Finite Element Method — Part 1
Notes

Alexander Manov

January 16, 2019

Contents

1 Piecewise Polynomials in 1D. Interpolation, L_2-projection, A priori Error Estimates

We shall consider piecewise polynomials in 1D with the help of which we shall interpolate and we shall find the L_2-projection of a given function. Our main goal will be to see how these approximations are achieved, to see what kinds of errors we get and to compare them with the a priori estimates for each method. For those purposes we shall recall a few important things which we shall need.

Let $I : I = [0, l]$ split into subintervals $0 = x_0 < ... < x_n = l$, $I_i := [x_{i-1}, x_i], i = 1, 2, ..., n$, $h_i := x_i - x_{i-1}$. We define the so called "hat functions":

$$\varphi_i(x) = \begin{cases} \frac{x-x_{i-1}}{h}, & x \in I_i, \\ \frac{x_{i+1}-x}{h}, & x \in I_{i+1}, \\ 0, & \text{otherwise.} \end{cases}$$

As we know, these functions define a nodal basis in the space of continuous piecewise linear functions and therefore we can write any function from that space as a linear combination of the basis:

$$v(x) = \sum_{i=0}^{n} v(x_i)\varphi_i(x).$$

Then, the piecewise linear interpolant of a given function $f(x)$ can be defined as

$$u_I(x) = \sum_{i=0}^{n} f(x_i)\varphi_i(x).$$

For the L_2-projection of a given function $f(x)$ we have

$$P_h f = \sum_{j=0}^{n} q_j \varphi_j(x). \tag{1}$$

where the coefficients $\mathbf{q} = (q_0, q_1, ..., q_n)$ are found as a solution of the system

$$M\mathbf{q} = \mathbf{b}, \tag{2}$$

In the latter system, M is the mass matrix and $\mathbf{b}$ is the load vector (see the lecture notes). We shall illustrate the main ideas on the basis of a few problems. Let us first implement a function in the system Wolfram Mathematica which defines the nodal basis and which we shall use in the following examples.

```
ϕ[i_, nodes_, x_] := (
  res = 0;
  If[i == 1,
   res = Piecewise[
      {{(nodes[[i + 1]] - x)/(nodes[[i + 1]] - nodes[[i]]), x > nodes[[i]] && x ≤ nodes[[i + 1]]}}];
   , (*else*)
   If[i == Length[nodes],
    res = Piecewise[
       {{(x - nodes[[i - 1]])/(nodes[[i]] - nodes[[i - 1]]), x > nodes[[i - 1]] && x ≤ nodes[[i]]}}];
    , (*else*)
    res = Piecewise[
       {{(x - nodes[[i - 1]])/(nodes[[i]] - nodes[[i - 1]]), x > nodes[[i - 1]] && x ≤ nodes[[i]]},
        {(nodes[[i + 1]] - x)/(nodes[[i + 1]] - nodes[[i]]), x > nodes[[i]] && x ≤ nodes[[i + 1]]},
        {0, x ≤ nodes[[i - 1]] || x > nodes[[i + 1]]}}];
   ]];
  res
 );
```

1.1 Linear Interpolant

Problem 1. For the function $u(x) = 2x\ \sin(2\pi x) + 3$, using the nodal basis of P_1, over the interval $I = [0, 1]$:

(a) Find the linear interpolant of $u(x)$;

(b) Plot the function and its interpolant. Plot the error;

(c) Compute the L_2-norm and the a priori estimate of the error. Compare both results.

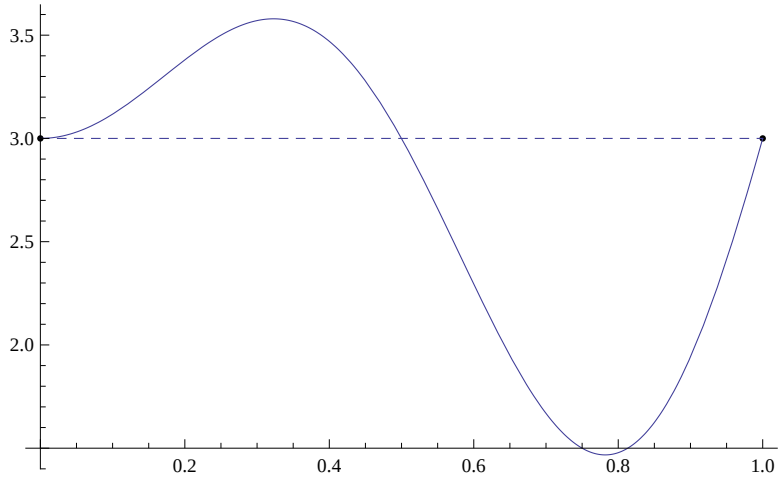
Solution: For solving this problem we shall use the system Wolfram Mathematica. The linear interpolant u_I in I is:

$$\begin{aligned} u_I &= u(0)\varphi_0 + u(1)\varphi_1 \\ &= 3 \cdot \frac{1-x}{1} + 3 \cdot \frac{x-0}{1} \\ &= 3. \end{aligned}$$

We can find and visualize u_I using Mathematica:

In[2]:=
```
func[x_] := 2 x Sin[2 π x] + 3.;
Nodes = {0, 1};
h = Nodes[[2]] - Nodes[[1]];
uIP1[x_] := Sum[func[Nodes[[i]]] ϕ[i, Nodes, x], {i, 1, Length[Nodes]}];
Show[{ListPlot[{{Nodes[[1]], func[Nodes[[1]]]},
    {Nodes[[2]], func[Nodes[[2]]]}}, PlotStyle → Black],
  Plot[func[x], {x, Nodes[[1]], Nodes[[2]]}],
  Plot[uIP1[x], {x, Nodes[[1]], Nodes[[2]]}, PlotStyle → Dashed]}]
```

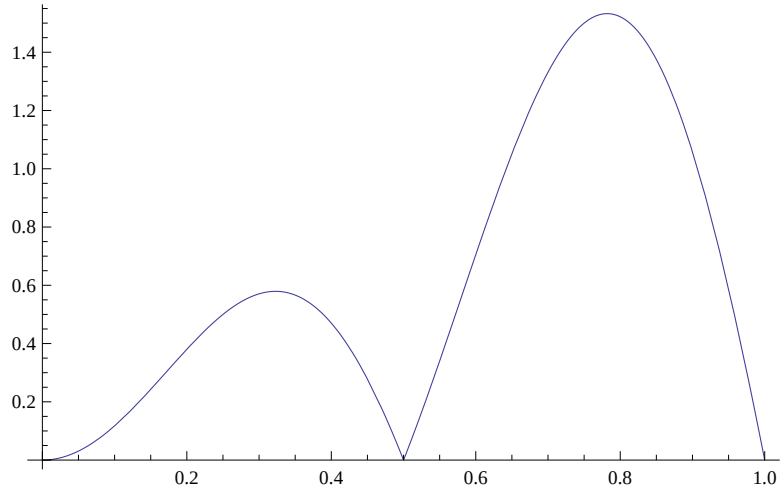
Out[6]=



For the error, we have:

In[7]:=
```
(*error*)
Plot[Abs[func[x] - uIP1[x]], {x, 0, 1}]
(*L2 norm of the error*)
actualErr =
 N[Sqrt[(NIntegrate[(func[x] - uIP1[x])^2, {x, Nodes[[1]], Nodes[[2]]}])]]
(*A priori error estimate*)
estimInterpErr =
 h^2 Sqrt[NIntegrate[D[func[x], {x, 2}]^2, {x, Nodes[[1]], Nodes[[2]]}]]
```

Out[7]=



Out[8]= `0.800835`

Out[9]= `38.3835`

In other words, the L_2 norm of the error is:

$$\|u(x) - u_I\|_{L_2[I]} = \left(\int_0^1 (u - u_I)^2 dx \right)^{\frac{1}{2}} = 0.800835,$$

and the a priori estimate is:

$$\|u - u_I\|_{L_2[I]} \leq Ch^2 \|u''\|_{L_2[I]}.$$

Using $C = 1$, we end up having:

$$\|u - u_I\|_{L_2[I]} \leq 38.3835.$$

We can see that the difference between the actual error and the estimate is quite significant. This is because the a priori estimate only gives us an upper limit for the error, not its actual value for the given function. Usually, the a priori estimates highly overestimate the error.

1.2 Piecewise Linear Interpolant

Now let us consider the piecewise linear interpolation.

Problem 2. For the function $u(x) = 2x\ \sin(2\pi x) + 3$, using the nodal basis of V_h, over the interval $I = [0, 1]$, discretized with n equidistant nodes:

(a) Find the piecewise linear interpolant, for $n = 6$ and $n = 51$;

(b) Plot the function and its interpolant. Plot the error;

(c) Compute the L_2-norm and the a priori estimate of the error. Compare both results.

Solution: We shall solve the problem with 6 nodes. With 51 nodes the whole process is analogical. We divide the interval $I = [0, 1]$ into 5 equal parts, each with length $h = 0.2$. The mesh nodes are $0 = x_0 < x_1 < ... < x_5 = 1$.

Using the nodal basis $\{\varphi_j\}_{j=0}^{5}$ for the space of continuous piecewise linear functions, we represent the piecewise linear interpolant in the following way:

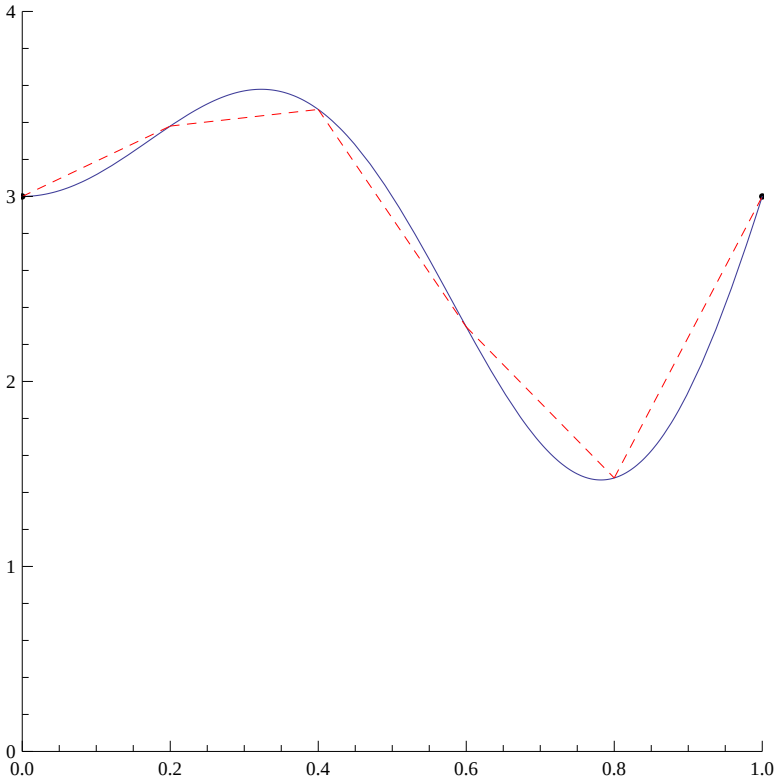
$$u_I = \sum_{j=0}^{5} u(x_j)\varphi_j$$

Now, we compute it, using Mathematica:

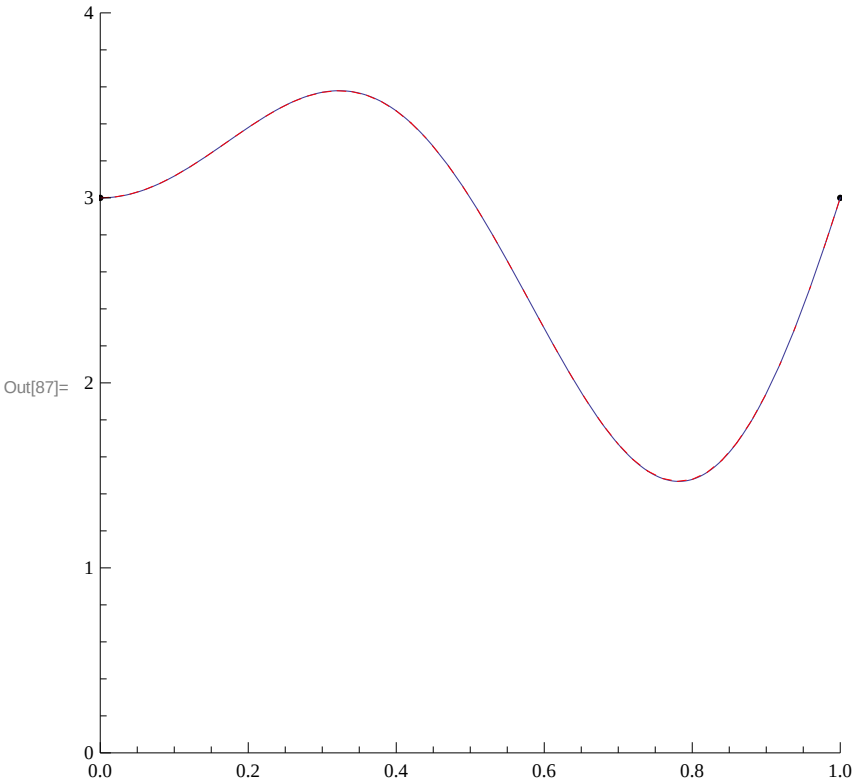
```
func[x_] := 2 x Sin[2 π x] + 3.;
n = 5;
a = 0.;
b = 1.;
h = (b - a)/n;
Nodes = Table[a + i h, {i, 0, n}];
uIVh[x_] := Sum[func[Nodes[[i]]] ϕ[i, Nodes, x], {i, 1, Length[Nodes]}]

Show[{ListPlot[{{a, func[a]}, {b, func[b]}},
    PlotStyle → Black], Plot[func[x], {x, a, b}],
  Plot[uIVh[x], {x, a, b},
   PlotStyle → {Red, Dashed, Thin}]},
 AspectRatio → 1, PlotRange → {{0, 1}, {0, 4}}]
```

Out[76]=



The result using 51 nodes is



We can see that as we increase the number of nodes, the two graphs (of the given function and the interpolant) become visually indistinguishable.

For the error, calculated in L_2 norm, we have:

$$e = \|u(x) - u_I\|_{L_2[I]} = \left(\int_0^1 (u - u_I)^2 dx \right)^{\frac{1}{2}}.$$

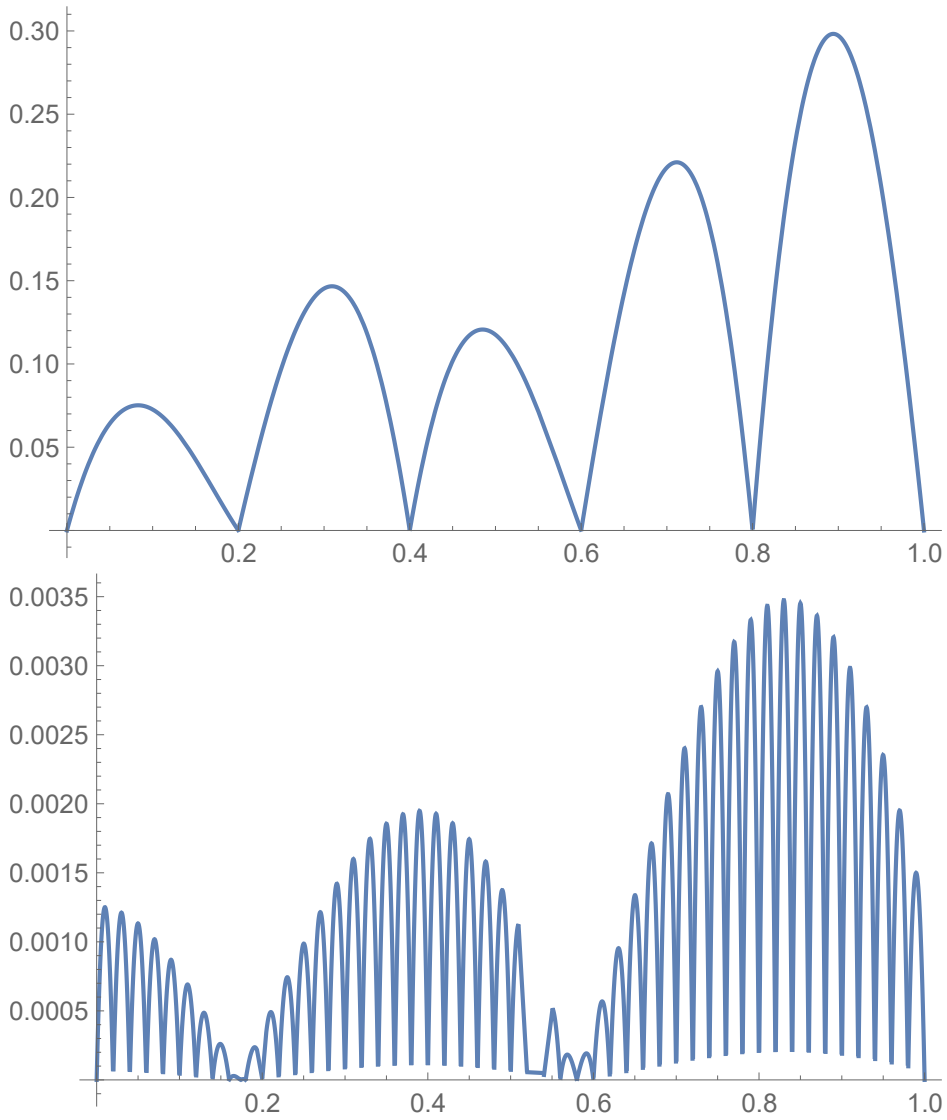
For $n = 6$, $e = 0.136611$ and for $n = 51$, $e = 0.00140114$ We see that as we decrease the step 10 times, the error decreases approximately 10^2 times, which is also reflected in the a priori estimate. We also see that in this case the error is much smaller than the one of the linear interpolant, which is our goal.

Now, let us compute the a priori error estimates for 6 and 51 nodes, respectively:

$$\|u - u_I\|_{L_2[I]} \leq Ch^2 \|u''\|_{L_2[I]}.$$

For $n = 6$ the error is limited by 1.53534 for $n = 6$ and for 0.0153534 for $n = 51$—0.0153534.

The graphs of the errors for 6 and 51 nodes, respectively, are depicted below:



1.3 L_2-Projection

Problem 3. For the function $u(x) = 2x\ \sin(2\pi x) + 3$ using the nodal basis φ_i over the interval $I = [0, 1]$, discretized with n equidistant nodes:

(a) Write a function that assembles the mass matrix M for given local matrix M_i and a function that assembles the load vector $\mathbf{b}$;

(b) Using

$$M_i = \frac{h}{6}\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

and

$$\mathbf{b}_i = \begin{bmatrix} \int_{x_{i-1}}^{x_i} u\varphi_{i-1}\mathrm{d}x \\ \\ \int_{x_{i-1}}^{x_i} u\varphi_i \mathrm{d}x \end{bmatrix}$$

assemble M and $\mathbf{b}$;

(c) Find the L_2-projection of the function for $n = 6$ and $n = 51$;

(d) Plot the L_2-projection of $u(x)$ and the approximation error;

(e) Compare the error and the a priori estimate for the L_2-projection.

Solution. In order to find the L_2 projection of the function u, we shall implement 2 functions in Mathematica that will help us assemble the global mass matrix and the load vector. Those functions must accept a local matrix (respectively vector) and use it to assemble the global one.

We do this, having in mind that in the future we shall need to assemble more global matrices for the finite element method. Having those functions implemented, we will only need to input the local matrices, in order to compute the global ones. One possible implementation is the following:

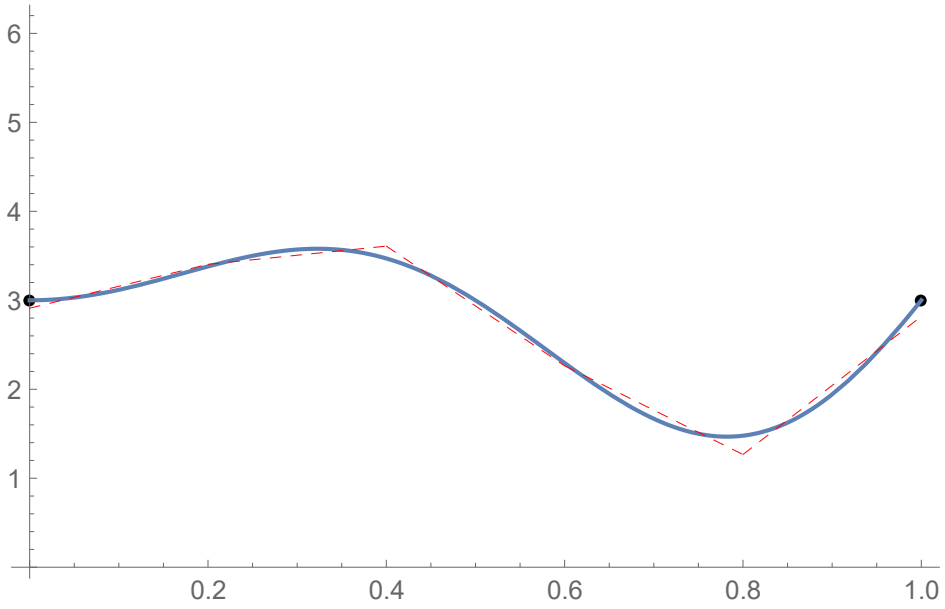
In[1]:=
```
(*L2 Projection*)
global[nodes_, Mloc_] := (
   M = Table[0, {i, Length[nodes]}, {j, Length[nodes]}];
   For[i = 2, i ≤ Length[nodes], i++,
    M[[i - 1, i - 1]] = M[[i - 1, i - 1]] + Mloc[i][[1, 1]];
    M[[i, i]] = M[[i, i]] + Mloc[i][[2, 2]];
    M[[i, i - 1]] = M[[i, i - 1]] + Mloc[i][[2, 1]];
    M[[i - 1, i]] = M[[i - 1, i]] + Mloc[i][[1, 2]];
   ];
   M
  );
load[f_, nodes_, bloc_] := (
   b = Table[0, {i, Length[nodes]}];
   For[i = 2, i ≤ Length[nodes], i++,
    b[[i - 1]] = b[[i - 1]] + bloc[f, i, nodes][[1]];
    b[[i]] = b[[i]] + bloc[f, i, nodes][[2]];
   ];
   b
  );
```

Using these two functions we can find the values $\{q_i\}_{i=0}^{n}$ in (1).

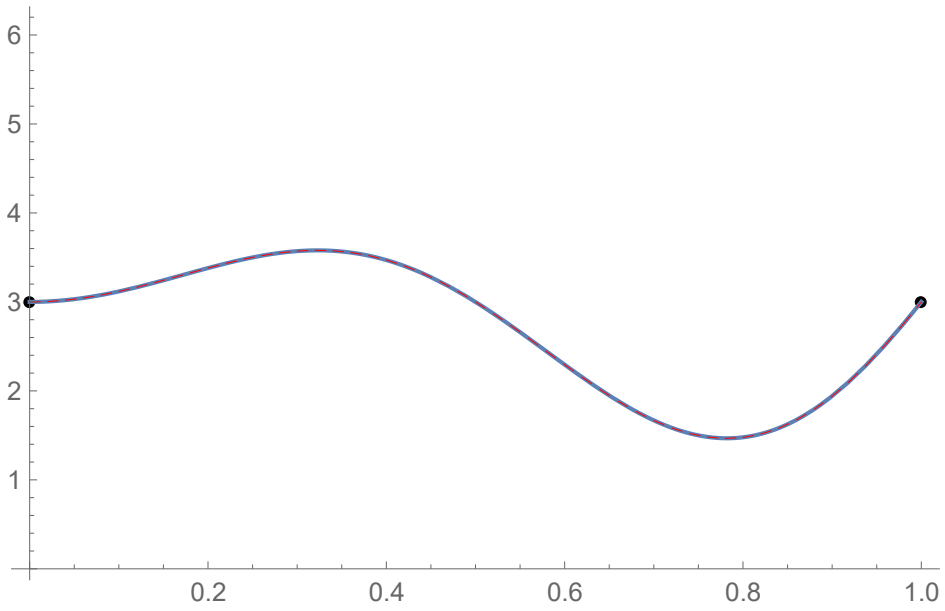
In[12]:=
```
Mloc[i_] := h/6 {{2, 1}, {1, 2}};
bloc[f_, i_, nodes_] :=
  {
   NIntegrate[f[x] ϕ[i - 1, nodes, x], {x, nodes[[i - 1]], nodes[[i]]}],
   NIntegrate[f[x] ϕ[i, nodes, x], {x, nodes[[i - 1]], nodes[[i]]}]
  };
M = global[newNodes, Mloc];
b = load[func, newNodes, bloc];
ξ = LinearSolve[M, b];
l2Proj = Sum[ϕ[i, newNodes, x] ξ[[i]], {i, 1, Length[newNodes]}];
Show[{ListPlot[{{Nodes[[1]], func[Nodes[[1]]]}, {Nodes[[2]], func[Nodes[[2]]]}},
    PlotStyle → Black],
  Plot[func[x], {x, Nodes[[1]], Nodes[[2]]}],
  Plot[l2Proj, {x, Nodes[[1]], Nodes[[2]]}, PlotStyle → {Red, Dashed, Thin}]}]
Plot[Abs[l2Proj - func[x]], {x, Nodes[[1]], Nodes[[2]]}, PlotRange → All]
l2Err = Sqrt[(NIntegrate[(func[x] - l2Proj)^2, {x, 0., 1.}])]
l2EstimErr = h^2 Sqrt[NIntegrate[D[func[x], {x, 2}]^2, {x, Nodes[[1]], Nodes[[2]]}]]
```

In the figures below, the piecewise linear function that gives the best approximation of u with respect to the L_2-norm is depicted in red:

- for $n = 6$:

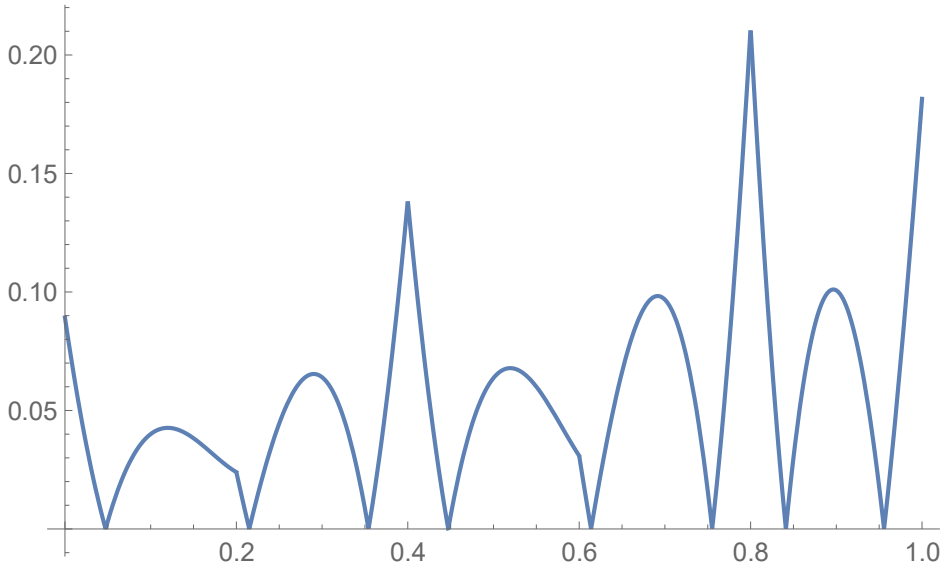


- for $n = 51$ the function u and its approximation are visually indistinguishable:

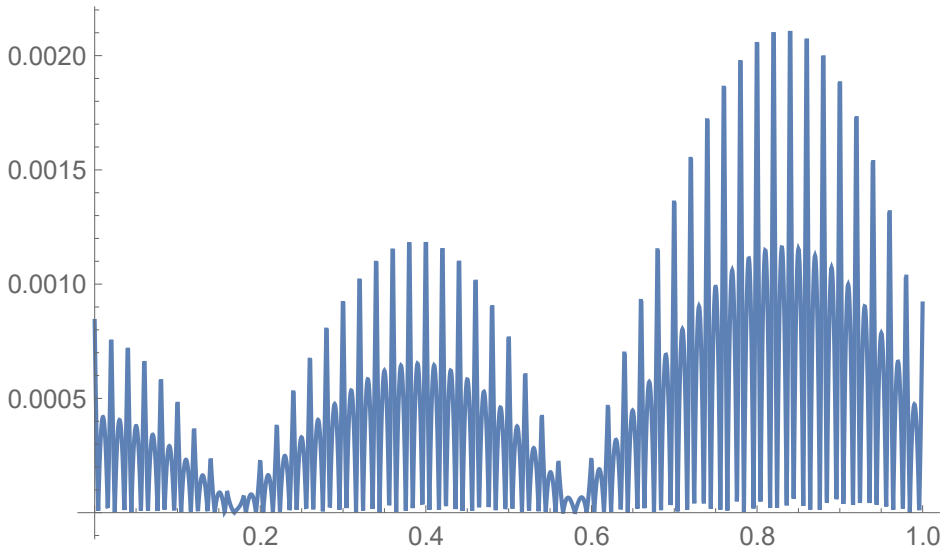


The graphs of the approximation error are:

- for $n = 6$:



- for $n = 51$:



The L_2-norm of the error for $n = 6$ is 0.0664705 and for $n = 51$ it is 0.000573338. The a priori error estimate:

$$\|u - u_I\|_{L_2[I]} \leq Ch^2 \|u''\|_{L_2[I]}$$

for $n = 6$ is 1.53534 and for $n = 51$ it is 0.0153534.

Let us remark that the error here is less than for the piecewise linear interpolant. This can be, of course, expected, since the L_2-projection gives the best approximation with respect to the L_2-norm.

We also note that where the function "bends" a lot (i.e., where the second derivative is large), both the piecewise linear interpolation, and the L_2-projection give relatively large errors. Of course if we introduce a better choice of the nodes, or we decrease h, then naturally the error also decreases and, theoretically, we can obtain as good an approximation as we need. In practice, of course, we are still limited by computational time and machine accuracy in floating-point arithmetics. □

2 FEM in 1D with homogeneous boundary conditions

In this section, we shall introduce the Finite Element Method in 1D with homogeneous boundary conditions over three examples. Each one will build up on the previous as we go over different right-hand side functions and different boundary conditions.

Before we begin with the examples, we shall introduce the Dirac delta function, as it will be used in one of the examples later. Dirac delta functions $\delta(x-a)$ is a special function that represents a point load or source. This function is 0 everywhere except the point a, where it is $+\infty$.

Definition 1. The Dirac delta function $\delta(x-a)$, defined over the real line, is

$$\delta(x-a) = \begin{cases} +\infty, & x = a, \\ 0, & x \neq a \end{cases}$$

with the property of

$$\int_{-\infty}^{+\infty} \delta(x-a)dx = 1.$$

Thus, from this definitions, it easily follows that

$$\int_{-\infty}^{+\infty} \delta(x-a)f(x)dx = f(a). \tag{3}$$

Now another thing we shall need before we begin is a mesh and elements. We will introduce a mesh with 4 points over the interval $[0,1]$ evenly distributed with $h = \frac{1}{3}$. The element will be linear and we shall use the "hat" functions from the previous chapter φ_i as basis. We are ready to begin with the first example.

Example 1. Let us have the following 2 point boundary-value problem:

$$\begin{cases} -u'' = 1, \\ u(0) = u'(1) = 0. \end{cases} \tag{D}$$

Our first step is to derive the weak form of (D). This is done by multiplying both sides of the equation by a test function v and then integrating over I:

$$-\int_0^1 u''v dx = \int_0^1 v dx.$$

Now, integrating by parts, on the left-hand side of the equation we get

$$\begin{aligned} -\int_0^1 u''v dx = -\int_0^1 v du' = -vu'\Big|_0^1 + \int_0^1 u'v'dx = \\ = \underbrace{-v(1)u'(1)}_{=0} + \underbrace{v(0)u'(0)}_{=0} + \int_0^1 u'v'dx = \int_0^1 u'v'dx. \end{aligned}$$

In order for $v(0)u'(0)$ to be 0, we need to impose additional requirement for the functions v, that is $v(0) = 0$. Thus the Hilbert space, in which we are looking for solutions is:

$$V := \{v \in H^1 : v(0) = 0\},$$

i.e. we have the following variational problem:

Find $u \in V$ such that:

$$a(u,v) = F(v), \forall v \in V, \tag{V}$$

where

$$\begin{aligned} a(u,v) &:= \int_0^1 u'v'dx, \\ F(v) &:= \int_0^1 v dx. \end{aligned}$$

From here we need to look for an approximation of the solution. We will look for it in a subspace of $V, V_h := span(\varphi_1, \ldots, \varphi_n)$. We formulate the following problem:

Find $u_h \in V_h \subset V$, such that

$$a(u_h, v) = F(v), \forall v \in V_h, \tag{R-G}$$

Now substituting

$$u_h = \sum_{j=1}^{n} q_j \varphi_j$$

we obtain the following form of (R-G)

$$a\left(\sum_{j=1}^{n} q_j \varphi_j, \varphi_i \right) = F(\varphi_i), \text{ for } i = 1, \ldots, n$$

The latter is a linear algebraic system of equations,

$$\mathbf{M}q = \mathbf{f},$$

where

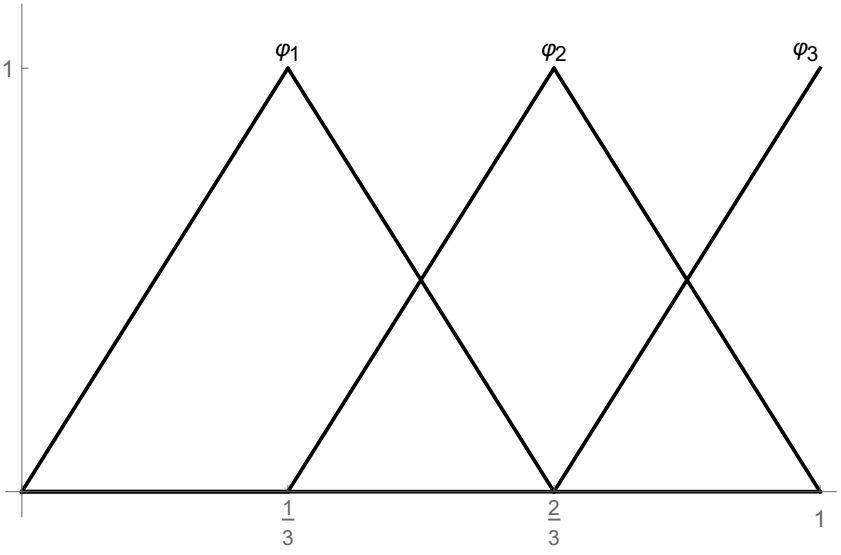
$$M_{ij} := \int_0^1 \varphi_i' \varphi_j' dx,$$
$$f_i := \int_0^1 \varphi_i dx.$$

Writing this in vector-matrix form, we have:

$$\begin{bmatrix} \int_0^1 \varphi_1'^2 dx & \int_0^1 \varphi_1' \varphi_1' dx & \ldots & \int_0^1 \varphi_1' \varphi_n' dx \\ \int_0^1 \varphi_1' \varphi_1' dx & \int_0^1 \varphi_1'^2 dx & \ldots & \int_0^1 \varphi_1' \varphi_n' dx \\ \vdots & \vdots & \ddots & \vdots \\ \int_0^1 \varphi_n' \varphi_1' dx & \int_0^1 \varphi_n' \varphi_1' dx & \ldots & \int_0^1 \varphi_n'^2 dx \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_n \end{bmatrix} = \begin{bmatrix} \int_0^1 \varphi_1 dx \\ \int_0^1 \varphi_1 dx \\ \vdots \\ \int_0^1 \varphi_n dx \end{bmatrix}.$$

Taking into account that in our particular case $n = 4$ as well as the form of $\varphi_1, \varphi_2, \varphi_3$, depicted below:

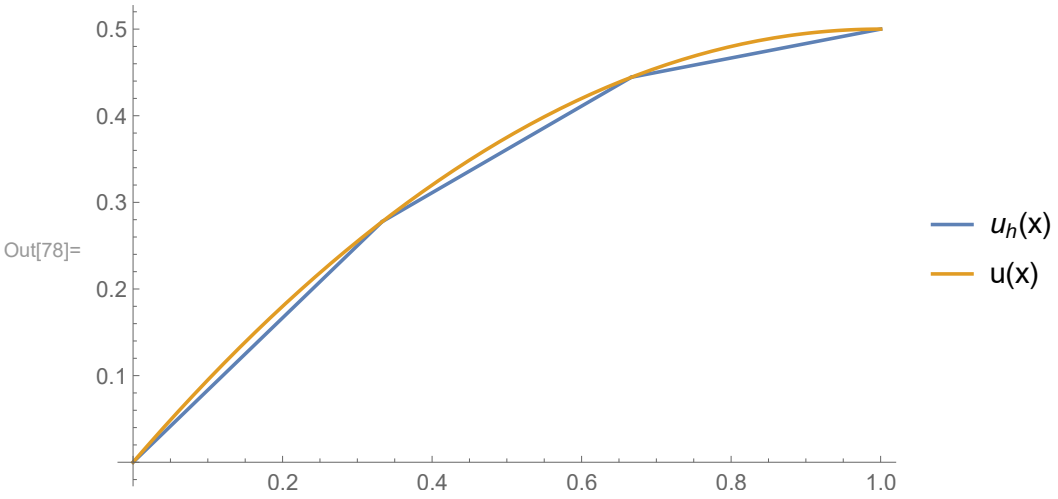
Out[20]=



after computing the integrals, we obtain:

$$\begin{bmatrix} 6 & -3 & 0 \\ -3 & 6 & -3 \\ 0 & -3 & -3 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} \begin{bmatrix} 1/3 \\ 1/3 \\ 0 \end{bmatrix}.$$

Solving this we get (q_1, q_2, q_3). Now, to illustrate this solution we shall plot it along with the analytical solution of the problem, $u(x) = -\frac{x}{2} + x$.



With this we conclude the first example.

The next example will be very similar to the first one. We will use it to illustrate the ability of FEM to solve differential problems variable coefficients, including disconitnuous ones, point sources, etc.

Example 2. Let us have the following 2 point boundary problem:

$$\begin{cases} -\frac{d}{dx}(c(x)\frac{du}{dx}) = \delta(x - \frac{1}{2}), \\ u(0) = \frac{du}{dx}\Big|_{x=1} = 0, \end{cases} \tag{D}$$

where

$$c(x) = \begin{cases} 2, x < \frac{1}{3}, \\ 4, x \geq \frac{1}{3}. \end{cases}$$

Doing the same steps as in the previous example, we derive the variational problem, corresponding to (D):

Find $u \in V$ such that:

$$a(u, v) = F(v), \forall v \in V, \tag{V}$$

where

$$a(u,v) := \int_0^1 cu'v'dx,$$
$$F(v) := \int_0^1 \delta(x-\frac{1}{2})vdx.$$

Note that the space V here is the same as in the previous example. Also, we consider the same elements and mesh. We proceed by writing the linear algebraic system of equations, $\mathbf{Mq} = \mathbf{f}$, where

$$\mathbf{M} := \begin{bmatrix} \int_0^1 c\varphi_1'^2 dx & \int_0^1 c\varphi_1'\varphi_2' dx & \int_0^1 c\varphi_1'\varphi_3' dx \\ \int_0^1 c\varphi_2'\varphi_2' dx & \int_0^1 c\varphi_2'^2 dx & \int_0^1 c\varphi_2'\varphi_3' dx \\ \int_0^1 c\varphi_3'\varphi_1' dx & \int_0^1 c\varphi_3'\varphi_2' dx & \int_0^1 c\varphi_3'^2 dx \end{bmatrix},$$

$$\mathbf{f} := \begin{bmatrix} \int_0^1 \delta(x-\frac{1}{2})\varphi_1 dx \\ \int_0^1 \delta(x-\frac{1}{2})\varphi_2 dx \\ \int_0^1 \delta(x-\frac{1}{2})\varphi_3 dx \end{bmatrix}.$$

Using (3), for the load vector, we obtain

$$\mathbf{f} := \begin{bmatrix} \varphi_1(\frac{1}{2}) \\ \varphi_2(\frac{1}{2}) \\ \varphi_3(\frac{1}{2}) \end{bmatrix}.$$

Now let us take a look at the stiffnes matrix. The function $c(x)$ is discontinuous at the point $\frac{1}{3}$. But in our case, $\frac{1}{3}$ is a point of the mesh. We can compute the integral just by splitting it. We shall give here only the computation of M_{11}:

$$M_{11} = \int_0^1 c\varphi_1'^2 dx = \int_0^{1/3} 2\varphi_1'^2 dx + \int_{1/3}^{2/3} 4\varphi_1'^2 dx$$

From here on, the example goes on as the previous one.

It is important to note that FEM can approximate such problems, showing that it is more flexible than other approximation methods as long as our mesh is good enough. With this we conclude this chapter.

3 2D Finite elements

3.1 2D elements

In this section, we shall learn how to define a triangulation over a given region. For that purpose we shall again use the system Wolfram Mathematica[1]. Our first few tasks shall be

[1]In this and future sections a lot of the built in functions which we are going to use are available in Mathematica 11.0 or later. A lot of them will not run in earlier versions of the system. In order to run the code presented in the notes on an earlier version, those functions should be implemented in order to work.

to define the "hat functions" over a 2D region. We shall demonstratethe idea by considering an example function and computing the piecewise linear interpolant and the L_2-projection.

Most of the times in the future, when we construct a triangulation over a region, the result is going to be in the form of few matrices—$\mathbf{P}$ that stores the nodes (their coordinates), and $\mathbf{T}$ that stores the elements (the indices of the vertices of each element). Therefore, we shall now introduce a region given by those 2 matrices and plot it. Then we shall proceed with the implementation of the nodal basis.

We shall illustrate all this with the following problem:

Problem 4. By given $\mathbf{P}$ and $\mathbf{T}$:

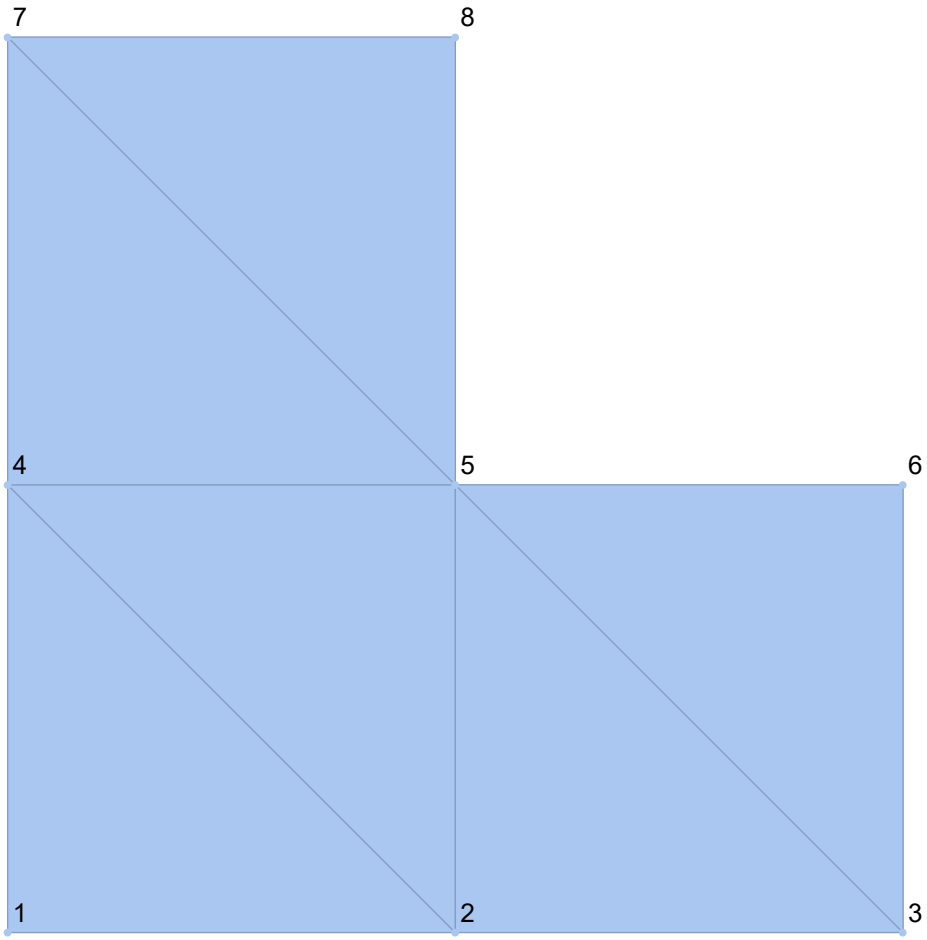
$$\mathbf{P} := \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 2 & 0 \\ 0 & 1 \\ 1 & 1 \\ 2 & 1 \\ 0 & 2 \\ 1 & 2 \end{bmatrix}, \mathbf{T} := \begin{bmatrix} 1 & 2 & 4 \\ 2 & 5 & 4 \\ 2 & 3 & 5 \\ 3 & 6 & 5 \\ 4 & 5 & 7 \\ 5 & 8 & 7 \end{bmatrix}$$

(a) Draw the region and its triangulation;

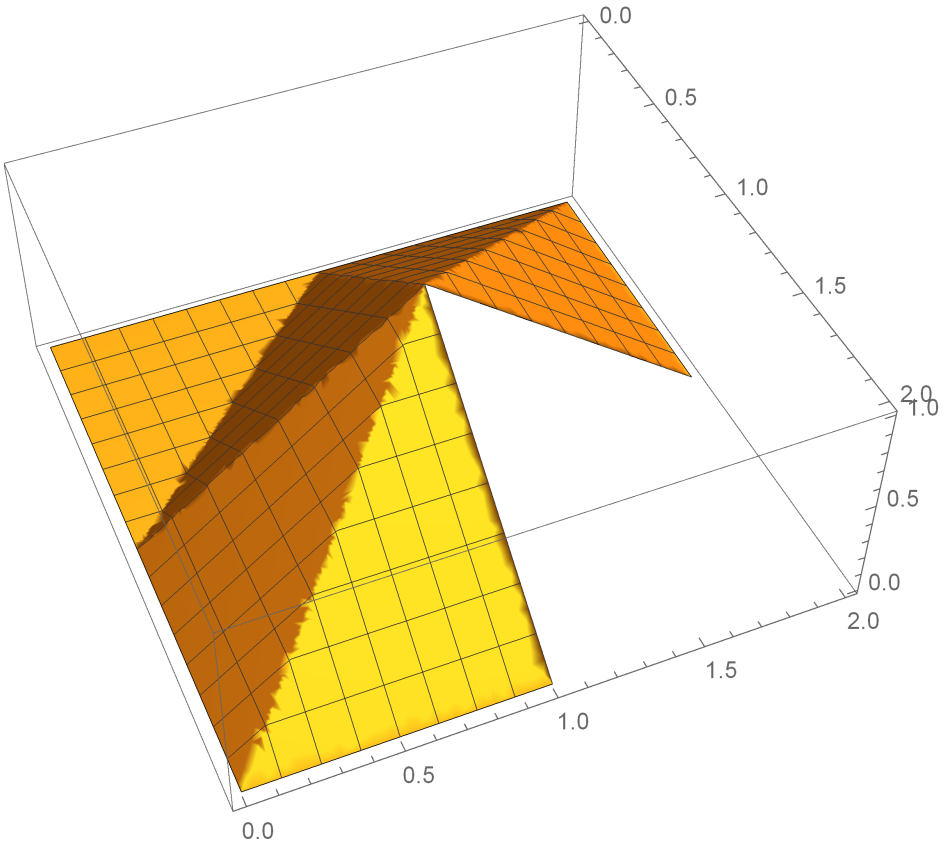
(b) Implement a function that computes the i-th "hat" function;

(c) Find the linear interpolant for the function $f(x, y) := x^2 + y^2$ using the nodal basis.

Solution. In order to draw the region, using the provided matrices, we shall use the "MeshRegion" function in Mathematica:



The next step is to implement the nodal basis functions. Analogously to the 1D case, 2D "hat" functions are linear in respect to both variables over each element. For example the function $\varphi_5(x, y)$ over our region looks like this:



As seen from the graph, a nodal basis function is has non-zero values only over the elements which contain the corresponding node. Furthermore, its value is 1 at that node and 0 in all the other nodes. Stated otherwise, if we restrict the function $\varphi_m(x, y)$ to element (P_m, P_k, P_l) that contains the m-th node, we are looking for $\varphi_m(x, y) = a + bx + cy$, such that

$$\begin{bmatrix} 1 & x_m & y_m \\ 1 & x_k & y_k \\ 1 & x_l & y_l \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}.$$

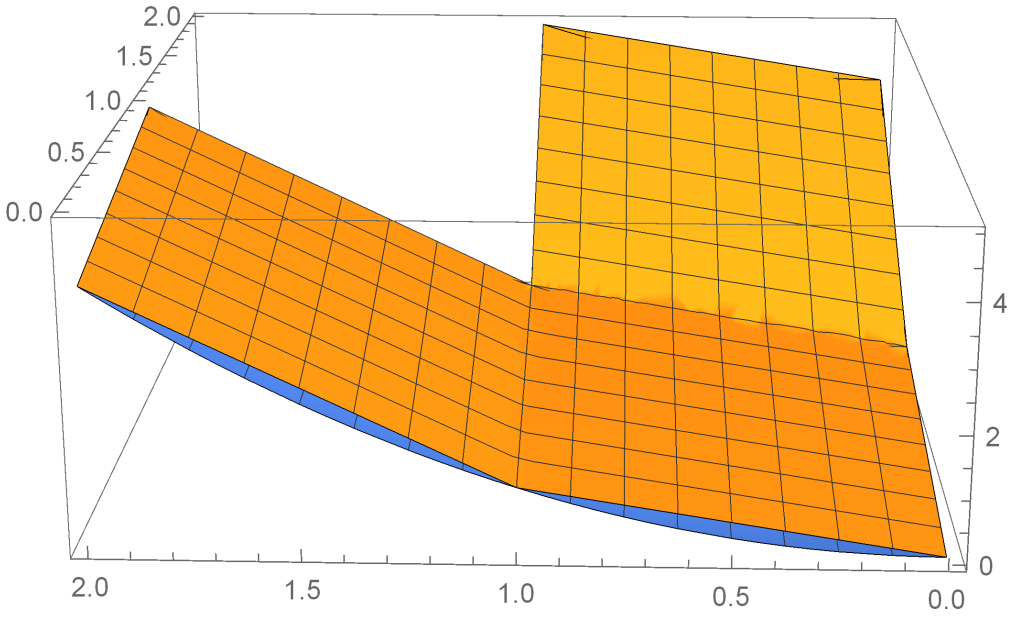
In Mathematica, an example implementation is:

```
ϕ[i_, x_, y_] := (
   result = 0;
   (*locate the element which contains {x,y}*)
   For[elementIndex = 1, elementIndex ≤ Length[T], elementIndex++,
    If[RegionMember[triangleRegions[[elementIndex]], {x, y}],
     (*stop if we find it*)
     Break[];
    ]
   ];
   (*check if i-th node is parth of the current element*)
   If[MemberQ[T[[elementIndex]], i],
    matr = Table[{1, P[[T[[elementIndex, j]], 1]], P[[T[[elementIndex, j]], 2]]}, {j, 1, 3}];
    right = {0, 0, 0};
    (*find the position of the i-th node in the elementIndex-th element.*)
    (*It is 1, 2 or 3. Set that index to 1 in the right hand side*)
    right[[Position[T[[elementIndex]], i][[1, 1]]]] = 1;
    sol = LinearSolve[matr, right];
    result = sol[[1]] + sol[[2]] x + sol[[3]] y
   ];
   result
  );
```

We have utilized a few built-in functions, in order to find out whether a specific point is within a given region, or not. For a given point (x, y), we locate the element which contains it. Then we solve the system above over this element.

The only thing left is to plot the linear interpolant of $f(x, y) = x^2 + y^2$. It is analogous to the 1D case. Over our region it looks as follows:

Out[91]=



□

3.2 Global matrix and load vector

Our next task will be to implement 2 functions which we shall use to assemble global matrices (e.g. stiffness or mass matrices). The functions will accept elements, nodes and a function that computes a local matrix. This way we can reuse the function for any mass or stiffness matrix for any problem we solve. The load vector function is analogous to the matrix one. The following code in Mathematica does exactly that:

```
In[92]:= assembleGlobalMatrix[elements_, nodes_, locMatr_] := (
           M = Table[0, {i, 1, Length[nodes]}, {j, 1, Length[nodes]}];
           For[i = 1, i ≤ Length[elements], i++,
            elem = nodes[[elements[[i]]]];

            For[j = 1, j ≤ 3, j++,
             For[k = 1, k ≤ 3, k++,
               M[[elements[[i, j]], elements[[i, k]]]] =
                 M[[elements[[i, j]], elements[[i, k]]]] + locMatr[elem][[j, k]];
              ];
            ];
           ];
           N[M]
          );
         assembleGlobalVector[elements_, nodes_, locVec_] := (
           F = Table[0, {i, 1, Length[nodes]}];

           For[i = 1, i ≤ Length[elements], i++,
            elem = nodes[[elements[[i]]]];
            For[j = 1, j ≤ 3, j++,
             F[[elements[[i, j]]]] = F[[elements[[i, j]]]] + locVec[elem][[j]];
            ];
           ];
           F
          );
```

To illustrate how the above functions work, we shall use them in an example. We shall find the L_2-projection of the function from the last problem—$f(x, y) = x^2 + y^2$. For that we need to implement functions that compute a local mass matrix and a local load vector.

In the case of the local load vector we shall make a linear transformation of the element into the standard triangularelement, in order to illustrate the use of the standard element and computing integrals over it.

In[1]:=
```
b[trigElem_] := {trigElem[[2, 2]] - trigElem[[3, 2]],
   trigElem[[3, 2]] - trigElem[[1, 2]],
   trigElem[[1, 2]] - trigElem[[2, 2]]};
c[trigElem_] := {trigElem[[3, 1]] - trigElem[[2, 1]],
   trigElem[[1, 1]] - trigElem[[3, 1]],
   trigElem[[2, 1]] - trigElem[[1, 1]]};
ψ[ξ_, η_] := {1 - ξ - η, ξ, η};
localMassMatrix[elem_] := (
   S = Area[Triangle[{elem[[1]], elem[[2]], elem[[3]]}]];
   (*S=Area[Polygon[Table[elem[[i]],{i,1,Length[elem]}]]];*)
   k0 = S/12 {{2, 1, 1}, {1, 2, 1}, {1, 1, 2}}
  );
localLoadVector[elem_] := (
   B = {{c[elem][[3]], -c[elem][[2]]}, {-b[elem][[3]], b[elem][[2]]}};
   J = N[Abs[Det[B]]];
   u[ξ_, η_] := (
     trans = {elem[[1, 1]], elem[[1, 2]]} + B.{ξ, η};
     f[trans[[1]], trans[[2]]]
    );
   (*Quadrature*)
   quad = Table[J 1/6 (ψ[0, 1/2][[i]] u[0, 1/2] + ψ[1/2, 1/2][[i]] u[1/2, 1/2] +
        ψ[1/2, 0][[i]] u[1/2, 0]), {i, 1, 3}];
   (*NIntegrate*)
   intr = J NIntegrate[ψ[ξ, η] u[ξ, η], {ξ, η} ∈ Triangle[{{0, 0}, {1, 0}, {0, 1}}]];
   quad
  );
```

With this we are ready to plot the L_2-projection:

```
q = LinearSolve[A, r];
L2[x_, y_] := Sum[q[[i]] ϕ[i, x, y], {i, 1, Length[P]}];
Plot3D[{L2[x, y], f[x, y]}, {x, y} ∈ region, PlotPoints → {30, 30}]
```