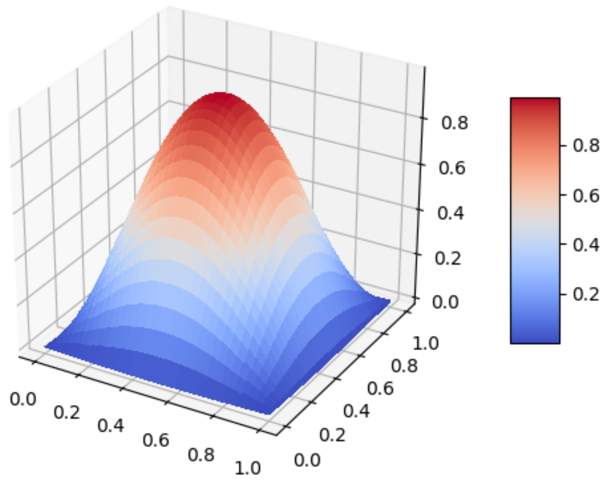
A Demonstration Code For Isogeometric Elements

Rafael Bramm, Hai Dang Nguyen, Daniel Stefani



Project-INF

Course: Bachelor of Science Informatik
Examiner: Prof. Dr. rer. nat. habil. Miriam Mehl
Supervisor: Dr. rer. nat. Stefan Zimmer

Institute for Parallel and Distributed Systems
University of Stuttgart
Germany

Start: 10.03.2019
End: 10.09.2019

Abstract

This project is a demonstration code of an Isogeometric Analysis (IGA) tool written in python to solve one (1D) and two (2D) dimensional L2 best approximations and Poisson's equations. IGA uses a Galerkin method similar to the Finite Element Method (FEM). This paper describes the maths and structure of the implemented program, also some examples are listed. The described program has cutbacks in different fields, like supported dimensions, to assure good readability. It is slightly oriented after the GeoPDEs tool described in [1] and [4].

1 Introduction

Exhaustive research in Isogeometric Analysis began in the 2000s. One of the pioneers of Isogeometric Analysis is T.J.R. Hughes, who has arguably written the first paper [3] about IGA. Since then the interest in IGA grew because of benefits like the exact geometrical domain that is used in comparison to Finite Element Method (FEM), which aren't a direct topic of this paper. GeoPDEs is a more complex tool, but open source program, which is recommended for deeper interest in this field.
This paper is structured as follows. Section 2 lists the basics of the mathematical methods used in the program. Section 3 describes the implementation and the usage of the program. Examples are presented in section 4. Section 5 gives a short discussion, section 6 gives a summary and section 7 an short outlook into further implementation possibilities.

2 Basics

This is just a short overview of the used maths, further information on Isogeometric Analysis are found in [4] and [3]. The Isogeometric Analysis is computing a numerical approximation for the solution of a partial differential equation (PDE) with a Galerkin method, which means a system of equations needs to be solved. The Finite Element Method (FEM) also uses a Galerkin method to compute numerical approximations of the solutions of PDE's. In both cases a system of linear equations needs to be solved to determine the solution. The IGA program finds an u_h such that:

$$a(u_h, v_h) = (f, v_h) \forall v_h \in V_h \tag{1}$$

While $a(\cdot, \cdot)$ is a bilinearform, $(\cdot, \cdot)$ is the L^2 Inner Product and V_h a set of Spline basis functions. In this program the equation will be solvable for 1D/2D L2 best approximation and Poisson's problem.

2.1 Workflow

This workflow describes the maths, but also the program, this way the user can stick to this order.

1. The mesh is built from the number of intervals n and the dimension k of the basis functions.
2. Basis functions are constructed as B-Splines with the mesh.
3. The stiffness matrix A is computed with the basis functions, also a vector b with the input function and the basis.
4. After solving the system of equations $A\alpha = b$ for α the result function is generated.

2.2 Numerical Quadrature

For the approximation integrals need to be computed, which are handled by the Gauss-Legendre Quadrature in every method of the program. Therefore Gauss-Legendre points and weights are generated for a domain [a,b]. The sum over the function evaluated in these quadrature points multiplied by the corresponding weight α_i approximates the integral.

$$\int_a^b f(x)\mathrm{d}x \approx \frac{b-a}{2}\sum_{i=1}^{n} f(\frac{b-a}{2}x_i + \frac{a+b}{2}) \cdot \alpha_i \tag{2}$$

2.3 Domain, Mesh and Functions

All computation in the program are done in the fixed interval $\hat{\Omega} = [0,1]^D$ where D is the dimension of the evaluated problem. This will improve the readability of the code in program and simplify the computation of a mesh because of the fixed boundaries. But not every new problem is defined on this fixed interval. Therefore a transformation from the fixed interval $\hat{\Omega}$ to the target interval Ω is introduced in the subsection Transformation.
The knot vector t in $\hat{\Omega}$ is constructed by setting the number of intervals n and the dimension of the used basis functions k. With this vector Gauss-Legendre quadrature points in every interval are generated.
Because every function only needs to be evaluated in the finite point set of the mesh, they are saved as vectors, which are evaluated only in these points. This reduces the overhead of recomputing the values for each function.

2.4 Basis Functions

The basis in this program is a set consisting of B-Spline functions. The 1D B-Spline basis has m=n+k functions, which are derived with the previously established mesh. The B-Spline constructor is given the knot vector t, the position c where the function should be different from 0 and the degree k of the function to compute the m basis functions. The 2D B-Spline functions are evaluated by the pairwise tensor product of the 1D B-Spline functions.

2.5 Result Function

This function is computed with the solution u_h of a problem, which corresponds to α in the program. The α_i are used as coefficients for the basis functions ϕ_i. This function is given in $\hat{\Omega}$ and is transformed via push-forward to Ω.

$$r = \sum_i \alpha_i \phi_i \tag{3}$$

2.6 Transformation

A transformation function F is used to map the result function from the computational domain $\hat{\Omega}$ to the physical domain Ω. To achieve a solid transformation the computations in the solver modules are fixed by additional factors in form of an integral transformation and a derivative transformation following a push-forward into the target domain.

2.6.1 Push-Forward

This directly transforms the coordinates into the target domain: $\Omega = F(\hat{\Omega})$

$\hat{x}$	0	1	2	3	4	5
$f(\hat{x})$	3	4	5	6	7	8

Table 1: 1D function vector before transformation

The function values remain unchanged, but the evaluated $\hat{x}$ changes to $F(\hat{x})$.

x = F($\hat{x}$)	0	2	4	6	8	10
$f(x)$	3	4	5	6	7	8

Table 2: 1D function vector after transformation with 2x

2.6.2 Integral Transformation

An integral is transformed with the following formula where F is the transformation function and J_F the jacobian matrix of F.

$$\int_{\Omega} f(x)\mathrm{d}x = \int_{\hat{\Omega}} f(F(\hat{x}))|J_F|\mathrm{d}\hat{x} = \int_{\hat{\Omega}} \hat{f}((\hat{x})|J_F|\mathrm{d}\hat{x} \tag{4}$$

The jacobian determinant is computed in equation (5) and is the absolute value of the derivation in the 1D case.

$$|J_F| = \sqrt{det(J_F^T J_F)} \tag{5}$$

2.6.3 Derivative Transformation

The derivatives in the program need to be transformed as well. The $grad(\cdot)$ operator denotes the gradient of a function.

$$grad(f)|_{x=F(\hat{x})} = (J_F^+|_{\hat{x}})^T grad(\hat{f}|_{\hat{x}}) \tag{6}$$

The J_F^+ is the Moore-Penrose pseudoinverse, computed with the formula in (7).

$$(J_F^+|_{\hat{x}})^T = (J_F^T J_F)^{-1} J_F \tag{7}$$

2.7 L2 best approximation

The goal of this approximation is to find a function, such that equation 8 holds true:

$$u = \sum_i \alpha_i \cdot \phi_i \text{ with } u \overset{!}{=} f \tag{8}$$

The ϕ_i are the basis functions of the mesh. This approximation is handled with the L2 inner Scalar product with a integral transformation, which is defined as follows:

$$a_{L2}(u, v) = \int_{\hat{\Omega}} u(\hat{x}) \cdot v(\hat{x}) \cdot |J_F(\hat{x})| d\hat{x} \tag{9}$$

The stiffness matrix A and the vector b which build the system of linear equations follow like this:

$$A = \begin{pmatrix} a_{L2}(\phi_0, \phi_0) & \cdots & a_{L2}(\phi_0, \phi_{m-1}) \\ \vdots & \ddots & \vdots \\ a_{L2}(\phi_{m-1}, \phi_0) & \cdots & a_{L2}(\phi_{m-1}, \phi_{m-1}) \end{pmatrix} b = \begin{pmatrix} a_{L2}(f, \phi_0) \\ \vdots \\ a_{L2}(f, \phi_{m-1}) \end{pmatrix} \tag{10}$$

The solution of $A\alpha = b$ is used to built the result function with $r = \sum_i \alpha_i \cdot \phi_i$. This solution function is pushed forward by plotting it in the target domain.

2.8 Poisson's Equation

The goal of this approximation is to find a function, such that equation 11 holds true:

$$u = \sum_i \alpha_i \cdot \phi_i \text{ such that } -\triangle u \overset{!}{=} f \tag{11}$$

For Poisson's equation boundary conditions need to be set, which are chosen as Dirichlet boundary conditions. They are set individually by a 2 element vector in 1D and fixed for corners and four boundary functions $(b_d(x), b_r(y), b_l(y), b_u(x))$ for every edge in 2D. Those boundary points in 2D are computed with the 1D L2 best approximation with transformation, while one entry of the 2D transformation function is fixed for every edge to construct a 1D function from the

2D. The stiffness matrix for non boundary points is set up the same as for L2 approximation, except it is using the Laplacian operator with derivative and integral transformations

$$a_{Lap}(\phi_i, \phi_j) = \int_{\hat{\Omega}} ((J_F^{+}|_{\hat{x}})^T \cdot \bigtriangledown u(\hat{x}))^T \cdot ((J_F^{+}|_{\hat{x}})^T \cdot \bigtriangledown v(\hat{x})) \cdot |J_F(\hat{x})| d\hat{x} \tag{12}$$

Computed for all $i, j < m^D$ (D is the dimension - in our case 1 or 2). The vector b is computed exactly like the one in the L2 approximation. A special feature in this program is the reordering of the matrix in the 2D Poisson solver. The corners of the interval are set to fixed values 1, followed by the entry of the transformed edges which is solved first in a separate stiffness matrix A and vector b. Finally the inner points of the problem are added and a complete solution is computed. This way the values of the matrix can be printed and evaluated individually by an interested user. The distribution of the values is also shown in figure 1. The darker colored parts consist of non zero values. The ordering in corner, edges/boundaries and inner points is denoted by the red lines. After solving $A\alpha = b$, the result function is computed like for the L2 best approximation and pushed into the target domain.

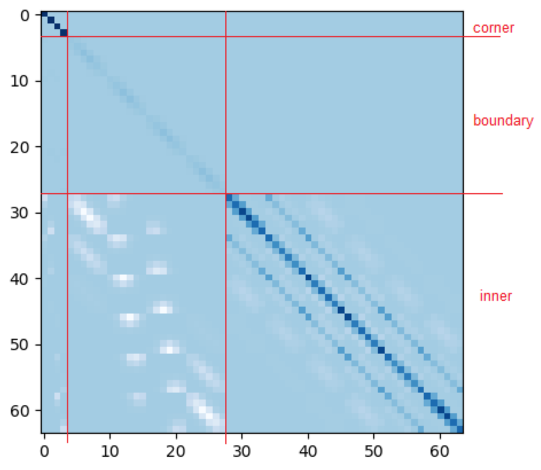


Figure 1: Colored Matrix in Poisson 2D

3 Implementation

Our program is written in Python and the user will need to have python3 including the libraries numpy, scipy and matplotlib installed to run it.

3.1 Program Structure

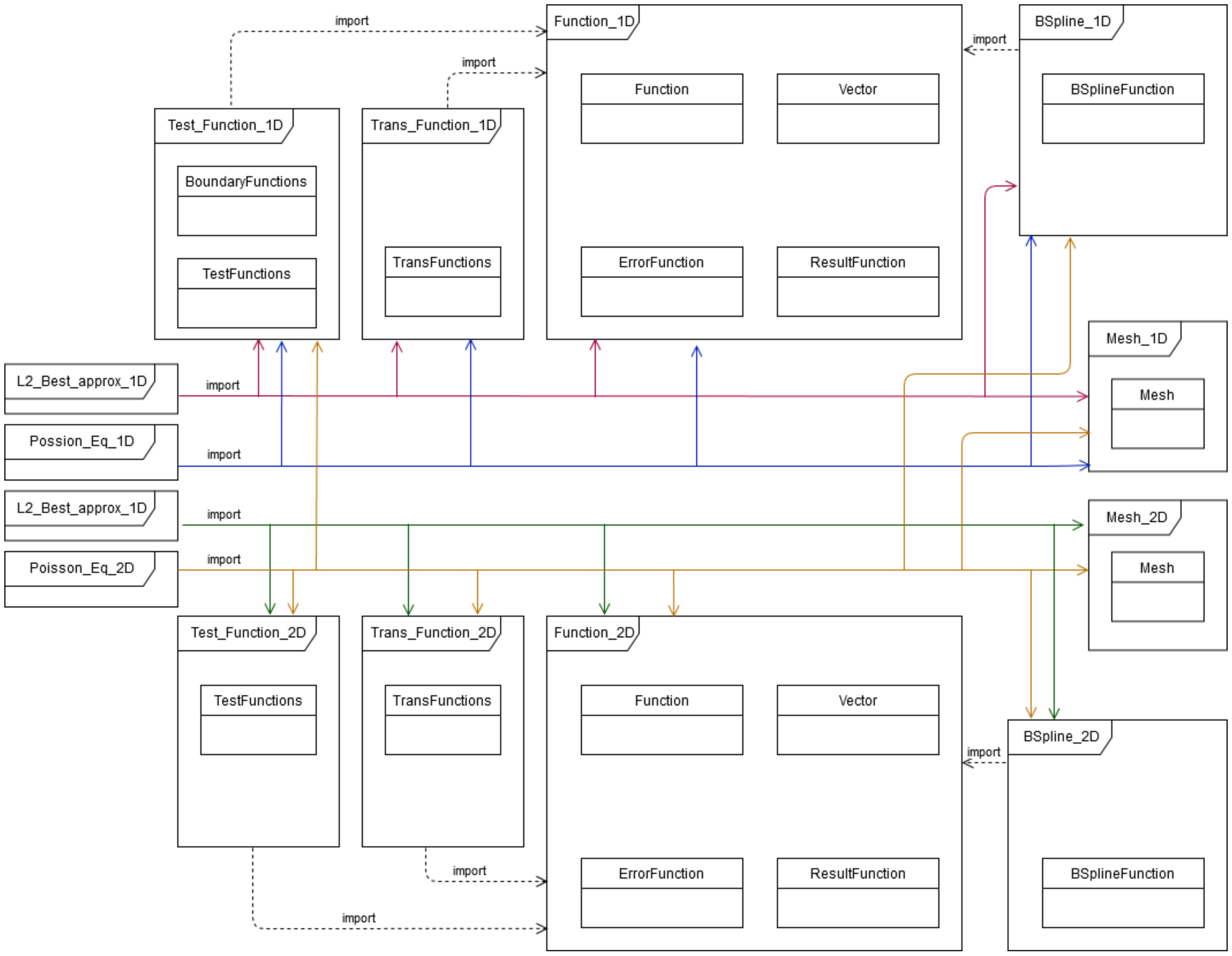


Figure 2: Python modules

Our program is organized in different modules: there are 4 solver modules and 10 supporting modules.
The modules `Function_1D` and `Function_2D` contain the `Function` class which is used to represent mathematical functions. All other functions classes inherit from this class and those modules are imported by all the others using functions. Those modules also define the vector form of functions and the various plot functions of the program.
The modules `Mesh_1D` and `Mesh_2D` hold information about calculation parameters such as the degree of our basis functions k, the grid size n and the quadrature

points and their respective weights. The mesh modules are inspired by [2] and [1].
`Test_Function_1D`, `Trans_Function_1D` and their two dimensional counterparts contain sample functions. A user could extend those for additional function. Other than that, they are not needed in the calculation.

Furthermore, there are modules for the basis functions, in our case we are using B-Splines. Lastly, there are the solver modules, where the computation is done. Each solver module can solve one problem type and imports all necessary supporting modules. As an example, the module `Poisson_Eq_2D` solves the Poisson's Equation for two dimensional functions. For that it has to import the module for two dimensional functions, B-Splines, transformation functions, test functions and the two dimensional mesh. Since Poisson's Equation has boundary conditions, also `Test_Function_1D`, `BSpline_1D` and `Mesh_1D` are needed. The relationship between all modules can be seen in Figure 2.

3.2 Programflow

The program flow is similar to the workflow described in 2.1. Additionally, our program will draw the result function as well as other relevant plots. This step however can be quite time and memory consuming.

3.2.1 Usage

To solve a problem the user has to find and open the relevant solver module. In each of those modules is a main function. There the user has to define the test function, the tranformation function and, as may be the case, the boundary function (or a list of boundary functions in the case of `Poisson_Eq_2D`). For Poisson's Equation there is the possibility to add an exact solution, provided the user wants to ascertain the exactitude of the solution of this program. If the user does this, they should be aware to use the identity as transformation function.
The user can use the sample functions we provided but may also define their own functions. After that, the user needs to decide on the amount of intervalls (n) in the grid as well as the degree of the basis functions (k). Higher parameters will yield more exact solutions but also increase the computation time. The user can also activate or deactivate the push-forward, which is not recommended. Solutions, which are not mapped into the target space, were a subject of testing, but have actually no usage for a user. The user can also tell the program, if the input function is already defined on $\hat{\Omega}$, or in the Ω domain. If the input is defined on $\hat{\Omega}$ no additional measures have to be taken, but if it is defined on Ω the points are evaluated in F(x) instead of x. This is essentially a reversal of a push-forward. Lastly the user can modify the plot functionss to his likings and afterwards call the solve() method with all relevant parameters.

4 Examples

4.1 Example 1: One dimensional L2 best approximation

This example is chosen especially easy, such that the calculation can be shown. The test function is set to

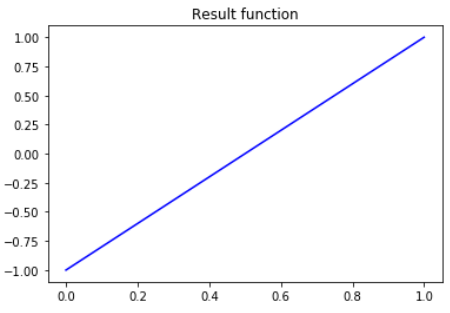
$$f(x) = 2x - 1 \text{ such that } u(x) \overset{!}{=} 2x - 1$$

The transformation function is chosen as: $F(x) = x$
This corresponds in all factors added through transformation in being 1. They have no effect on any output. Also the push-forward at the end is an identity function, which can be neglected.
The number n of intervals is chosen as 2 and the dimension k of the B-Splines is chosen as 1. The corresponding three B-Spline functions are:

$$\phi_0(x) = \begin{cases} -2x+1 & \text{for } 0 \le x \le \frac{1}{2} \\ 0 & \frac{1}{2} < x \le 1 \end{cases} \quad \phi_1(x) = \begin{cases} 2x & \text{for } 0 \le x \le \frac{1}{2} \\ -2x+2 & \frac{1}{2} < x \le 1 \end{cases}$$

$$\phi_2(x) = \begin{cases} 0 & \text{for } 0 \le x \le \frac{1}{2} \\ 2x-1 & \frac{1}{2} < x \le 1 \end{cases}$$

This stiffness matrix $A = \begin{pmatrix} \frac{1}{6} & \frac{1}{12} & 0 \\ \frac{1}{12} & \frac{1}{3} & \frac{1}{12} \\ 0 & \frac{1}{12} & \frac{1}{6} \end{pmatrix}$ and right hand side b $= \begin{pmatrix} -\frac{1}{6} & 0 & \frac{1}{6} \end{pmatrix}^T$ follow. Solving the system of equations $A\alpha = b$ results in $\alpha = \begin{pmatrix} -1 & 0 & 1 \end{pmatrix}^T$
The result function is computed like $r(x) = -1 \cdot \phi_0(x) + 1 \cdot \phi_2(x) = 2x - 1$, which is the exact analytical solution.
The left plot is the computed function of the program and the right one shows the deviation from the exact solution s(x). The plots show the high accuracy of the method with the deviation being near zero.



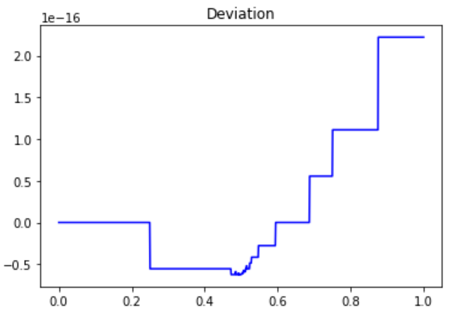


Figure 3: numerical result function r(x) (left)
deviation $d(x) = s(x) - r(x)$ (right)

4.2 Example 2: Two dimensional Poisson's Equation

This example represents the Laplace equation in 2D.

$$- \triangle u \overset{!}{=} 0$$

The lower bound is chosen as $b_d(x) = 0$, the right bound as $b_r(y) = y$, the left bound as $b_l(y) = 0$ and the upper bound as $b_u(x) = x$ to set the values in the edges of the interval. The transformation function is set to: $F(x, y) = (x, y)$
This corresponds in all factors added through transformation in being the identity matrix. They have no effect on any output. Also the push-forward at the end is an identity function, which can be neglected.
The number n of intervals is chosen as 4 and the dimension k of the B-Splines is chosen as 2. The exact solution is $s(x, y) = x \cdot y$ and $r(x, y)$ is the result function of the program. The following plot shows the deviation of the program output in comparison to the exact solution.

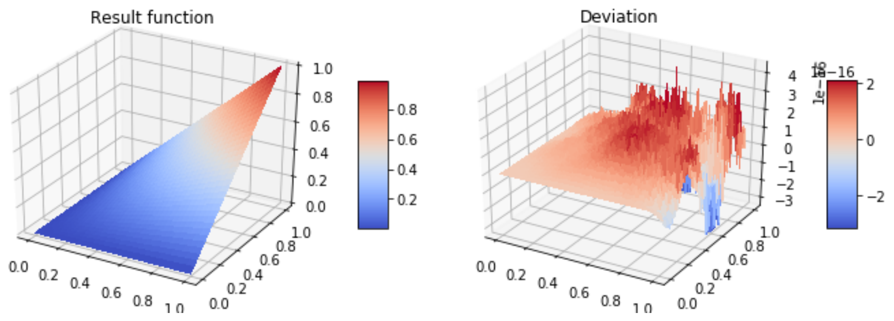


Figure 4: result function r(x,y) (left), deviation $d(x, y) = s(x, y) - r(x, y)$ (right)

In this example the deviation is also nearly 0 in every point.

4.3 Example 3: Two dimensional L2 best approximation

This example function is given as:

$$f(x, y) = \sin(\pi x) \cdot \sin(\pi y)$$

The transformation function is set to: $F(x, y) = \begin{pmatrix} x^2 + y^2 \\ y^2 \end{pmatrix}$
The general jacobi matrix is $J_F = \begin{pmatrix} 2x & 2y \\ 0 & 2y \end{pmatrix}$
The transformation factors and the final push-forward remap the function into target domain Ω. The following plot shows the transformed function in the target domain. The title page also shows the approximated function, but it is transformed with the identity.

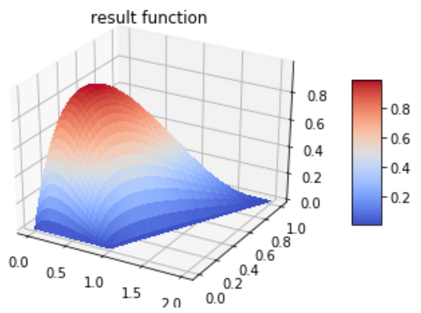


Figure 5: Example 3 pushed result r(x,y)

5 Discussion

The described program has good performance if no plot is returned. The plot function pulls the performance down. The performance is boosted by the vector representation of the functions, which avoids recomputation of values. The inputs are given as functions, not vectors to simplify the usage for the tester. But essentially the program could also be used with only function vector input. In the program some methods have a double occurence, because different solver need the same function. This design decision is made, such that the code improves in readability. Because of the listing of the same function in different modules, those modules don't need to interact with each other. This boosts the clarity of the program as a whole, what was set as a design goal.

6 Summary

The IGA uses a Galerkin method similar to the FEM, but with a domain transformation. This way the actual evaluation is done in an fixed and easier to compute domain. The code of the program is following the structure of the underlying maths. Which supports the interested reader in understanding the code after reading this paper. The solver modules are approached individually, so that the user can start with easy examples as given in example 1 or 2 and construct more complex examples like example 3.

7 Outlook

The cutdowns of the program could be eradicated, which could worsen the readability and representation of the code. A support for higher dimension could also be implemented. Other plot libraries or changing of plot parameters should be evaluated to potentially boost the performance. Implementing different basis functions, e.g. NURBS instead of B-Splines, is also conceivable. Additionally, parsing of inputs and outputs in CAD programs could be a possibility.

References

[1] C. De Falco, A. Reali, and R. VáZquez. Geopdes: A research tool for isogeometric analysis of pdes. *Adv. Eng. Softw.*, 42(12):1020–1034, December 2011.

[2] Eduardo M. Garau and Rafael Vzquez. Algorithms for the implementation of adaptive isogeometric methods using hierarchical b-splines. *Appl. Numer. Math.*, 123(C):58–87, January 2018.

[3] T.J.R. Hughes, J.A. Cottrell, and Y. Bazilevs. Isogeometric analysis: Cad, finite elements, nurbs, exact geometry and mesh refinement. *Computer Methods in Applied Mechanics and Engineering*, 194(39):4135 – 4195, 2005.

[4] R. Vázquez. A new design for the implementation of isogeometric analysis in octave and matlab. *Comput. Math. Appl.*, 72(3):523–554, August 2016.

Eigenständigkeitserklärung

Ich erkläre mit meiner Unterschrift, dass ich diese Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen benutzt habe. Alle Stellen dieser Arbeit, die dem Wortlaut, dem Sinn oder der Argumentation nach anderen Werken entnommen sind (einschließlich des *World Wide Web* und anderer elektronischer Text- und Datensammlungen), habe ich unter Angabe der Quellen vollständig kenntlich gemacht.

Ort, Datum Unterschrift