttv(N,x,d)`, where N is the tensor x the vector and d the direction from which to multiply the vector at the tensor. For example $N[x,\cdot,v]$ would be coded as `ttv( ttv(N, v, 3), x, 1)`.

c) Once all the matrices and tensors are assembled it remains to solve the following *non*-linear system of equations (coefficients and possible minus-signs are included in the matrices, i.e. for example $-r \cdot M \to M$)

$$K \cdot u + M \cdot u + N[u,\cdot,u] = 0$$

after incorporating the Dirichlet data by bringing them to the right hand side (this is what you should do in this exercise), it remains to solve for internal DOFs only:

$$\begin{aligned} K_{\text{int,int}} \cdot u_{\text{int}} + M_{\text{int,int}} \cdot u_{\text{int}} + N_{\text{int,int,int}}[u_{\text{int},},\cdot,u_{\text{int}}] = \\ -K_{\text{int,drchlt}} \cdot u_{\text{drchlt}} - M_{\text{int,drchlt}} \cdot u_{\text{drchlt}} - N_{\text{drchlt,int,drchlt}}[u_{\text{drchlt}},\cdot,u_{\text{drchlt}}] \end{aligned}$$

where we call the whole right hand side $\underline{f}$ and (for simplicity) $K_{\text{int,int}} =: K$ and like wise for $M_{\text{int,int}}$ etc. (i.e. we just drop the indices again). Now to solve:

$$K \cdot u + M \cdot u + N[u,\cdot,u] = \underline{f}$$

we will implement two different methods:

d) **Fixedpoint-iteration:** Assuming that the solution u^* fulfills

$$\begin{aligned} K \cdot u^* + M \cdot u^* + N[u^*,\cdot,u^*] &= \underline{f} \\ \Leftrightarrow (K+M) \cdot u^* &= \underline{f} - N[u^*,\cdot,u^*] \\ \Leftrightarrow u^* &= (K+M)^{-1}\left[\underline{f} - N[u^*,\cdot,u^*]\right] =: F(u^*) \end{aligned}$$

we start with some initial guess u^0 and iterate $u^{k+1} = F(u^k)$ for as long as the residuum $\text{Res} := K \cdot u^* + M \cdot u^* + N[u^*,\cdot,u^*] - \underline{f}$ is larger than some tolerance, i.e. $\|\text{Res}\|_2 \geq \text{TOL}$.

Implement this fixed-point approach and test it with the following set of parameters: $D = 1, r = 0.1, \text{TOL} =$`1e-8`. As an initial guess u^0 use the L^2 projection of the function $h(x) = 1 - \frac{x}{3}$.

Hint: To speed things up, make an LU-decomposition of $(K+M)$ before the iteration loop. Also do not forget to update the residuum after every iteration step, otherwise the loop will run forever.

Admitted: The result is quite boring, however, what happens in the much more interesting case of $r = 30$?

e) **Newton's method:** An alternative to the fixed-point iteration is Newton's method to find a solution of (again be careful, the indices are dropped): $F(u) := K \cdot u + M \cdot u + N[u,\cdot,u] - \underline{f} = 0$. Again starting with some initial guess u^0 the method reads as follows:

1.) Compute the Newton Update $\delta u := \left(JF(u^k)\right)^{-1} \cdot \left(-F(u^k)\right)$

2.) Update u^k to u^{k+1} via: $u^{k+1} = u^k + \delta u$

where $JF(u^k)$ is the Jacobian-matrix of F evaluated at u^k. For the given F the Jacobian computes to:

$$(JF(u))_{ij} = K_{ij} + M_{ij} + N[\cdot, \cdot, u]_{ji} + N[u, \cdot, \cdot]_{ij}$$

which due to the symmetries of N equals

$$JF(u) = K + M + 2\,N[\cdot, \cdot, u]$$

with the same initial guess, residual and tolerance use the Newton method to solve the PDE problem once again for $r = 0.1$ but also for $r = 30$.

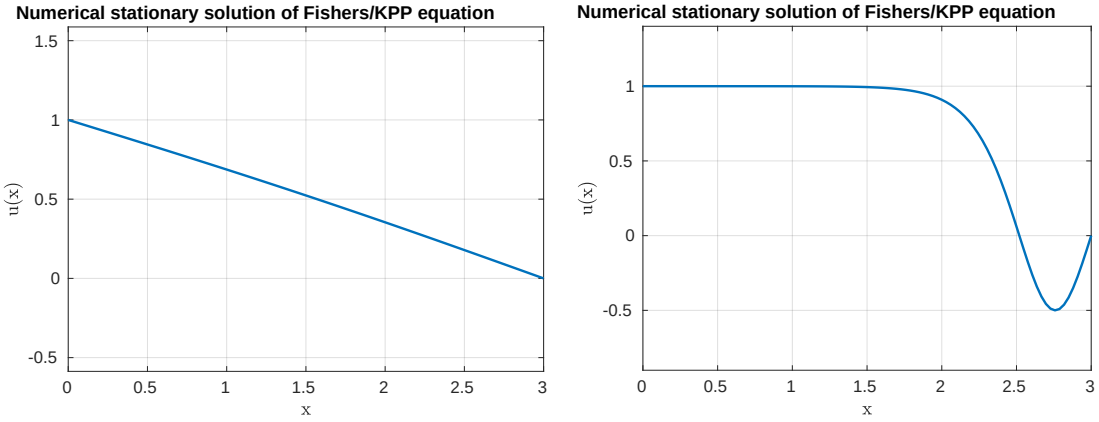


Fig. 8: left: $r = 0.1$, **right:** $r = 30$

Hint: To plot the solution, which is basically a cross-section of the (in y-direction constant) solution, you can use the following code:

```
figure(1)
samp = linspace(0, 1, 100), 0.5;
[u_eval, ~] = sp_eval (u, space, geometry, samp);
plot (linspace(0,3,100), u_eval);
```

Dynamical version (Master task)

In the dynamical case we again add the time derivative $\partial_t u$, which was left out so far, such that the equation again reads as in the very beginning of the exercise including the initial datum. You can use large parts of the stationary code also in the dynamical case, however you have to add at least the following:

a) Two versions of the mass matrix are needed. One including the factor $-r$ (as before called M) and an other one without the factor (for the time derivative $\dot{u}$), called M_t.

b) To start the simulation, project the initial datum $u_0(x) = \chi_{[0,1/10]}(x)$ (characteristic function of the interval $[0, 1/10]$) into the NURBS space and use the result for the solution vector `u`.

As in the heat-equation example we use the implicite Euler method for time discretization, i.e.

$$M_t \cdot \frac{\underline{u}^{k+1} - \underline{u}^k}{\Delta t} + K \cdot \underline{u}^{k+1} + M \cdot \underline{u}^{k+1} + N[\underline{u}^{k+1}, \cdot, \underline{u}^{k+1}] = \underline{0}$$

bringing Dirichlet DOFs to the right hand side one ends up with:

$$\begin{aligned} M_{t,\text{int,int}} \cdot \underline{u}^{k+1}_{\text{int}} + \Delta t\,(K+M)_{\text{int,int}} \cdot \underline{u}^{k+1}_{\text{int}} + \Delta t\, N_{\text{int,int,int}}[\underline{u}^{k+1}_{\text{int}}, \cdot, \underline{u}^{k+1}_{\text{int}}] = \\ M_{t,\text{int,int}} \cdot \underline{u}^{k}_{\text{int}} - M_{t,\text{int,drchlt}} \cdot \underbrace{\left(\underline{u}^{k+1}_{\text{drchlt}} - \underline{u}^{k}_{\text{drchlt}}\right)}_{=0} - \Delta t (K+M)_{\text{int,drchlt}} \cdot \underline{u}^{k+1}_{\text{drchlt}} \\ - \Delta t N_{\text{drchlt,int,drchlt}}[\underline{u}^{k+1}_{\text{drchlt}}, \cdot, \underline{u}^{k+1}_{\text{drchlt}}] \end{aligned}$$

calling the right hand side again $\underline{f}$ and dropping the indices on the left hand side (just for notational convenience, we still only solve for internal DOFs), this gets:

$$(M_t + \Delta t(K+M)) \cdot \underline{u}^{k+1} + \Delta t N[\underline{u}^{k+1}, \cdot, \underline{u}^{k+1}] = \underline{f}$$

which is again a nonlinear system that has to be solved to get $\underline{u}^{k+1}$.

c) Of course you have to wrap the nonlinear-iteration solver (here we will only use Newton's method) into a time stepping loop, since you want to solve the nonlinear system in every timestep. Within a given timestep $k \to k+1$ you proceed as follows:

c1) As an initial guess $\underline{u}^{k+1}_0$, solve the system only for the linear part, while for the nonlinear you use the solution of the previous timestep, i.e.

$$\begin{aligned} (M_t + \Delta t(K+M)) \cdot \underline{u}^{k+1}_0 &= \underline{f} - \Delta t N[\underline{u}^k, \cdot, \underline{u}^k] \\ \underline{u}^{k+1}_0 &= (M_t + \Delta t(K+M))^{-1} \cdot \left(\underline{f} - \Delta t N[\underline{u}^k, \cdot, \underline{u}^k]\right) \end{aligned}$$

c2) Compute the residuum

$$\text{Res} = (M_t + \Delta t(K+M)) \cdot \underline{u}^{k+1}_0 + \Delta t N[\underline{u}^{k+1}_0, \cdot, \underline{u}^{k+1}_0] - \underline{f}$$

c3) As long as the norm of the residuum is above `1e-8` repeat the Newton-update

$$\begin{aligned} \delta \underline{u}^{k+1}_{n+1} &= \left(JF(\underline{u}^{k+1}_n)\right)^{-1} \cdot \left(-\text{Res}(\underline{u}^{k+1}_n)\right) \\ \underline{u}^{k+1}_{n+1} &= \underline{u}^{k+1}_n + \delta \underline{u}^{k+1}_{n+1} \\ \text{Res} &= (M_t + \Delta t(K+M)) \cdot \underline{u}^{k+1}_{n+1} + \Delta t N[\underline{u}^{k+1}_{n+1}, \cdot, \underline{u}^{k+1}_{n+1}] - \underline{f} \end{aligned}$$

Here the upper index $k+1$ stands for time step $k+1$ while the lower index n resp. $n+1$ indicates the nonlinear iteration number.

Hint: The Jacobian for the dynamical problem reads:

$$JF(\underline{u}) = M_t + \Delta t\,(K+M) + 2\Delta t\, N[\cdot, \cdot, \underline{u}]$$

d) Use again at least 512 elements in x-direction, parameters $D = 0.1$ and $r = 200$, final time $T = 0.5$ and around 1000 timesteps to solve the problem. The solution should look similar to the following pictures. It shows a so called traveling wave phenomenon:

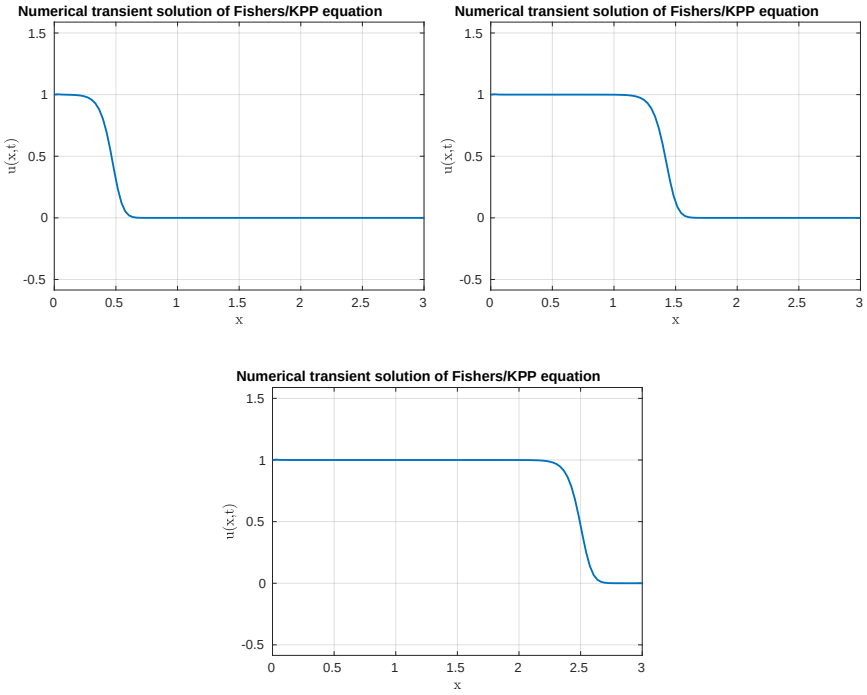


Fig. 9: top left: $\approx 10\%$ of simulation time, **top right:** $\approx 30\%$ of simulation time, **bottom:** $\approx 50\%$ of simulation time

---

~~Exercise 23 (Simulation of a wave-pool fountain – Wave-equation)~~

Goal:

Simulate a nice special setting for the linear wave equation. As in the heat-equation case, a suitable time integrator is needed, but this time for a second order in time equation. In addition one has to be careful how the mesh for a circle has to be created.

Very roughly speaking we want to recreate / simulate the following behavior of a wave: https://youtu.be/RHTcSKkUU8U?t=32s, i.e. a circular wave pool with a concentric excitation at the boundary where the converging waves will form a "wave-singularity" at the center of the pool. To do this, we will solve the linear wave equation $\partial_{tt}u - c^2\Delta u = 0$ over the unit disc, where c is the wave speed (speed of sound if it was an acoustic wave). Mathematically the problem reads:

$$\begin{aligned} \frac{\partial^2 u}{\partial t^2} - c^2\Delta u = 0 \qquad & \text{in } \Omega = \mathbb{B}_1(0) \\ u = a \cdot \sin(2\pi \cdot f \cdot t) \qquad & \text{on } \partial\Omega \\ u(0,x) = u_t(0,x) = 0 & \end{aligned}$$

a) Create the unit disc as a NURBS domain. The idea is to start with a torus segment and then drive the inner radius to zero, while the outer angle tends to 2π (close the segment). For the curved torus parts, you can again use the command `nrbcirc(..,..,..,..)`. The following picture shows the strategy:

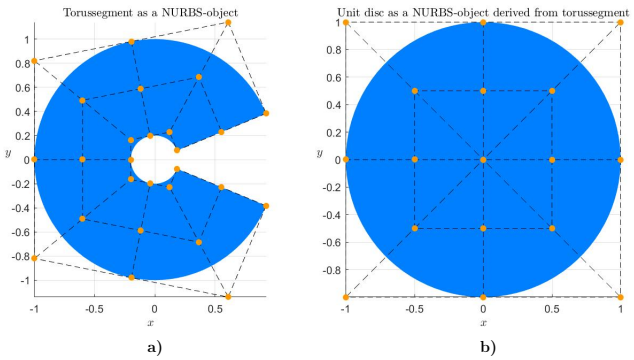


Fig. 10: a) the torus segment, **b)** the "closed" torus segment, i.e. the unit disc.

Once the unit disc is created, refine it to around 4500 degrees of freedom

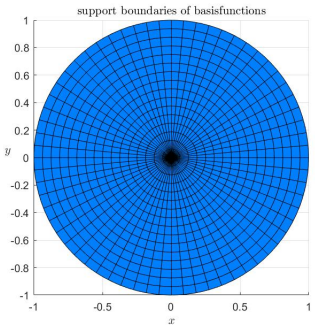


Fig. 11: Grid of the unit disc.

b) Set up the usual data structures like mass and stiffness matrices. What is new: You have to add lines and columns of those matrices corresponding to the degrees of freedom that got "glued together" either on that part, where the torus segment was closed or at the center, where all degrees of freedom of the inner arc of the torus fall together. It might be helpful to write a function `identify_dofs(..,..,..)` for this purpose which translates between the two systems of numeration.

c) **From now on everything in the "glued together"-DOF-system:** Since it is a dynamical problem you have to loop over the time DOFs. The time stepping itself is done by the Newmark-Predictor-Corrector-method, which reads as follows:

Predictor step:

$$u_{\text{pred}} = u_n + \Delta t v_n + \frac{\Delta t^2}{2}(1-2\beta)a_n$$
$$v_{\text{pred}} = v_n + \Delta t(1-\gamma)a_n$$

Solve linear system:

$$a_{n+1} = (M + \beta \Delta t^2 c^2 K)^{-1} \cdot \left(\text{rhs} - c^2 K u_{\text{pred}}\right)$$

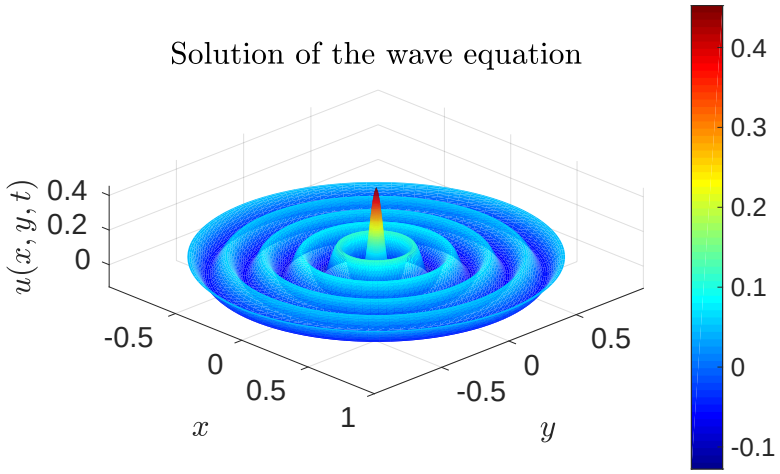
Corrector step:

$$u_{n+1} = u_{\text{pred}} + \beta \Delta t^2 a_{n+1}$$
$$v_{n+1} = v_{\text{pred}} + \gamma \Delta t a_{n+1}$$

Here v stands for u_t and a for u_{tt}, while β, γ are parameters of the Newmark-scheme (usually $\gamma = 1/2, \beta = 1/4$). As in the last exercises the time-stepping method is only applied to internal DOFs, the Dirichlet DOFs are set to their values at the beginning of the time step and then carried to the right hand side, where they enter the Newmark scheme in the linear solver step as

$$\text{rhs} = -M_{\text{int,drchlt}} \cdot a_{\text{drchlt}} - K_{\text{int,drchlt}} \cdot u_{\text{drchlt}}$$

d) At the end (or wherever you want to plot something) you have to translate back from the "glued-together" DOF system to the original one in order to use the plot commands from our last exercises. With the following parameters, start the simulation: $T_{\text{final}} = 3$, `time_ndof` $= 2000$, $c = 8, f = 40, a = 0.05$.



~~Exercise 24 (A wave equation from nonlinear acoustics - Westervelt's equation)~~

Goal:

Westervelt's equation is a non-linear extension to the linear wave equation. Solve this equation to study the influence of nonlinearity to a (shock)-wave front.

This exercise is quite similar to the wave-pool singularity exercise. On the one hand, it is even simpler, since no identification (gluing together) of DOFs is necessary as the domain is simpler. On the other hand, the equation itself is again a non-linear PDE, hence we have to include

again a nonlinearity loop (this time a fixpoint iteration) into the Newmark-time integrator. The initial-boundary-value problem we want to solve reads:

$$\begin{aligned} \ddot{\psi} - c^2\Delta\psi - b\Delta\dot{\psi} - 2k\rho\dot{\psi}\ddot{\psi} &= 0 && \text{in } [0, l_{\text{channel}}] \\ \frac{\partial\psi}{\partial\underline{n}} &= \psi_{\text{exc}} && \text{at } x = 0 \\ \frac{\partial\psi}{\partial\underline{n}} &= 0 && \text{at } x = l_{\text{channel}} \\ \psi(0,x) = \dot{\psi}(0,x) &= 0 \end{aligned}$$

where ψ is the so called "acoustic potential". It relates to the pressure in the following way: $p = \rho\dot{\psi}$.

This equation is called Westervelt equation (in potential form) and it is used to model the propagation of high intensity focused ultrasound (HIFU) used for example in extracorporeal shock wave lithotripsy when treating kidney stones. The usual (linear) wave equation would not be sufficient, since it can not resolve effects like the steepening of the wave front and especially not the "breaching" of the wave after the shock formation distance. These are effects of *non*-linear acoustics, however in the before mentioned application these effects are important to consider!

a) The spatial domain is again one-dimensional, hence you proceed as before and create a channel of length 0.3 and unit width, where in y-direction homogeneous Neumann conditions create the one-dimensional setting while in x-direction you use the conditions (also Neumann) that are prescribed.

b) Use the following material and discretization parameters.
   - Speed of sound $c = 1500$, Diffusivity of sound $b = 6e-9$, density $\rho = 1000$, $B/A = 5$, $k = \frac{1}{\rho c^2}\left(1 + \frac{B}{2A}\right)$
   - $\psi_{\text{exc}} = a\sin(\omega\cdot t)\cdot\sin(\frac{\omega}{4}\cdot t)$ for $t < \frac{4\pi}{2\omega}$ otherwise $\psi_{\text{exc}} = a\sin(\omega\cdot t)$ with $a = 38, f = 7e4, \omega = 2\pi\cdot f$
   - Newmark-parameter $\beta = 0.45, \gamma = 0.75$
   - Number of elements in x-direction: 4096
   - Final time $T = 0.85\cdot 9e-5$, number of time DOFs: 8751
   - Nonlinear tolerance $1e-6$

c) The weak form reads:

$$\int_\Omega \ddot{\psi}v\,\mathrm{d}\Omega + c^2\int_\Omega \nabla\psi\cdot\nabla v\,\mathrm{d}\Omega + b\int_\Omega \nabla\dot{\psi}\cdot\nabla v\,\mathrm{d}\Omega + 2k\rho\int_\Omega \dot{\psi}\ddot{\psi}v\,\mathrm{d}\Omega = c^2\int_{\Gamma_{\text{exc.}}} \psi_{\text{exc}}(t)v\,\mathrm{d}S + b\int_{\Gamma_{\text{exc.}}} \dot{\psi}_{\text{exc}}(t)v\,\mathrm{d}S$$

with the usual notion of mass- and stiffness matrices, the tensor N and $u = \psi, v = \dot{\psi}, a = \ddot{\psi}$ the discrete version becomes:

$$Ma + c^2Ku + bKv + 2k\rho N[\cdot, a, v] = c^2\underline{G}(t) + b\dot{\underline{G}}(t)$$

again including the prefactors into the matrices / tensors this equals:

$$Ma + Ku + Cv + N[\cdot, a, v] = \underline{G}(t) + \dot{\underline{G}}(t)$$

Set up all these necessary quantities and compute an initial value for a via $a_0 = M^{-1}\cdot\left(\underline{G}(0) + \dot{\underline{G}}(0) - C\cdot v_0 - K\cdot u_0\right)$

d) Include a nonlinearity loop (fixpoint iteration) into the Newmark scheme such that in every timestep the following steps are executed:

Set $k = 1$ (k is the nonlinear-iteration counter, while the subscript n marks the time step)

Predictor step:

$$\begin{aligned} u_{n+1}^k &= u_{\text{pred}} = u_n + \Delta t v_n + \frac{\Delta t^2}{2}(1-2\beta)a_n \\ v_{n+1}^k &= v_{\text{pred}} = v_n + \Delta t(1-\gamma)a_n \\ a_{n+1}^k &= a_n \end{aligned}$$

while stopping criterion is not matched, repeat all the following:

Solve linear system:

$$a_{n+1}^{k+1} = (M + \gamma\Delta t C + \beta\Delta t^2 K)^{-1} \cdot \left(\underline{G}(t_{n+1}) + \dot{\underline{G}}(t_{n+1}) - K u_{\text{pred}} - C v_{\text{pred}} + N[\cdot, v_{n+1}^k, a_{n+1}^k] \right)$$

Corrector step:

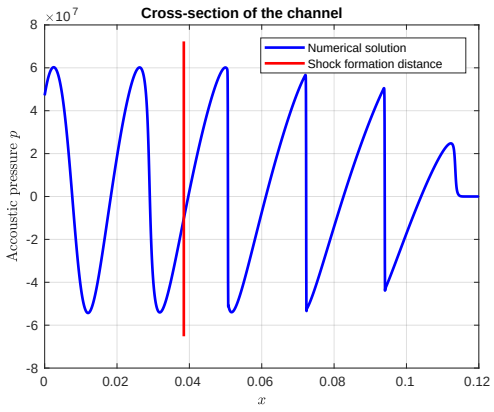
$$\begin{aligned} u_{n+1}^{k+1} &= u_{\text{pred}} + \beta\Delta t^2 a_{n+1}^{k+1} \\ v_{n+1}^{k+1} &= v_{\text{pred}} + \gamma\Delta t a_{n+1}^{k+1} \end{aligned}$$

Set $k = k + 1$

e) As a stopping criterion use:

$$\frac{\|a_{n+1}^{k+1} - a_{n+1}^k\|_2}{\|a_{n+1}^{k+1}\|_2} \leq \text{TOL}$$

f) **Post-processing:** Plot the pressure-cross-section of the channel (remember the solution of the PDE was the acoustic potential, hence you have to postprocess $p = \rho\dot{\psi}$).



g*) Also plot the pressure signal at some specific positions in the channel, i.e. no plot w.r.t. space but at one fixed location a plot w.r.t. time! The positions where to plot are given by $x = \sigma \cdot \overline{x}$ with $\sigma \in \{0.2, 0.6, 2\}$ and $\overline{x} := \frac{c^2}{\omega \cdot a \cdot \left(1 + \frac{B}{2A}\right)}$ being the so called shock-formation distance.