
A Minicourse on the Finite Element method

Gustavo C. Buscaglia

ICMC-USP, São Carlos, Brasil
gustavo.buscaglia@gmail.com

1 Motivation

What is the Finite Element Method?

- It is a family, or a collection, of numerical techniques used for the approximation of partial differential equations.
- **Finite Element approximations** are characterized in general by being based on **variational formulations** of the problem and discretized by a well defined subspace of the appropriate function spaces. They are usually defined by a so-called **Discrete Variational Formulation**.
- **Finite Element spaces** are also quite characteristic. They are based on a division of the domain of the problem into a finite number of pieces, or **subdomains**. Then **polynomial functions** are defined on each subdomain, building **piecewise polynomial spaces**. A transformation is used to define convergent approximations at curved boundaries
- Software used in industry for the simulation of **elliptic** and **parabolic** problems are usually **finite element codes**. Such problems include: **thermal conduction**, **electrostatics**, **magnetostatics**, **solid mechanics**, **low-inertia flows**, among others.

What are the strengths of the FEM?

- It is a **general** method, in the sense that it naturally deals with multi-dimensionality, with anisotropy, with geometrical complexity and with nonlinearity.
- Not all numerical methods accomplish that. **Finite difference methods** lose much of their appeal when the geometry is complex, or varying in time. **Boundary integral methods**, being based on Green's functions, cannot deal with even mild nonlinearities, at least not without major reformulation.

What can one actually **do** with Finite Elements?

That depends on your role:

- As a **software user**, you can explore all sorts of problems. The software has become so userfriendly that you hardly need to learn the theory at all... Perhaps out of curiosity? 😊
- As an **analyst**, you can program finite elements to have a quick approximate solution to some differential or optimality problem. Problems that can be formulated in terms of an **energy** are especially easy. This minicourse is mainly oriented towards this role.
- Finally, as a **developer**, you can work on open source projects of Finite Elements, implement new models as they are required by analysts of specific applications, develop algorithms for the many **open problems** in the field.

2 Finite element partitions

Let Ω be the **domain of definition** of a problem, for example the interval $(0, L)$ in 1D, or an arbitrary polygon in 2D, etc.

The FEM partitions Ω into a collection of subdomains K , where each K is the **image**, by a suitable transformation F_K , of a **unique** subdomain \hat{K} .

$$K = F_K(\hat{K}) \tag{2.1}$$

First example: $\Omega = (0, L)$

The **master subdomain** consists of:

- A **compact set**, the interval $\widehat{K} = [-1, 1]$.
- A **finite dimensional space** \widehat{G} , which we take as \mathbb{P}_1 , the polynomials of degree ≤ 1 .
- A **set of geometrical degrees of freedom** $\widehat{\Theta}$, which we take as the **values** at the **nodes** $\{-1, 1\}$.
- This defines two **geometrical basis functions** on \widehat{K} :

$$G_1(\hat{x}) = \frac{1}{2}(1 - \hat{x}); \quad G_2(\hat{x}) = \frac{1}{2}(1 + \hat{x}). \quad (2.2)$$

- Any linear polynomial p in \widehat{K} can be written as

$$p(\hat{x}) = c_1 G_1(\hat{x}) + c_2 G_2(\hat{x}). \quad (2.3)$$

- This is a **Lagrange basis**, i.e., the coefficients are nothing but the nodal values, $c_1 = p(-1)$, $c_2 = p(1)$, so that

$$p(\hat{x}) = p(-1) G_1(\hat{x}) + p(1) G_2(\hat{x}). \quad (2.4)$$

The **transformation**

$$F_K(\hat{x}) = a G_1(\hat{x}) + b G_2(\hat{x}). \quad (2.5)$$

maps \widehat{K} onto any interval $K = [a, b]$. Check that $F_K(-1) = a$ and $F_K(1) = b$. Positions are linearly interpolated in between.

```

function [g dg d2g] = masterp1(id,x)

n=max(size(x));
for i=1:n
if (id == 1)
g(1,i) = 0.5*(1-x(i));
g(2,i) = 1-g(1,i);

dg(1,i) = -0.5;
dg(2,i) = 0.5;
d2g(1,i) = 0;
d2g(2,i) = 0;
end
end
end

```

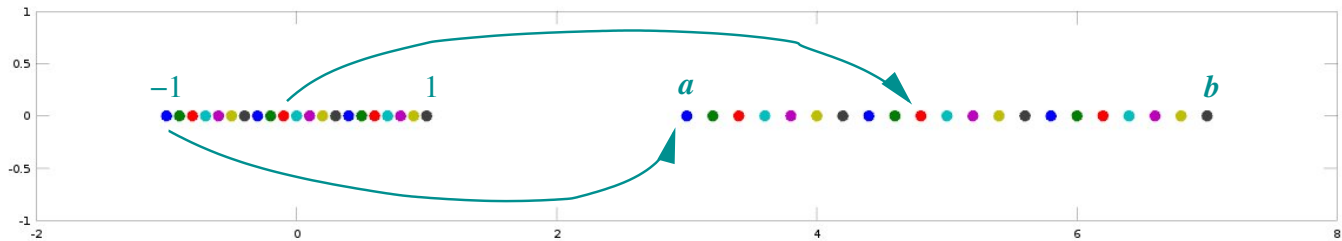
exa1.m

```

xhat=-1:0.1:1;
[g dg d2g] = masterp1(1,xhat);
a=3;b=7;

x=a*g(1,:)+b*g(2,:);
plot(xhat,0,x,0)

```



A **mesh** is composed of:

- A **connectivity matrix**.
- An **array of coordinates**.

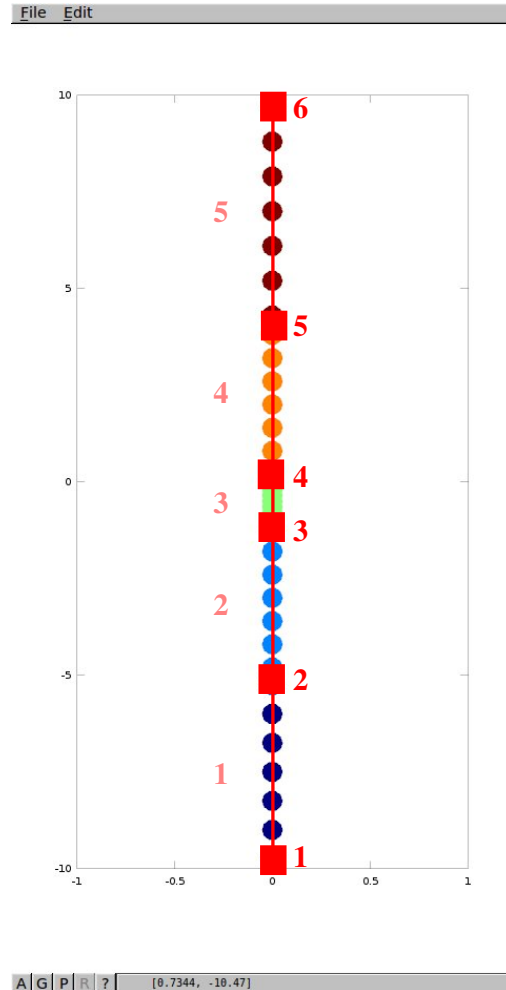
exa2.m

```
conec=[1 2;2 3;3 4;4 5;5 6];
coord=[-10 -5 -1 0 4 10];
nel=size(conec,1); npe=size(conec,2);
nod=size(coord,2);
xhat=-0.9:0.3:0.9; nhat=size(xhat,2);
zhat=zeros(nhat,1);
figure 1;

for k=1:nel

[g dg d2g] = masterp1(1,xhat);
a=coord(conec(k,1));b=coord(conec(k,2));
x=a*g(1,:)+b*g(2,:);
c=k;
scatter(zhat,x,20,c,"filled")
hold on;

end
```



The mesh concept **generalizes easily** to different geometries and topologies.

exa3.m

```

conec1=[1 2;2 3;3 4;4 5;5 6];
coord1=[-10 -5 -1 0 0 0;
        0 0 0 0 4 10];
conec2=[1 2;2 3;3 4;4 5;5 6;6 7;7 8;8 1];
coord2=[-10 -5 -2 -2 -2 -5 -10 -10;
        3 3 3 6 9 9 9 6];
nel1=size(conec1,1);npe=size(conec1,2);
nod1=size(coord1,2);
nel2=size(conec2,1);npe=size(conec2,2);
nod2=size(coord2,2);

xhat=-0.9:0.3:0.9;nhat=size(xhat,2);
[g dg d2g] = masterp1(1,xhat);

figure 1;

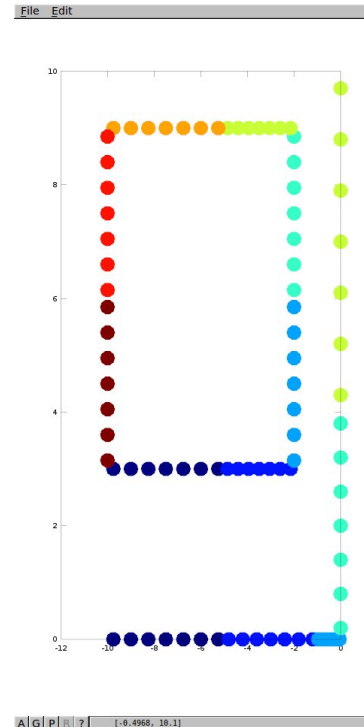
for k=1:nel1
a=coord1(:,conec1(k,1));b=coord1(:,conec1(k,2));
x(1,:)=a(1)*g(1,:)+b(1)*g(2,:);
x(2,:)=a(2)*g(1,:)+b(2)*g(2,:); c=k;
scatter(x(1,:),x(2,:),20,c,"filled")
hold on;

```

```

end
for k=1:nel2
a=coord2(:,conec2(k,1));b=coord2(:,conec2(k,2));
x(1,:)=a(1)*g(1,:)+b(1)*g(2,:);
x(2,:)=a(2)*g(1,:)+b(2)*g(2,:); c=k;
scatter(x(1,:),x(2,:),20,c,"filled")
hold on;
end

```



Disclaimer: Why Octave?

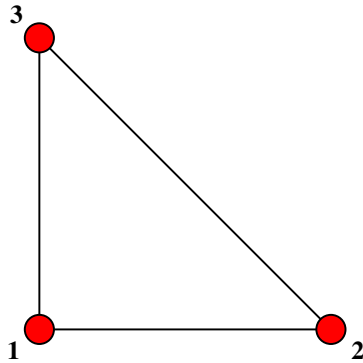
I am giving you examples in Octave. They should also work in Matlab. The reasons for this are:

- I want to **show a complete implementation** of the concepts. Show that it is **simple** and that it **works**.
- Octave is not the best-performing Language for most applications, but **it can do amazing things!**. With some FE concepts and Octave, one can easily solve quite sophisticated models in several applications. 😊
- If you know what to do in Octave but your application exceeds Octave's capabilities, all you need to do is to look for more efficient software packages and replace the Octave functions by the appropriate counterparts.

So, try to understand the Octave coding, the codes are available at my site and the package is free software. It will already allow you to compute interesting things, and if you need more performance the remedies are not hard to find.

Second example: A two-dimensional polygon

- We take the **master subdomain** as the unit triangle $\hat{K} = \{x \in \mathbb{R}^2, 0 \leq x_1 \leq 1, 0 \leq x_2 \leq 1 - x_1\}$.



- The **geometrical basis functions**

$$G_1(\hat{x}) = 1 - \hat{x}_1 - \hat{x}_2 \quad (2.6)$$

$$G_2(\hat{x}) = \hat{x}_1 \quad (2.7)$$

$$G_3(\hat{x}) = \hat{x}_2 \quad (2.8)$$

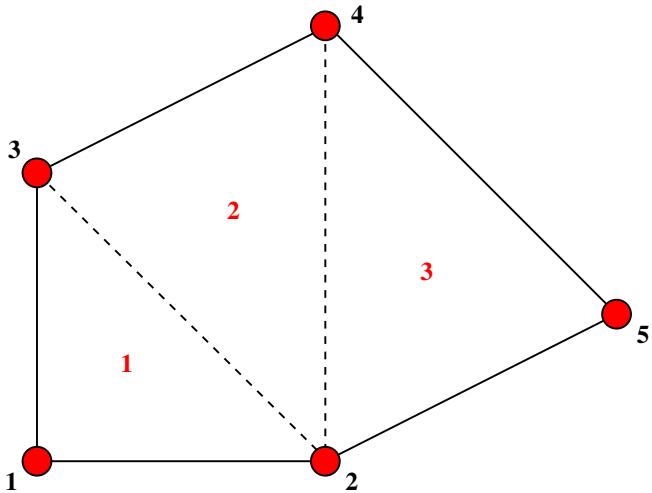
```
function [g dg d2g] = masterp1(id,x)
n=size(x,2);
for i=1:n
if (id == 1)
g(1,i) = 0.5*(1-x(1,i));
g(2,i) = 1-g(1,i);
```

```
dg(1,i) = -0.5;
dg(2,i) = 0.5;
d2g(1,i) = 0;
d2g(2,i) = 0;
elseif (id == 2)
g(1,i) = 1-x(1,i)-x(2,i);
g(2,i) = x(1,i);
g(3,i) = x(2,i);
dg(1,1,i) = -1;
dg(1,2,i) = -1;
dg(2,1,i) = 1;
dg(2,2,i) = 0;
dg(3,1,i) = 0;
dg(3,2,i) = 1;
d2g(1:3,1:2,i)=zeros(3,2);
end
end
end
```

- The **transformation**

$$F_K(\hat{x}) = \sum_{i=1}^3 \mathbf{X}^i G_i(\hat{x}) \quad (2.9)$$

which transforms \hat{K} into the arbitrary triangle with vertices \mathbf{X}^1 , \mathbf{X}^2 and \mathbf{X}^3 .

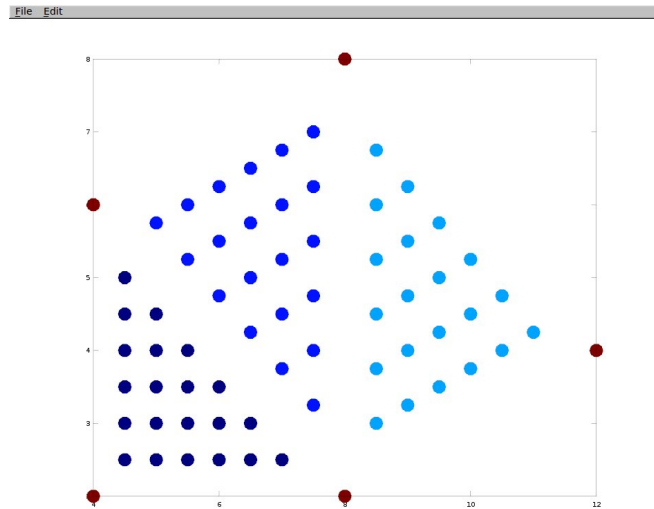
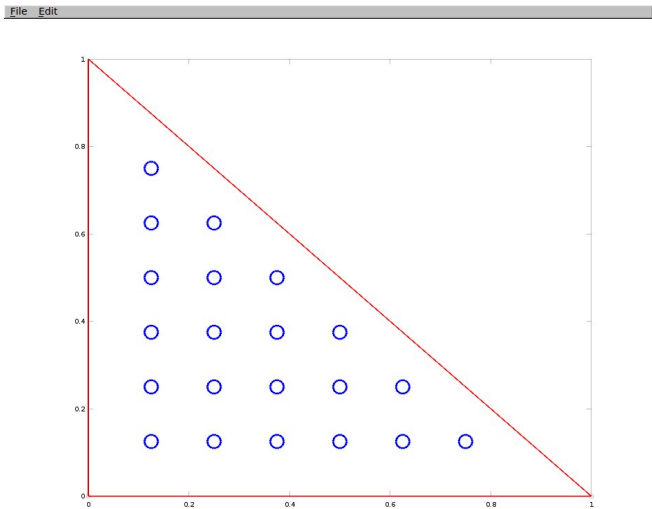


Mesh and transformation ([exa4.m](#))

```

conec=[1 2 3;3 2 4;4 2 5];
coord=[4 8 4 8 12;
       2 2 6 8 4 ];
for k=1:nel
    XX(:,1)=coord(:,conec(k,1));
    XX(:,2)=coord(:,conec(k,2));
    XX(:,3)=coord(:,conec(k,3));
    x=zeros(2,nhat);
    for i=1:npe
        x(1,:)=x(1,:)+XX(1,i)*g(i,:);
        x(2,:)=x(2,:)+XX(2,i)*g(i,:);
    end
end
end

```

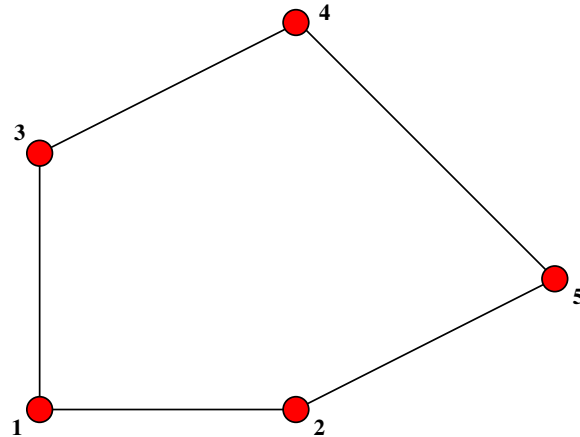


Mesh generation: GMesh

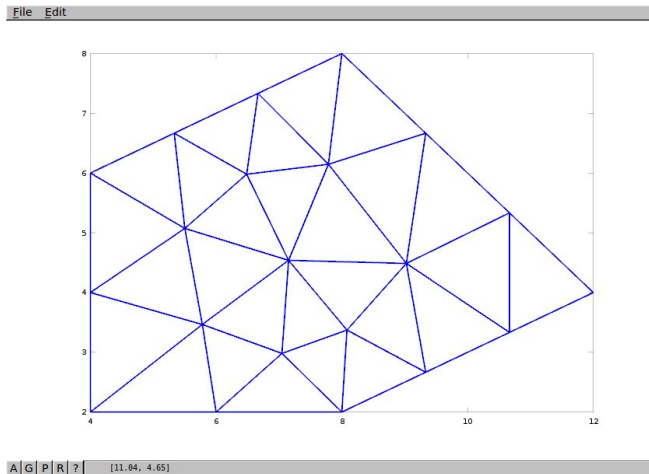
Software package for mesh generation. Two steps: (1) Define the geometry. (2) Generate mesh.

```
function [mesh, GMSH_OUT]=CreateMesh(r,ip)
    name=[tmpnam ".geo"];
    fid=fopen (name, "w");
    % POINTS
    string=sprintf("Point(1)={4,2,0,%f};\n",r);
    fputs (fid, string);
    string=sprintf("Point(2)={8,2,0,%f};\n",r);
    fputs (fid, string);
    string=sprintf("Point(3)={4,6,0,%f};\n",r);
    fputs (fid, string);
    string=sprintf("Point(4)={8,8,0,%f};\n",r);
    fputs (fid, string);
    string=sprintf("Point(5)={12,4,0,%f};\n",r);
    fputs (fid, string);
    % LINES joining the POINTS
    fputs (fid, "Line(1) = {1, 2};\n");
    fputs (fid, "Line(2) = {2, 5};\n");
    fputs (fid, "Line(3) = {5, 4};\n");
    fputs (fid, "Line(4) = {4, 3};\n");
    fputs (fid, "Line(5) = {3, 1};\n");
```

```
% SURFACES grouping the LINES
fputs(fid,"Line Loop(6)={1,2,3,4,5};\n");
fputs (fid, "Plane Surface(7) = {6};\n");
fclose (fid);
% MESH and glue all SURFACES
[mesh GMSH_OUT] = msh2m_gmsh...
(canonicalize_file_name(name)(1:end-4),...
"clscale", "1.0");
if(ip == 1)
    trimesh(mesh.t(1:3,:)',mesh.p(1,:)',...
    mesh.p(2,:)'');
end
% Delete auxiliary files
unlink (canonicalize_file_name (name));
end
```



```
[mm idum]=CreateMesh(2,1)
```



```
mm.p → coordinates
```

```
octave:> mm.p'
```

```
ans =  
  4.0000    2.0000  
  8.0000    2.0000  
  4.0000    6.0000  
  8.0000    8.0000  
 12.0000    4.0000  
  6.0000    2.0000  
  9.3333    2.6667  
 10.6667    3.3333  
 10.6667    5.3333  
etcetera
```

```
mm.t → connectivity
```

```
octave:> mm.t'
```

```
ans =  
   1   19   13    7  
  14   15   17    7  
   7    8   15    7  
   6   19    1    7  
   8    9   15    7  
  10   17   15    7  
etcetera
```

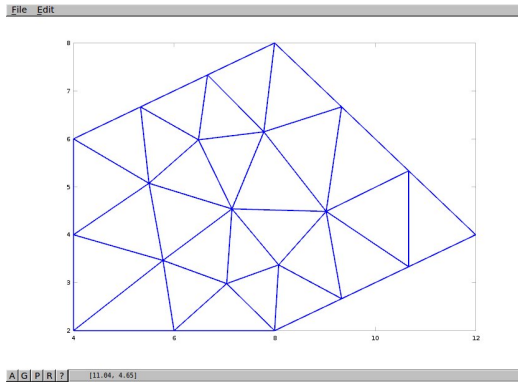
```
mm.e → boundary edges
```

```
octave:> mm.e'
```

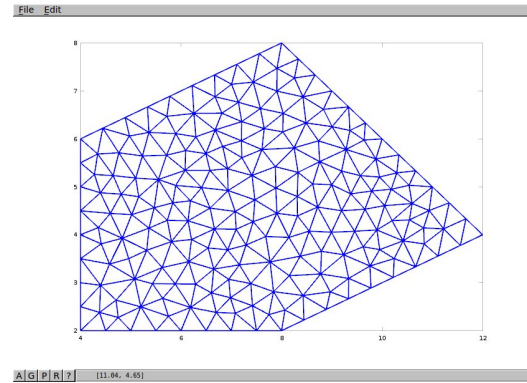
```
ans =  
   1    6    0    0    1    7    0  
   6    2    0    0    1    7    0  
   2    7    0    0    2    7    0  
   7    8    0    0    2    7    0  
   8    5    0    0    2    7    0  
   5    9    0    0    3    7    0  
   9   10    0    0    3    7    0  
  10    4    0    0    3    7    0  
   4   11    0    0    4    7    0  
  11   12    0    0    4    7    0  
  12    3    0    0    4    7    0  
   3   13    0    0    5    7    0  
  13    1    0    0    5    7    0
```

Meshes of arbitrary refinement can be generated on general geometries 1D/2D/3D.

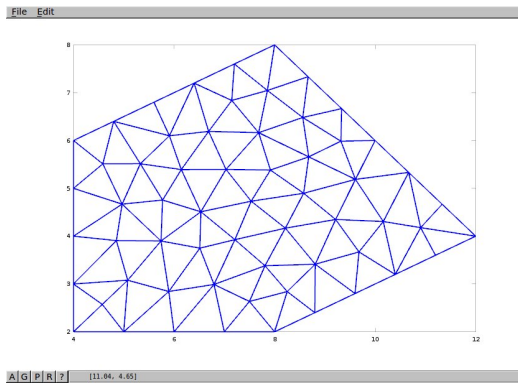
```
[mmm idum]=CreateMesh(2,1)
```



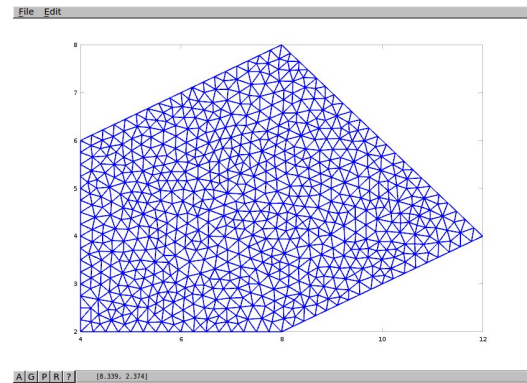
```
[mmm idum]=CreateMesh(0.5,1)
```



```
[mmm idum]=CreateMesh(1,1)
```

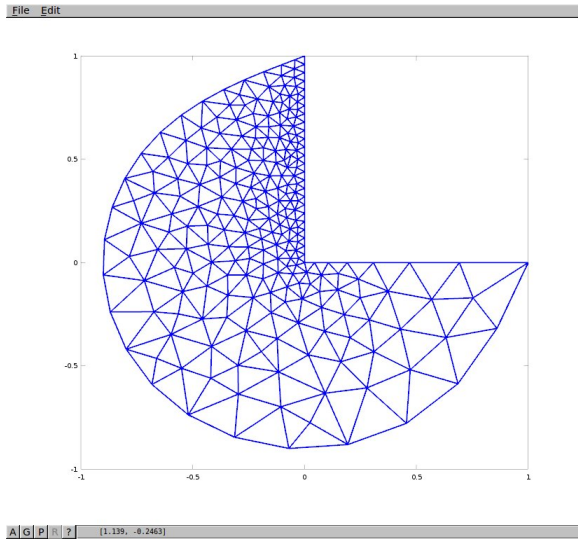


```
[mmm idum]=CreateMesh(0.25,1)
```

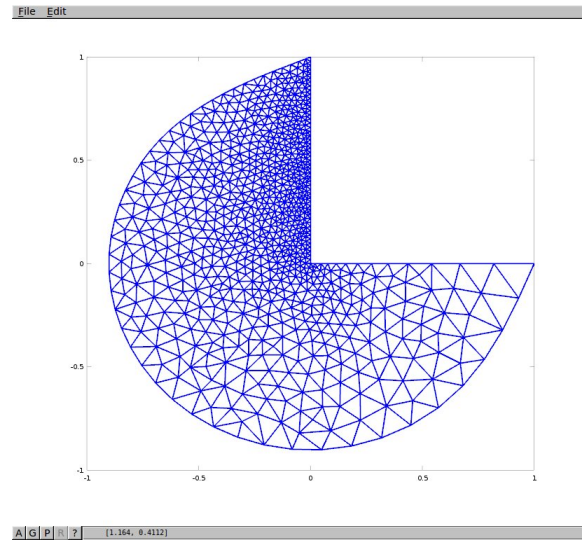


An example using **localized refinement** and **bsplines**.

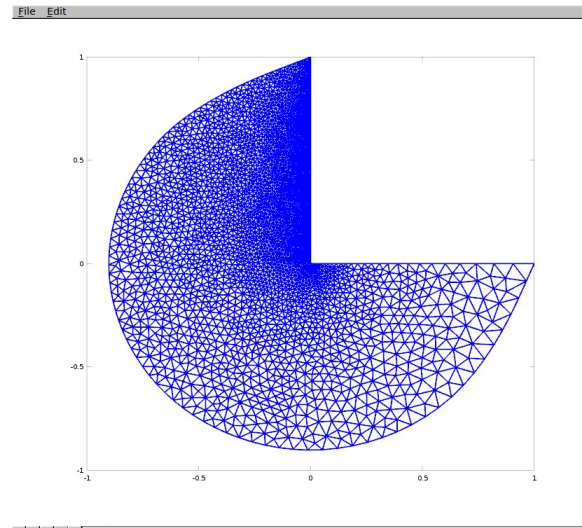
CreateMeshhexa5(0.4,1)



CreateMeshhexa5(0.2,1)



CreateMeshhexa5(0.1,1)



Takeaway

The three ingredients:

- **Master subdomain**
- **Geometrical basis functions**
- **Mesh** (connectivity + coordinates)

allow for the easy construction of transformations

$$F_K(\hat{x}) = \sum_i \mathbf{X}^i G_i(\hat{x})$$

so that each point of Ω is the image of **one point** \hat{x} for the **one subdomain** K to which it belongs (except at subdomain edges).

3 Finite element spaces

3.1 The master element

The **master element** (simplified version) consists of

- A **master subdomain**, i.e., \widehat{K} equipped with its **geometrical basis functions** $\{G_i\}_{i=1}^{n_g}$.
- A set of **shape functions** (basis functions) $\{\widehat{N}_j\}_{j=1}^{n_s}$.

The linear combinations of the shape functions define a vector space of functions of \hat{x} :

$$\widehat{P} = \{p : \widehat{K} \rightarrow \mathbb{R} \mid p(\hat{x}) = \sum_{i=1}^{n_s} \sigma_i \widehat{N}_i(\hat{x}), \text{ where } \sigma_i \in \mathbb{R}, \forall i\} . \quad (3.1)$$

The element is **isoparametric** if $G_i = \widehat{N}_i$.

Example 1: Lagrange P_p element in 1D

- Set $\hat{K} = [-1, 1]$.
- The geometrical basis functions are, as before:

$$G_1(\hat{x}) = \frac{1}{2}(1 - \hat{x}); \quad G_2(\hat{x}) = \frac{1}{2}(1 + \hat{x}) \quad (3.2)$$

- Let $\{\hat{X}^i\}_{i=1}^{p+1}$ be evenly distributed points with $\hat{X}^1 = -1$ and $\hat{X}^{p+1} = 1$. These will be the **nodes** of \hat{K} . The integer p is the **degree** of the element.
- The **shape functions** are the Lagrange polynomials of $\hat{X}^1, \dots, \hat{X}^{p+1}$.

$$\hat{N}_i(\hat{x}) = \prod_{j \neq i=1}^{p+1} \frac{\hat{x} - \hat{X}^j}{\hat{X}^i - \hat{X}^j} \quad (3.3)$$

- Notice that if $p = 1$ then $\hat{N}_1 = G_1$ and $\hat{N}_2 = G_2$.

The following code computes the shape functions of the Lagrange P_p element for all the \hat{x} -values given in the vector `xhat`.

masterlag.m

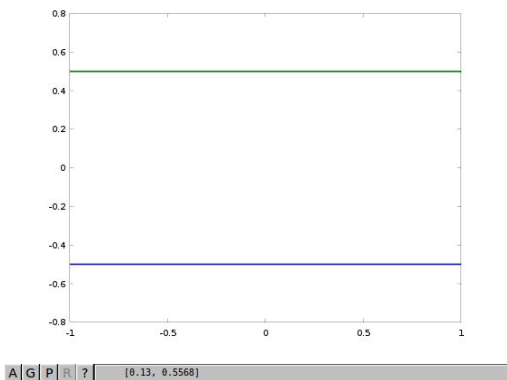
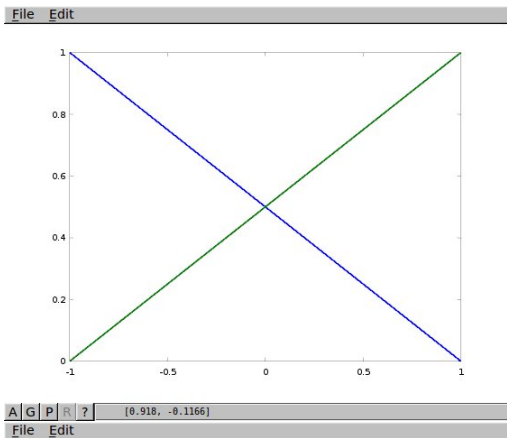
```
function sf=masterlag(p,xhat)
#Lagrange basis
nh=max(size(xhat));
d=2; #master is [-1,1]
n=p+1; #number of nodes
dx=d/p; #distance between nodes
xno(1:n)=(-1:dx:1);
for ih=1:nh
    xh=xhat(ih);
    for i=1:n
        aux=1;
        for j=1:n
            if (j!=i)
                aux=aux*(xh.-xno(j))/(xno(i)-xno(j));
            end
        end
        sf(i,ih)=aux;
    end
end
```

Actually, we will need not just the value of $\widehat{N}_i(\hat{x})$ but also those of $\widehat{N}'_i(\hat{x})$ and possibly of $\widehat{N}''_i(\hat{x})$. We can do the derivatives by hand or **differentiate the code** as follows:

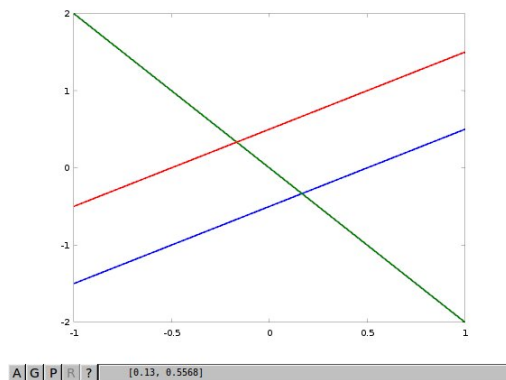
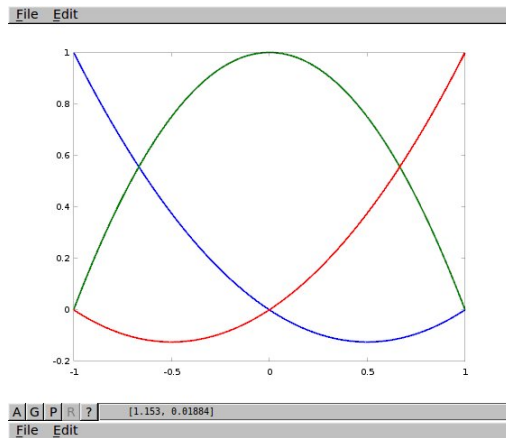
masterlag.m

```
function [sf dsf]=masterlag(p,xhat)
#Lagrange basis
nh=max(size(xhat));
d=2; #master is [-1,1]
n=p+1; #number of nodes
dx=d/p; #distance between nodes
xno(1:n)=(-1:dx:1);
for ih=1:nh
  xh=xhat(ih);
  for i=1:n
    daux=0;#derivative of next line
    aux=1;
    for j=1:n
      if (j!=i)
        daux=aux/(xno(i)-xno(j))+daux*(xh-xno(j))/(xno(i)-xno(j));#deriv. of next line
        aux=aux*(xh.-xno(j))/(xno(i)-xno(j));
      end
    end
    dsf(i,ih)=daux;#deriv. of next line
    sf(i,ih)=aux;
  end
end
```

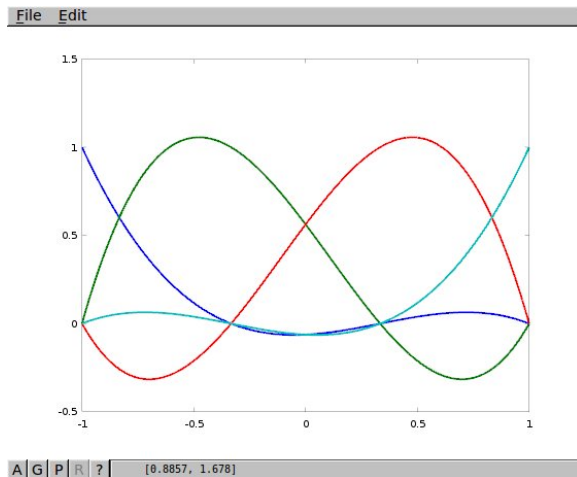
```
octave:> xh=-1:0.01:1;
octave:> pp=1;
octave:> [sf dsf]=masterlag(pp,xh);
octave:> plot(xh,sf,"linewidth",2)
```



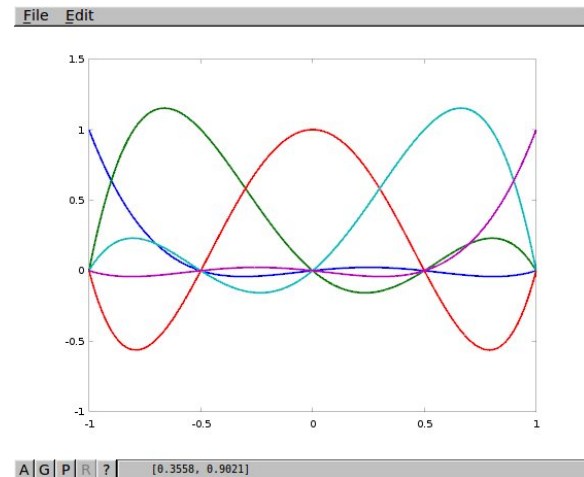
```
octave:> xh=-1:0.01:1;
octave:> pp=2;
octave:> [sf dsf]=masterlag(pp,xh);
octave:> plot(xh,sf,"linewidth",2)
```



```
octave:> xh=-1:0.01:1;
octave:> pp=3;
octave:> [sf dsf]=masterlag(pp,xh);
octave:> plot(xh,sf,"linewidth",2)
```



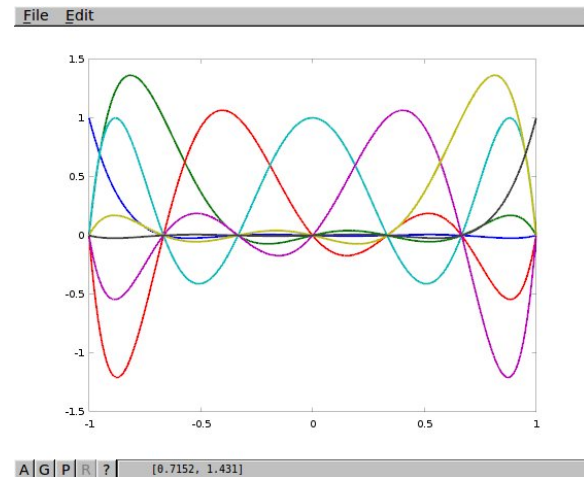
```
octave:> xh=-1:0.01:1;
octave:> pp=4;
octave:> [sf dsf]=masterlag(pp,xh);
octave:> plot(xh,sf,"linewidth",2)
```



```
octave:> xh=-1:0.01:1;
octave:> pp=5;
octave:> [sf dsf]=masterlag(pp,xh);
octave:> plot(xh,sf,"linewidth",2)
```



```
octave:> xh=-1:0.01:1;
octave:> pp=6;
octave:> [sf dsf]=masterlag(pp,xh);
octave:> plot(xh,sf,"linewidth",2)
```



The use of the **shape functions** $\{\widehat{N}_i\}$ creates a **reparameterization** of the polynomials $\mathbb{P}_p(\widehat{K})$. Any quadratic

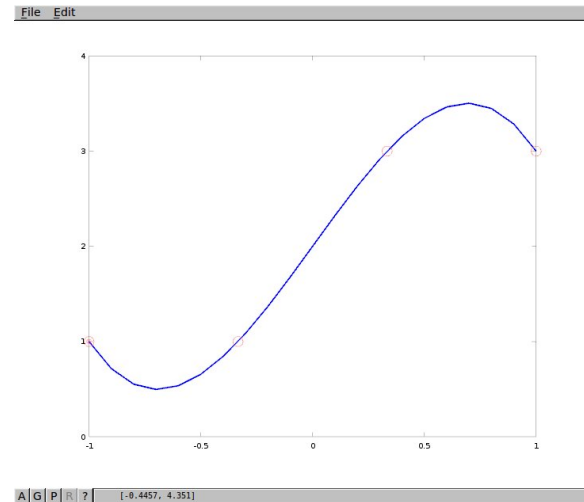
$$q(\widehat{x}) = \alpha_1 + \alpha_2 \widehat{x} + \alpha_3 \widehat{x}^2 ,$$

for example, can be written as

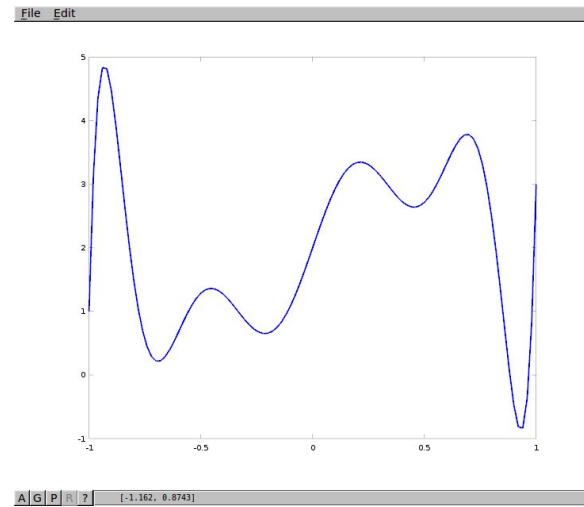
$$\begin{aligned} q(\widehat{x}) &= \beta_1 \widehat{N}_1^{(p=2)}(\widehat{x}) + \beta_2 \widehat{N}_2^{(p=2)}(\widehat{x}) + \beta_3 \widehat{N}_3^{(p=2)}(\widehat{x}) \\ &= \gamma_1 \widehat{N}_1^{(p=3)}(\widehat{x}) + \gamma_2 \widehat{N}_2^{(p=3)}(\widehat{x}) + \beta_3 \widehat{N}_3^{(p=3)}(\widehat{x}) + \beta_4 \widehat{N}_4^{(p=3)}(\widehat{x}) \\ &= \dots \end{aligned}$$

The degrees of freedom are now $\beta_1 - \beta_3$, if $p = 2$, or $\gamma_1 - \gamma_4$, if $p = 3$, etc. They are nothing but the values of q at the **nodes** of \widehat{K} . Each set of DOFs uniquely defines a polynomial.

```
octave:> x=-1:0.1:1;
octave:> pp=3;
octave:> gamma=[1 1 3 3];
octave:> plot(x,gamma*sf,"linewidth",2)
```




```
octave:> x=-1:0.01:1;
octave:> pp=9;
octave:> gamma=[1 1 1 1 1 3 3 3 3 3];
octave:> plot(x,gamma*sf,"linewidth",2)
```



Exo. 3.1 *Compute by hand the derivative of the shape functions of the P_3 element and compare with the functions coded in dsf.*

Exo. 3.2 *Extend the function masterlag to also calculate the second derivatives.*

```
function [sf dsf ddsf]=masterlag(p,xhat)
```

Takeaway

- It is **easy and automatic** to compute the Lagrange basis of $\mathbb{P}_p(\hat{K})$, and its derivatives. Simply call `masterlag(p,xhat)`! 😊
- **Code differentiation is awesome!** 😊😊

In fact, it can itself be **automated**. Automatic differentiation is considered one of the important algorithms of the 20th century (together with FFT, Monte Carlo, and others).

What do we have up to now?

- A **mesh**, this is, a **partition** of the domain Ω into a collection of **subdomains** $\{K\}$. This can be done by hand (in 1D), or automatically with available codes such as GMesh.
- A **transformation** F_K of a unique **master subdomain** \widehat{K} onto each K in the mesh.
- A set of **shape functions** which are a basis of $\widehat{P} = \mathbb{P}_p(\widehat{K})$, with a specific **parameterization** which defines suitable **degrees of freedom**.

To build a vector space of functions over Ω it only remains to:

- Use the transformation F_K to define **basis functions** in $K = F_K(\widehat{K})$. This also defines a set of **degrees of freedom** on the space $P(K)$.
- The finite element functions in Ω are defined piecewise, for each K , as a linear combination of the basis.

3.2 The finite element

Let \widehat{K} and K be given, and let $F_K : \widehat{K} \rightarrow K$ be a one-to-one mapping. Then to every function $\widehat{\psi} : \widehat{K} \rightarrow \mathbb{R}$ corresponds a function $\psi : K \rightarrow \mathbb{R}$ defined by

$$\psi(x) = \widehat{\psi}(F_K^{-1}(x)) \quad (3.4)$$

or, equivalently,

$$\psi(x) = \widehat{\psi}(F_K(\hat{x})) = \widehat{\psi}(\hat{x}) .$$

- If $\widehat{\psi}$ is a polynomial of degree k and F_K is affine (each component a polynomial of degree one), then ψ is a polynomial of degree k .
- $\widehat{\mathbb{P}}_k = \mathbb{P}_k$, if F_K is affine.
- This is **not** the only correspondence between functions in \widehat{K} and functions in K that is used. It is certainly the most frequent.

The **finite element** consists of

- A **master element**.
- A compact K , which is the image of a **transformation** F_K that maps \widehat{K} onto K .
- A set of **basis functions** which are the **transformed images** of the **shape functions**. In **most** finite elements,

$$N_i(F_K(\hat{x})) = \frac{1}{\alpha_i} \widehat{N}_i(\hat{x}) . \quad (3.5)$$

In the case of **Lagrange** elements, among others, $\alpha_i = 1$ for all i .

Denoting $x = F_K(\hat{x})$, we have

$$\sum_k \frac{\partial N_i}{\partial x_k}(x) B_{kj}(\hat{x}) = \frac{1}{\alpha_i} \frac{\partial \widehat{N}_i}{\partial \hat{x}_k} \quad (3.6)$$

where $B_{kj} = \partial(F_K)_k / \partial \hat{x}_j$ is the Jacobian matrix of the transformation $\hat{x} \mapsto x$. Another way of writing the previous equation is

$$B^T \nabla N_i(x) = \frac{1}{\alpha_i} \nabla \widehat{N}_i(\hat{x}) . \quad (3.7)$$

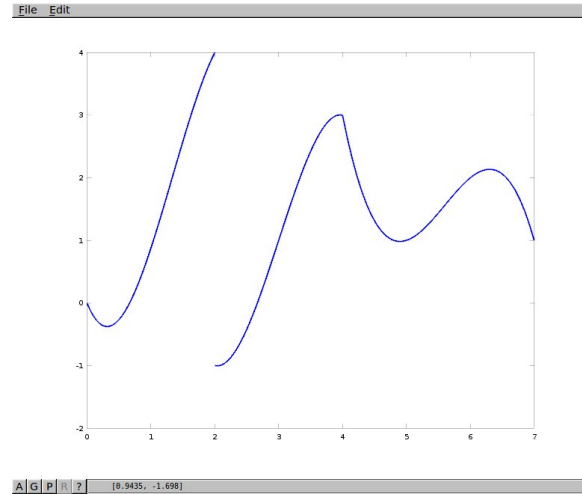
Example 1: Lagrange P_p element in 1D

Notice below the **mesh**, the **degree** and the **DOF values**

exapp.m

```
conecg=[1 2;2 3;3 4];
coordg=[0 2 4 7];
xh=-1:0.05:1;#drawing points
nhat=size(xh,2);
nel=size(conecg,1);npe=size(conecg,2);
nodg=size(coordg,2);
pp=3;
[g dg d2g] = masterp1(1,xh);
[sf dsf ddsf]=masterlag(pp,xh);
vdofs = [0 0 2 4;
        -1 0 2 3;
         3 1 2 1];
for k=1:nel
XX(:,1)=coordg(:,conecg(k,1));
XX(:,2)=coordg(:,conecg(k,2));
x=zeros(1,nhat);
for i=1:npe
x(1,:)=x(1,:)+XX(1,i)*g(i,:);
end
```

```
f=vdofs(k,:)*sf;
plot(x,f,"linewidth",2); hold on;
hold on;
end
hold off
```



If the last DOF of an element does not coincide with the first DOF of the next one, the function is **discontinuous**.

The unknowns are **renumbered**, so that all continuity requirements of the possible functions are enforced (identification of unknowns). This is easily accomplished with a **mesh**,

```
ndof=10;conecdof=[1 2 3 4;4 5 6 7;7 8 9 10];
```

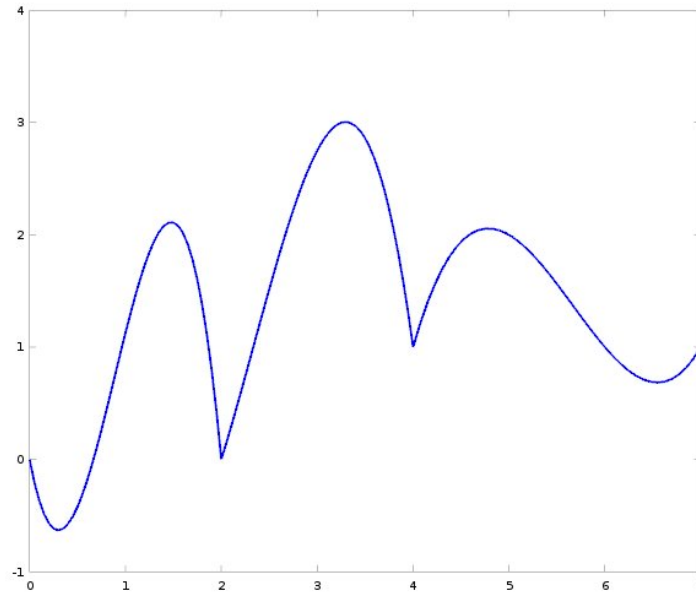
and the DOF vector

```
vdof=[0 0 2 0 2 3 1 2 1];
```

In this way, the following code draws the finite element function that corresponds to `vdof`.

exapp2.m

```
vdof=[0 0 2 0 2 3 1 2 1 1];
conecg=[1 2;2 3;3 4];
coordg=[0 2 4 7];
conecdof=[1 2 3 4;4 5 6 7;7 8 9 10];
pp=3;
xh=-1:0.05:1;#drawing points
nhat=size(xh,2);
nel=size(conecg,1);npe=size(conecg,2);
nodg=size(coordg,2);
[g dg d2g] = masterp1(1,xh);
[sf dsf ddsf]=masterlag(pp,xh);
for k=1:nel
XX(:,1)=coordg(:,conecg(k,1));
XX(:,2)=coordg(:,conecg(k,2));
x=zeros(1,nhat);
for i=1:npe
x(1,:)=x(1,:)+XX(1,i)*g(i,:);
end
f=vdof(conecdof(k,:))*sf;
plot(x,f,"linewidth",2); hold on;
hold on;
end
hold off
```



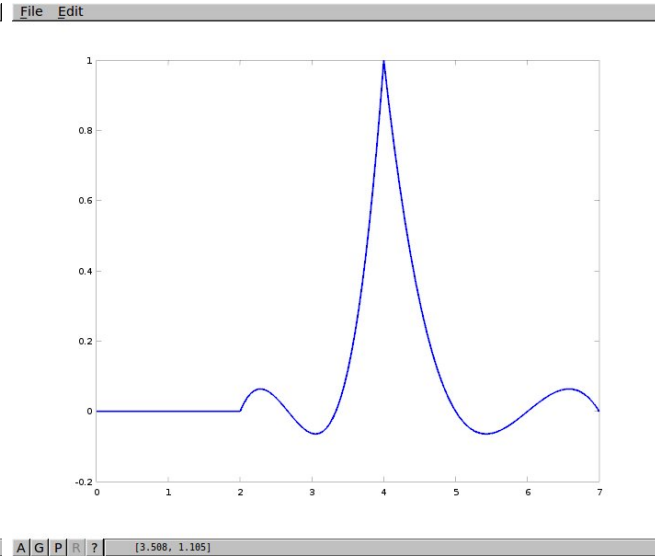
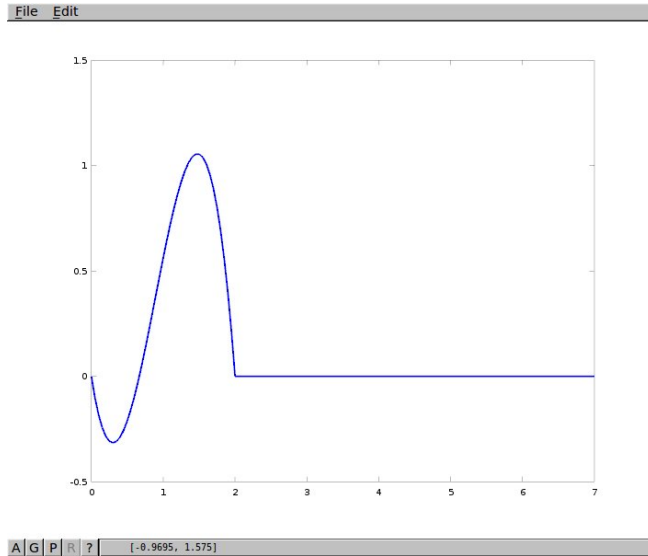
By changing the values in `vdof` we explore the **finite element space** W_h .

In this case the FE space is the set of piecewise polynomial functions, that are continuous at element boundaries.

$$W_h \in C^0(\Omega) , \text{ but } W_h \notin C^1(\Omega) . \quad (3.8)$$

vdof=[0 0 1 0 0 0 0 0 0]

vdof=[0 0 0 0 0 0 1 0 0]



There is a **correspondence** between the DOF vector U and the function $u_h \in W_h$. Setting $U_j = \delta_{ij}$ one obtains the **global basis function** ϕ_i . Above, $\phi_3(x)$ and $\phi_7(x)$.

Example 2: Lagrange P_p element in 2D

A similar construction can be performed to build P_p elements in 2D and 3D.

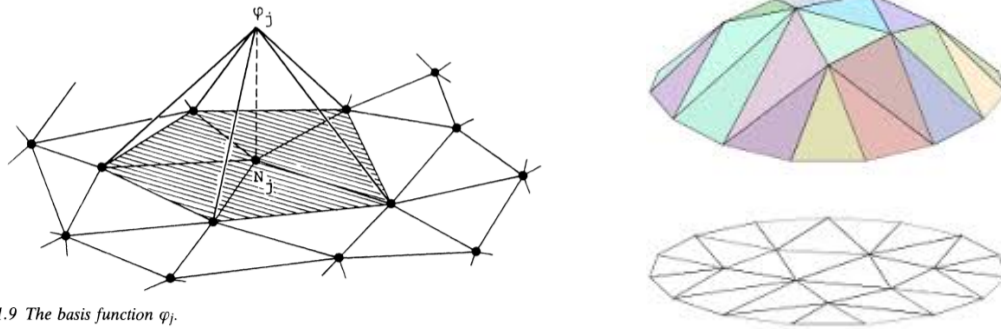
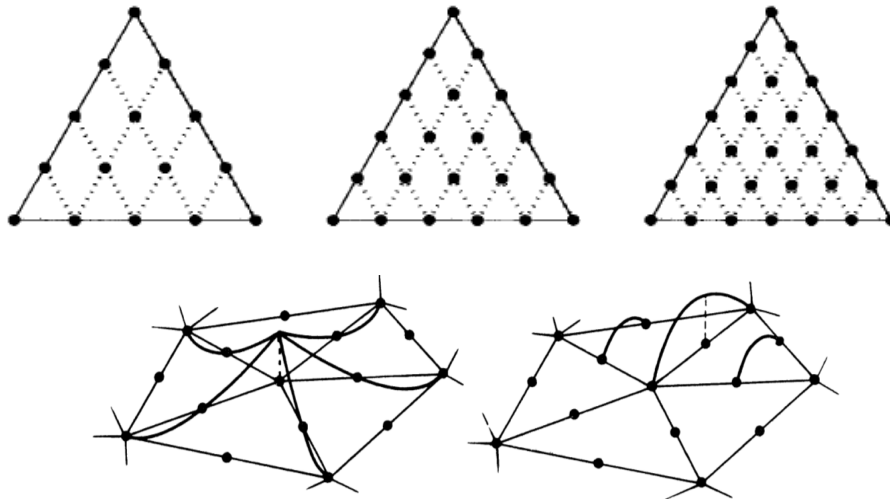



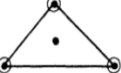
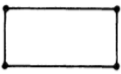
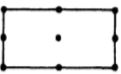
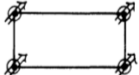









Fig 1.9 The basis function φ_j .



Degrees of freedom Σ Geometry	Function space P_K	Degree of continuity of corresponding FEM-space V_h
 3	$P_1(K)$	C^0
 6	$P_2(K)$	C^0
 10	$P_3(K)$	C^0
 10	$P_3(K)$	C^0
 4	$Q_1(K)$	C^0
 9	$Q_2(K)$	C^0

 16	$Q_3(K)$	C^1
 2	$P_1(K)$	C^0
 3	$P_2(K)$	C^0
 4	$P_3(K)$	C^1
 21	$P_5(K)$	C^1
 18	$P_5'(K)$ (see Problem 3.7)	C^1
 4	$P_1(K)$	C^0
 10	$P_2(K)$ (See Problem 3.4)	C^0

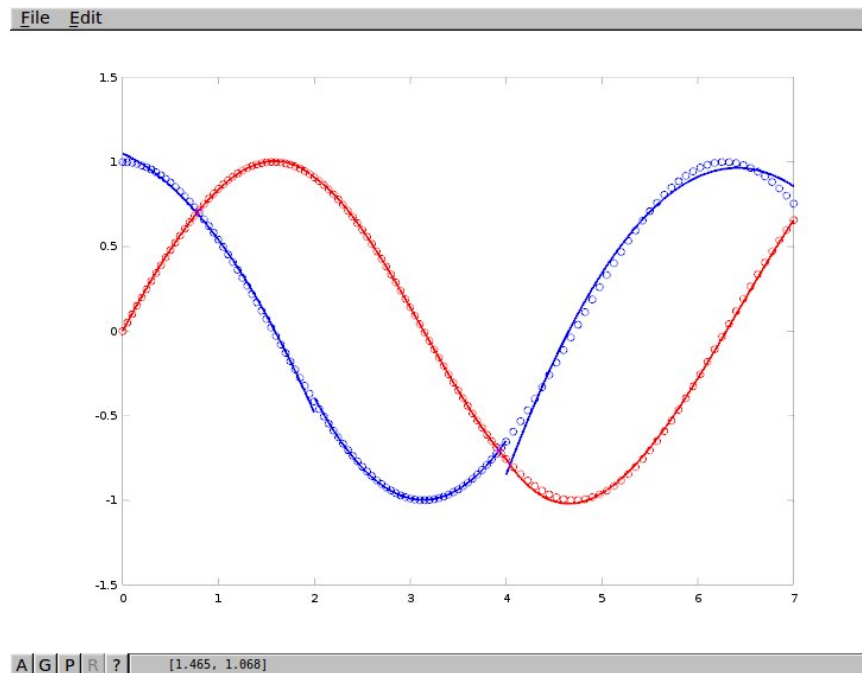
3.3 Finite element spaces and approximation

- The **finite element space** is the vector space obtained by assigning arbitrary values to the **degrees of freedom** of the **finite element mesh**.
- By suitably defining the DOFs, it is possible to ensure continuity of all the functions in the FE space. Notice that each **finite element** comes with its particular choice of DOF.
- There are **many** continuous FEs in 2D/3D, but **C^1 -continuity is rare**.
- The FE basis is **sparse** (for each i , there are few j 's such that $\text{supp}(\phi_i) \cap \text{supp}(\phi_j) \neq \emptyset$). This leads to **sparse matrices**.
- The DOF of the FE make it easy to impose **boundary conditions**.

Lemma: (Babuska, Suri, 1987) The space W_h has the capabilities of approximating functions quite accurately. If h is the size of the largest element, p the degree, then there exists C_k such that

$$\min_{v_h \in W_h} \|u - v_h\|_{L^2(\Omega)} \leq C_k \frac{h^{\min(p+1,k)}}{p^k} \|D^k u\|_{L^2(\Omega)} \quad (3.9)$$

which tends to zero as $h \rightarrow 0$ (refining the mesh) or $p \rightarrow +\infty$ (increasing the order). Below we see in red the function $\sin(x)$ (circles) and its interpolation u_h with the previously used mesh and $p = 3$ (continuous line). In blue, the function $\cos(x)$ and the derivative $u'_h(x)$.



Exo. 3.3 *The previous figure was obtained with the code `exapp5.m` below. Explain why the line*

```
df=vdof(conecdof(k,:))*dsf./dx;
```

computes f' , the derivative of the function f defined by the DOFs given by `vdof`.

```
cdof=[0 2/3 4/3 2 8/3 10/3 4 5 6 7];
vdof=sin(cdof);
conecg=[1 2;2 3;3 4];
coordg=[0 2 4 7];
conecdof=[1 2 3 4;4 5 6 7;7 8 9 10];
pp=3;
xh=-1:0.05:1;#drawing points
nhat=size(xh,2);
nel=size(conecg,1);npe=size(conecg,2);
nodg=size(coordg,2);
[g dg d2g] = masterp1(1,xh);
[sf dsf ddsf]=masterlag(pp,xh);
for k=1:nel
XX(:,1)=coordg(:,conecg(k,1));
XX(:,2)=coordg(:,conecg(k,2));
x=zeros(1,nhat); dx=zeros(1,nhat);
for i=1:npe
x(1,:)=x(1,:)+XX(1,i)*g(i,:);
dx(1,:)=dx(1,:)+XX(1,i)*dg(i,:);
end
f=vdof(conecdof(k,:))*sf;
df=vdof(conecdof(k,:))*dsf./dx;
plot(x,sin(x),"or","linewidth",1); hold on;
plot(x,f,"r","linewidth",2); hold on;
plot(x,cos(x),"ob","linewidth",1); hold on;
plot(x,df,"b","linewidth",2); hold on;
hold on;
end
hold off
```

Takeaway

- Combining a **mesh** and a **master element** through

$$N_i(F_K(\hat{x})) = \frac{1}{\alpha_i} \hat{N}_i(\hat{x}) , \quad (3.10)$$

we obtain a **finite element space** $W_h \subset \mathcal{C}^0(\Omega)$.

- The basis functions $N_i(x)$ are piecewise polynomials with compact support (sparse matrices).
- This space has **good approximation properties**, if h is small and/or p large.
- To solve a problem with exact solution $u(x)$, it remains to **define a strategy** to choose some $u_h \in W_h$ that approximates u .

4 Minimizing the energy

4.1 Minimization problems

Many problems can be formulated as minimizing some energy function, be it physical or not.

Example: Filtering a noisy function

Given a function $w \in L^2(\Omega)$ that we assume noisy, we can remove some of the noise by minimizing the energy

$$E(v) = \frac{1}{2} \int_{\Omega} (w - v)^2 d\Omega + \frac{\epsilon^2}{2} \int_{\Omega} \|\nabla v\|^2 d\Omega . \quad (4.1)$$

This results in a function u that is a smoothed version of the original function w , averaged over distances of order ϵ .

Example: Deformation of a chord

Let a chord be attached to $x = 0$ and $x = L$. The tension of the chord is T and it is subject to a transverse force $f(x)$. The equilibrium vertical displacement $u(x)$ minimizes the energy

$$E(v) = \int_0^L \left(\frac{T}{2} v'(x)^2 - f(x) v(x) \right) dx \quad (4.2)$$

over the space of **continuous functions** that **are zero at $x = 0$ and at $x = L$** .

Example: Solving elliptic equations

Consider the differential problem in $\Omega = (a, b)$:

$$-\frac{d}{dx} \left(\beta(x) \frac{du}{dx} \right) + \gamma(x) u(x) = f(x) \quad (4.3)$$

$$u(a) = A \quad (\text{Dirichlet}) \quad (4.4)$$

$$\beta(b)u'(b) = B \quad (\text{Neumann}) \quad (4.5)$$

Its exact solution minimizes the energy

$$E(v) = \int_a^b \left[\frac{\beta}{2} v'(x)^2 + \frac{\gamma}{2} v(x)^2 - f(x) v(x) \right] dx - B v(b) \quad (4.6)$$

over the set of functions **that take the value A at $x = a$.**

Example: Thermal conduction problems

A solid with thermal conductivity $\beta(x)$ occupies the region $\Omega \subset \mathbb{R}^3$. Its boundary $\partial\Omega$ consists of two parts: (a) Γ_1 , where the heat flux $Q(x)$ is known; (b) Γ_2 , where the surface is in contact with the ambient (temperature T_a , convection coefficient H).

The differential problem for the temperature field $T(x)$ is

$$-\nabla \cdot (\beta \nabla T) = 0 \quad \text{in } \Omega \quad (4.7)$$

$$\beta \frac{\partial T}{\partial n} = Q \quad \text{on } \Gamma_1 \quad (4.8)$$

$$\beta \frac{\partial T}{\partial n} = H (T_a - T) \quad \text{on } \Gamma_2 \quad (4.9)$$

$$(4.10)$$

The solution minimizes the (mathematical) energy

$$E(v) = \frac{1}{2} \int_{\Omega} \beta \|\nabla v\|^2 d\Omega + \frac{1}{2} \int_{\Gamma_2} H (T_a - v)^2 d\Gamma - \int_{\Gamma_1} Q v d\Gamma . \quad (4.11)$$

4.2 The Galerkin method

- Let the exact solution u of a **mathematical problem** be defined as the (unique) minimizer of some **energy functional** E over some infinite-dimensional function space V .
- Let V_h be a **finite-dimensional subspace** of V .

Then, the **Galerkin approximation** u_h of u is **defined** as the (unique) minimizer of E over V_h .

Under realistic hypotheses it is possible to prove the **optimal error bound**

$$\|u - u_h\|_V \leq C \min_{v_h \in V_h} \|u - v_h\|_V, \quad (4.12)$$

with C independent of h (mesh size) and p (polynomial degree).

4.3 A first application and implementation

4.3.1 Numerical method

Problem: Solve the differential problem

$$-(\beta u')' + \gamma u = f, \quad u(0) = A, \quad \beta(L)u'(L) = B, \quad (4.13)$$

with

$$\beta(x) = \begin{cases} 1 & \text{se } x < 2L/3 \\ 9 & \text{se } x > 2L/3 \end{cases}, \quad \gamma = 2, \quad f = x^2 \quad \forall x. \quad (4.14)$$

The energy, to be minimized over $V = \{v : (0, L) \rightarrow \mathbb{R} \mid v(0) = A\}$ is

$$E(v) = \int_0^L \left(\frac{1}{2} \beta (u')^2 + \frac{1}{2} \gamma u^2 - f u \right) dx - B u(L). \quad (4.15)$$

Numerical method (Galerkin method): Given a finite element space V_h , compute the **approximate solution** $u_h \in V_h$ defined by

$$E(u_h) \leq E(v_h) , \quad \forall v_h \in V_h, v_h(0) = A . \quad (4.16)$$

This is a **minimization problem in** V_h subject to the constraint $v_h(0) = A$.

In terms of the **basis functions** $\phi_1, \phi_2, \dots, \phi_M$ (which are **fixed** once the mesh has been chosen), and denoting the coefficients by U_1, \dots, U_M , we have

$$u_h(x) = U_1 \phi_1(x) + \dots + U_M \phi_M(x) . \quad (4.17)$$

This defines a mapping $\mathcal{U}_h : \mathbb{R}^M \rightarrow V_h$,

$$u_h = \mathcal{U}_h(\underline{U}) = U_1 \phi_1 + \dots + U_M \phi_M . \quad (4.18)$$

We can thus rewrite (4.16) in terms of vectors in \mathbb{R}^M :

$$\mathcal{E}(\underline{U}) \stackrel{\text{def}}{=} E(\mathcal{U}_h(\underline{U})) \leq E(\mathcal{U}_h(\underline{V})) = \mathcal{E}(\underline{V}) , \quad \forall \underline{V} \in \mathbb{R}^M, V_1 = A \quad (4.19)$$

where we have assumed that the node at $x = 0$ has been assigned the number 1. This shows that we have reduced the problem to a **minimization problem in** \mathbb{R}^M .

Numerical strategy:

1. Build a function `energy.m` that computes $\mathcal{E}(\underline{V})$ for any $\underline{V} \in \mathbb{R}^M$.
2. Call the Octave function `sqp` and get the solution \underline{U} .

```
U0=zeros(ndof,1); U=sqp(U0,@energy);
```

3. End! 😊

4.3.2 The energy code

Let us assume that $B = 0$ for now. Then, defining

$$Z(v_h(x), v'_h(x)) = \frac{1}{2} (\beta(x) v'_h(x)^2 + \gamma(x) v_h(x)^2) - f(x) v(x) , \quad (4.20)$$

and using that $\Omega \simeq \cup_{K=1}^{N_{el}} F_K(\widehat{K})$, we can **decompose the energy integral** as

$$E = \int_0^L Z(v_h(x), v'_h(x)) dx = \sum_K \int_{F_K(\widehat{K})} Z(v_h(x), v'_h(x)) dx \quad (4.21)$$

and now **change variables to \widehat{K}** , to give

$$E = \sum_K \int_{\widehat{K}} Z(v_h(F_K(\hat{x})), v'_h(F_K(\hat{x}))) \frac{\partial F_K}{\partial \hat{x}}(\hat{x}) d\hat{x} . \quad (4.22)$$

The last ingredient is **numerical integration in \widehat{K}** .

Parenthesis: Numerical integration (quadrature)

1. One defines a **set of m points in \hat{K}** ,

$$\hat{q}_1, \hat{q}_2, \dots, \hat{q}_m ,$$

2. and a **set of m weights**,

$$W_1, W_2, \dots, W_m .$$

3. One approximates any integral in \hat{K} as

$$\int_{\hat{K}} g(\hat{x}) d\hat{x} \simeq g(\hat{q}_1)W_1 + g(\hat{q}_2)W_2 + \dots + g(\hat{q}_m)W_m = \sum_{j=1}^m g(\hat{q}_j) W_j . \quad (4.23)$$

Now we go back to the computation of the energy...

So,

$$\begin{aligned}
E &= \int_0^L Z(v_h(x), v'_h(x)) \, dx = \sum_K \int_{F_K(\hat{K})} Z(v_h(x), v'_h(x)) \, dx \\
&= \sum_K \int_{\hat{K}} Z(v_h(F_K(\hat{x})), v'_h(F_K(\hat{x}))) \frac{\partial F_K}{\partial \hat{x}}(\hat{x}) \, d\hat{x} \\
&= \sum_K \left[\sum_{j=1}^m Z(v_h(F_K(\hat{q}_j)), v'_h(F_K(\hat{q}_j))) J_K(\hat{q}_j) W_j \right]
\end{aligned}$$

where we have set $J_K = \partial F_K / \partial \hat{x}$.

For easy programming, one first computes $q_j = F_K(\hat{q}_j)$ and then calculates

$$E = \sum_K \left[\sum_{j=1}^m Z(v_h(q_j), v'_h(q_j)) J_K(\hat{q}_j) W_j \right] \tag{4.24}$$

Remember, $Z(x) = \frac{1}{2} (\beta(x) v'_h(x)^2 + \gamma(x) v_h(x)^2) - f(x) v_h(x)$, so that **if we can compute v_h and v'_h at the images of the quadrature points**, we are done. We now show how to do that.

Exo. 4.1 Let \mathbf{xh} be a point in $\hat{K} = [-1, 1]$, and let $\mathbf{XX}(1)$ and $\mathbf{XX}(2)$ be the extreme points of K , which is an element of type P_p . Let also $\mathbf{vdof}(1:\mathbf{pp}+1)$ be the values of \underline{U} corresponding to the degrees of freedom of element K .

Show that the following code computes:

- The variable \mathbf{x} , which is the image of \mathbf{xh} by F_K .
- The variable \mathbf{dx} , which is the value of F'_K at \mathbf{xh} .
- The variable \mathbf{uh} , which is the value of u_h at \mathbf{x} .
- The variable \mathbf{duh} , which is the value of u'_h at \mathbf{x} .

```
[g dg d2g]=masterlag(1,xh);  
[sf dsf ddsf]=masterlag(pp,xh);  
x=XX(1)*g(1)+XX(2)*g(2);  
dx=XX(1)*dg(1)+XX(2)*dg(2);  
uh=vdof*sف;  
duh=vdof*dsf/dx;
```

This exercise should be enough to understand the energy function below.

$$E = \sum_K \left[\sum_{j=1}^m \left(\frac{\beta(q_j)}{2} u_h'(q_j)^2 + \frac{\gamma(q_j)}{2} u_h(q_j)^2 - f(q_j) u_h(q_j) \right) J_K(\hat{q}_j) W_j \right] \quad (4.25)$$

```

function E=energy1(vdof)
    global L B pp nel
    global npe nodg ndof conecg conec dof coordg
    vdof=vdof';
    ## integration points and weights
    xh=-1:0.1:1;
    wh=[1/20 1/10*ones(1,19) 1/20];
    nhath=size(xh,2);
    [g dg d2g]=masterlag(1,xh);
    [sf dsf ddsf]=masterlag(pp,xh);
    E=0;
    for k=1:nel
        XX(:,1)=coordg(:,conecg(k,1));
        XX(:,2)=coordg(:,conecg(k,2));
        x=zeros(1,nhath);
        dx=zeros(1,nhath);
        for i=1:npe
            x(1,:)=x(1,:)+XX(1,i)*g(i,:);
            dx=dx+XX(1,i)*dg(i,:);
            uh=vdof(conecg(k,:))*sf;
            duh=vdof(conecg(k,:))*dsf./dx;
            #####data
            beta=(1+4*(1+sign(x-2*L/3))).*ones(1,nhath);
            gamma=2*ones(1,nhath);
            ff=x.*x.*ones(1,nhath);
            #####
            E=E+0.5*duh*(duh.*beta.*wh.*dx)'...
                +0.5*uh*(uh.*gamma.*wh.*dx)'...
                -ff*(uh.*wh.*dx)';
        end
        E=E-B*vdof(ndof);
    end

```

And now the code that builds the mesh, solves the problem and plots the solution.

```
## Application 1
##
#### data
global L=5;
global A=1;
global B=-30;
global pp=3;
global nel=3;
#### build mesh
global npe nodg ndof conecg conec dof coordg
npe=2;
nodg=nel+1;
ndof=nel*pp+1;
conecg=[1 2];
conec dof=[1:pp+1];
for i=2:nel
    conecg=[conecg;i,i+1];
    conec dof=[conec dof;(i-1)*pp+1:i*pp+1];
end
coordg=zeros(1,nodg);
hsize=L/nel;
for i=2:nodg
    coordg(i)=(i-1)*hsize;
end
####
U0=zeros(ndof,1);
[U obj info iter nf lambda]=...
sqp(U0,@energy1,@restrictions);
k=plotfem1(conecg,coordg,pp,conec dof,vv)

-----

function r=restrictions(vdof)
global A;
ndof=max(size(vdof));
r=[vdof(1)-A];
end
```

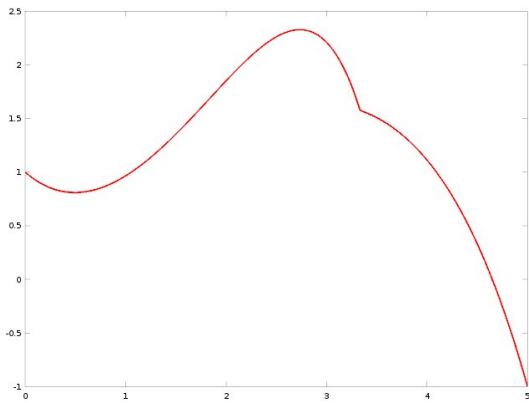
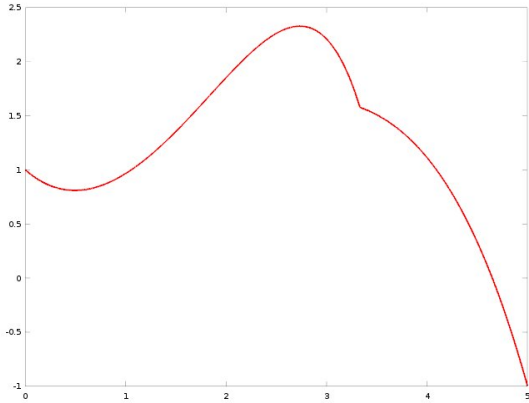
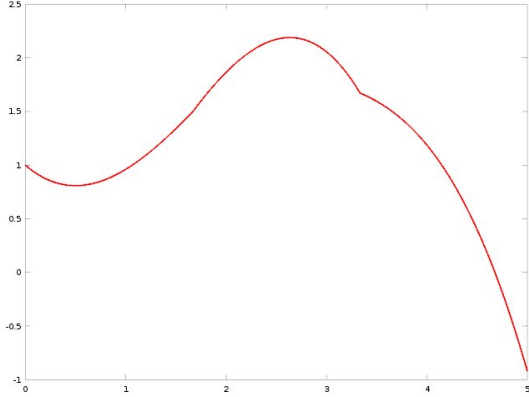
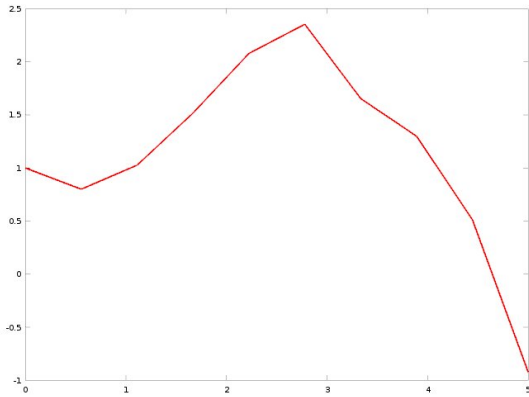
The whole code consists of: `masterlag.m` (21 lines), `energy1.m` (28 lines), `restrictions.m` (5 lines) and `applic1.m` (23 lines). Yes, there is the **plotting** function, too (see below).

4.3.3 The plotting function

This function receives the mesh (`conecg`, `coordg`), the degree `pp`, the connectivity of DOFs `conecdof` and the vector \underline{U} of DOFs (in `vdof`), and plots the corresponding function $\mathcal{U}_h(\underline{U})$.

```
function idum=plotfem1(conecg,coordg,...    XX(:,2)=coordg(:,conecg(k,2));
                        pp,conecdof,vdof)    x=zeros(1,nhat);
xh=-1:0.05:1;#drawing points              for i=1:npe
nhat=size(xh,2);                          x(1,:)=x(1,:)+XX(1,i)*g(i,:);
nel=size(conecg,1);npe=size(conecg,2);    end
nodg=size(coordg,2);                      f=vdof(conecdof(k,:))*sf;
[g dg d2g] = masterlag(1,xh);              plot(x,f,"r","linewidth",2); hold on;
[sf dsf ddsf]=masterlag(pp,xh);           hold on;
for k=1:nel                                idum=0;
XX(:,1)=coordg(:,conecg(k,1));            end
```

Just 18 lines! 😊



Numerical solutions with $9 P_1, 3 P_3, 90 P_1, 30 P_3$.

4.3.4 A more efficient code

A much more efficient implementation is to compute ∇E and solve for $\nabla E(u_h) = 0$. You will find this implemented in `applic1bis.m`.

The key step is, instead of minimizing, to invoke the (nonlinear) solver `fsolve` in the following way:

```
vv=fsolve(@denenergy1,vv0);
```

This will result in a vector `vv` such that all the derivatives of the energy with respect to the degrees of freedom are zero (i.e., the minimum).

The function `denenergy1` must provide the derivative of E with respect to the degrees of freedom. It is implemented in `denenergy1.m`.

Exo. 4.2 *Understand (line by line) the function `denenergy1.m`. Try to find the relationship between it and the functions `energy1.m` and `restrictions.m`.*