# An example using FEM

Stefano Piani

January 5, 2021

The exercise that I propose for today is to use the finite element method to solve the stationary reaction–diffusion equation, i.e. given two functions

$$\sigma\colon [a,b] \longrightarrow \mathbb{R}$$

$$f\colon [a,b] \longrightarrow \mathbb{R}$$

find a function $u\colon [a,b] \longrightarrow \mathbb{R}$ such that

$$\begin{cases} -u'' + \sigma u = f \\ u(a) = 0 \\ u(b) = 0 \end{cases} \tag{1}$$

We haven't required any kind of regularity (yet!) on $f$ and $\sigma$ because we want to rewrite the previous problem using a weak formulation (and, therefore, some hypothesis that may seem mandatory here could be useless later)

But, before we start, let me show you that this problem can be a lot "less innocent" than what it may look like!

For example, if $a = 0$, $b = \pi$, $\sigma = -1$ and $f = -x$, this problem does not have any solution.

$$\begin{cases} u'' + u = x \\ u(0) = 0 \\ u(\pi) = 0 \end{cases} \tag{2}$$

Indeed, it easy to show that the following problem does not have any solution

$$\begin{cases} g'' + g = 0 \\ g(0) = 0 \\ g(\pi) = -\pi \end{cases} \tag{3}$$

The first equation of the problem forces our solution to be

$$s(x) = c_1 \cos(x) + c_2 \sin(x)$$

for some real constants $c_1$ and $c_2$. Unfortunately, boundary condition on 0 requires $c_1$ to be 0 while the other boundary condition forces $c_1$ to be $\pi$; therefore, there is no solution.

Going back to problem 2, if $u$ would be a solution of problem 2, it is easy to check that

$$g \overset{\text{def}}{=} u - x$$

would be a solution of problem 3. Therefore, problem 2 does not have any solution.

We have shown that, even for simple choices of the constants (like $\sigma = -1$ or $f = -x$), it may happen that the problem 1 does not admit any solution.

Now, we want to obtain a weak formulation for problem 1. In order to do that, we consider a test function $v\colon [a,b] \longrightarrow \mathbb{R}$ such that

$$v(a) = 0 \qquad\qquad v(b) = 0$$

It is complicated to explain *why* we have to impose the conditions on the boundary for $v$. The general idea is that where you have Dirichlet boundary conditions on $u$ (homogeneous or not) you impose homogeneous (always homogeneous) boundary conditions on $v$ (let me remind you that “homogeneous” means that the right hand side of the equation that describes the boundary condition is 0).

Why must they always be homogeneous (even if the conditions on $u$ are not homogeneous)? Well, because otherwise the space where $v$ lives (we will call it $V$) will not be a vector space (indeed, if $v_1(a) = k$ and $v_2(a) = k$ then $(v_1 + v_2)(a) \neq k$ if $k \neq 0$).

Why do we have to impose them? Well, an intuitive idea could be that, in this way, the space where $u$ lives and the space of $v$ are "equally big" (whatever this means for infinite dimensional spaces).

Moreover, we already know the value of the solution near $a$ and $b$ (because of the boundary conditions) and therefore we do not need to test $u$ near the boundaries.

If you are reading this slide for the first time in your life, I do not expect it to be clear. Read until we write the weak formulation of the problem and then come back to those points. They will be (hopefully) a little bit more understandable.

Unfortunately, the most clear and understandable reason why we have to impose homogeneous boundary conditions on $v$ goes beyond the topics of this course and is related to the fact that the weak formulation is often a variational formulation where $u$ is a minimum of some functional and $v$ is a perturbation of this minimum (and, therefore, $u + v$ shall have the same boundary conditions of $u$). So, if this is not totally clear, do not worry too much: just take for granted that $v(a)$ and $v(b)$ are 0!

Let us go back to our formulation! Now that we have a test function $v$, we multiply our old equation by $v$ and we integrate over the interval $[a, b]$

$$\int_a^b -u''v \, \mathrm{d}x + \int_a^b \sigma uv \, \mathrm{d}x = \int_a^b fv \, \mathrm{d}x \tag{4}$$

If we write the problem with this formulation, we need $u$ to be derivable twice. We can loosen this requirement if we use the integration by parts rule

$$\int_a^b u''v \, \mathrm{d}x = u'(b)v(b) - u'(a)v(a) - \int_a^b u'v' \, \mathrm{d}x$$

In this way, the second derivative of $u$ does not appear in the right hand side of the equation.

Because of the fact that $v(a) = 0$ and $v(b) = 0$ we obtain that the equation 4 becomes

$$\int_a^b u'v' \, \mathrm{d}x + \int_a^b \sigma uv \, \mathrm{d}x = \int_a^b fv \, \mathrm{d}x$$

If we take a look at the previous equation, there are a few things that we need to ensure that exist

$$\int_a^b u'v' \,\mathrm{d}x + \int_a^b \sigma uv \,\mathrm{d}x = \int_a^b fv \,\mathrm{d}x$$

In particular, we need to ensure that the following terms exist

- $\displaystyle\int_a^b u'v' \,\mathrm{d}x$
- $\displaystyle\int_a^b \sigma uv \,\mathrm{d}x$
- $\displaystyle\int_a^b fv \,\mathrm{d}x$

To ensure that

$$\int_a^b u'v'\,\mathrm{d}x$$

exists, we will require that $u' \in L^2([a,b])$ and $v' \in L^2([a,b])$. In this way, the existence of the integral is guaranteed by the Cauchy–Schwartz inequality.

The existence of

$$\int_a^b \sigma uv\,\mathrm{d}x \tag{5}$$

is of course related to the property of the parameter $\sigma$. We add the hypothesis that $\sigma \in L^\infty([a,b])$, so we have just to ensure that

$$\int_a^b uv\,\mathrm{d}x$$

exists to prove that the integral 5 exists. Therefore, we require that $u \in L^2([a,b])$ and $v \in L^2([a,b])$ like we did in the previous case.

The last term is

$$\int_a^b f v \,\mathrm{d}x$$

that, of course, depends on the parameter $f$; taking into account that we already request $v$ to be in $L^2([a,b])$, we will add the hypothesis that $f \in L^2([a,b])$.

Saying that $u$ must be a function in $L^2$ whose (weak) derivative is still in $L^2$, means that $u \in H^1([a,b])$ (by definition of $H^1$). Moreover, because of the boundary conditions

$$u \in H_0^1([a,b]) \stackrel{\text{def}}{=} \left\{ f \in H^1([a,b]) \,|\, f(a) = 0 \text{ and } f(b) = 0 \right\}$$

For the same reason, we have also that $v \in H_0^1([a,b])$.

Finally, we still have to deal with the fact that, as we have seen before, there may not be any solution.

One of the reason why we did not find any solution in the problem 2 was that $\sigma$ was negative. Indeed, it can be shown that, if $\sigma(x) \geq 0$ almost everywhere, our problem has always one (and only one) solution.

We will not prove this result, but the proof is easy if you know the Lions–Lax–Milgram theorem.

We will add the hypothesis that $\sigma(x) \geq 0$ almost everywhere to ensure the existence and uniqueness of the solution.

We finally have a weak formulation for our problem.

*Let $[a,b]$ be a closed interval in $\mathbb{R}$. Let $f \in L^2([a,b])$ be a real function and $\sigma \in L^\infty([a,b])$ be a real function such that $\sigma(x) \geq 0$ almost everywhere.*

*Find a function $u \in H_0^1([a,b])$ such that, for every function $v \in H_0^1([a,b])$, we have*

$$\int_a^b u'v' \,\mathrm{d}x + \int_a^b \sigma u v \,\mathrm{d}x = \int_a^b f v \,\mathrm{d}x$$

Accordingly to the previous equation, we define the bilinear operator

$$\begin{aligned} \mathcal{A}\colon H_0^1([a,b]) \times H_0^1([a,b]) &\to \mathbb{R} \\ (u,v) &\mapsto \int_a^b u'v'\,\mathrm{d}x + \int_a^b \sigma uv\,\mathrm{d}x \end{aligned}$$

and the linear function

$$\begin{aligned} \mathcal{F}\colon H_0^1([a,b]) &\to \mathbb{R} \\ v &\mapsto \int_a^b fv\,\mathrm{d}x \end{aligned}$$

In this way, our problem becomes:

*Find a function u in* $H_0^1([a,b])$ *such that for every function* $v \in H_0^1([a,b])$

$$\mathcal{A}(u,v) = \mathcal{F}(v)$$

The idea of the finite element method is to choose two *finite* dimensional spaces $U_h$ and $V_h$ and to solve the following problem

*Find a function u in* $U_h$ *such that for every function* $v \in V_h$

$$\mathcal{A}(u,v) = \mathcal{F}(v)$$

This is the main difference between finite element methods and finite difference ones. In this case, the approximation is on the functional spaces, not on the operator $\mathcal{A}(u,v)$.

Now we have to choose some suitable spaces $U_h$ and $V_h$ in order of applying our finite element method.

For simplicity, in this case we will choose only one finite dimensional space $S_h$ and we will impose

$$U_h = S_h \qquad\qquad V_h = S_h$$

so that we have to choose only one space (this is a really common choice).

We have several possibilities for $S_h$: the most common one is to split the interval $[a, b]$ into some subintervals

$$[a, x_1], [x_1, x_2], \ldots, [x_{n-1}, b]$$

called *elements* and to impose that, on each element, the function $u$ is a polynomial.

But let us go step by step. Let us suppose that we have chosen a space $S_h$. How can we find $u \in S_h$ such that $\forall v \in S_h$

$$\mathcal{A}(u, v) = \mathcal{F}(v) \tag{6}$$

using our computer?

Let $e_1, \ldots, e_n$ be a vector basis of $S_h$. If we find a function $u \in S_h$ such that $\forall i \in \{1, \ldots, n\}$

$$\mathcal{A}(u, e_i) = \mathcal{F}(e_i)$$

then $u$ satisfy the equation 6 for every $v$.

Indeed, if $v = \alpha_1 e_1 + \cdots + \alpha_n e_n$, we have

$$\begin{aligned}
\mathcal{A}(u, v) &= \mathcal{A}(u, \alpha_1 e_1 + \cdots + \alpha_n e_n) \\
&= \alpha_1 \mathcal{A}(u, e_1) + \cdots + \alpha_n \mathcal{A}(u, e_n) \\
&= \alpha_1 \mathcal{F}(e_1) + \cdots + \alpha_n \mathcal{F}(e_n) \\
&= \mathcal{F}(\alpha_1 e_1 + \cdots + \alpha_n e_n) \\
&= \mathcal{F}(v)
\end{aligned}$$

Therefore, to find our solution $u$, we have just to solve $n$ equations (one for each element of the basis).

We can write explicitly these equations. Indeed, if we suppose to write our solution $u$ as a linear combination of the elements of the basis

$$u = \lambda_1 e_1 + \cdots + \lambda_n e_n$$

then each one of the $n$ equations can be rewritten as

$$\mathcal{A}(u, e_i) = \mathcal{F}(e_i)$$

$$\mathcal{A}(\lambda_1 e_1 + \cdots + \lambda_n e_n, e_i) = \mathcal{F}(e_i)$$

$$\lambda_1 \mathcal{A}(e_1, e_i) + \cdots + \lambda_n \mathcal{A}(e_n, e_i) = \mathcal{F}(e_i)$$

Defining the constants

$$A_{ij} \overset{\text{def}}{=} \mathcal{A}(e_j, e_i) \qquad\qquad F_i = \mathcal{F}(e_i)$$

the previous equation can be rewritten as

$$A_{i1}\lambda_1 + \cdots A_{in}\lambda_n = F_i$$

Our $n$ equations, therefore, define a linear system respect to the variables $\lambda_i$

$$\begin{cases} A_{11}\lambda_1 + \cdots + A_{1n}\lambda_n = F_1 \\ A_{2n}\lambda_1 + \cdots + A_{2n}\lambda_n = F_2 \\ \qquad\qquad \vdots \\ A_{n1}\lambda_1 + \cdots + A_{nn}\lambda_n = F_n \end{cases}$$

or, more shortly,

$$\mathbf{A}\lambda = \mathbf{F}$$

where **A** is the matrix whose entries are $A_{ij}$ and **F** is the vector whose entries are the numbers $F_i$.

This is something that all the methods (FDM, FVM and FEM) have in common: at the end, they approximate a PDE with a linear system.

There is an important difference, though, between finite difference methods and finite element methods! If you are using finite difference methods, the result of the linear system is an approximation of the real solution on some points, i.e. $\lambda_i$, the $i$–th element of the solution vector $\lambda$, is an approximation of the solution $u$ evaluated at a specific point $x_i$.

For finite element methods, this is not true. $\lambda_i$ does not represent (in principle) the value of $u$ at any point. For example, it is possible that your solution is always positive but that there exists $i$ for which $\lambda_i$ is negative. If you want to evaluate the solution generated by your method on a specific point $x_i$ you have to compute

$$\lambda_1 e_1(x_i) + \lambda_2 e_2(x_i) + \cdots + \lambda_n e_n(x_i)$$

The last problem we have to deal with is *"how do we choose the space $S_h$?"*.

A common approach is to generate a partition of our original domain (in our case, the interval $[a, b]$) with some subintervals

$$[x_0, x_1], [x_1, x_2], \dots, [x_{k-1}, x_k]$$

with $x_0 \overset{\text{def}}{=} a$ and $x_k \overset{\text{def}}{=} b$.

Moreover, we define $I_h \overset{\text{def}}{=} [x_{h-1}, x_h]$ and

$$S_h \overset{\text{def}}{=} \left\{ \begin{array}{c} \text{continuous } f \in C([a, b]) \text{ such that their restriction} \\ \text{on each interval } I_h \text{ is a polynomial of degree } \leq d \end{array} \right\}$$

We need to build a vector basis for this space!

Let us start from the easiest case: let us suppose that we are using only one interval and that, therefore,

$$x_0 \overset{\text{def}}{=} a \qquad\qquad x_1 \overset{\text{def}}{=} b$$

In this case, the space $S_h$ is just the space of the polynomial of degree smaller or equal than $d$. The dimension of this space is $d + 1$.

What basis can we chose for this space? For example, we could choose the monomial basis. But, from the theory of interpolation, we know that this is not a good idea!

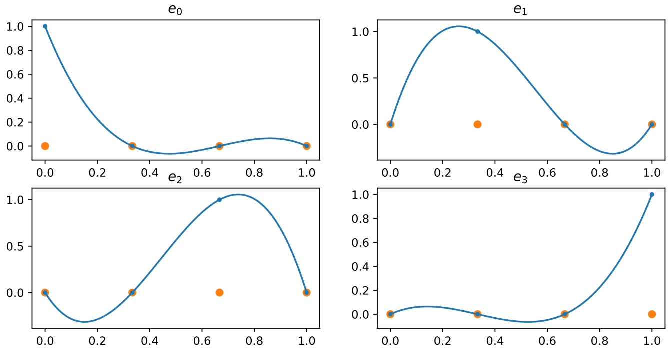
Instead we are going to use the *lagrangian basis*.

For this reason, we choose $d + 1$ points $p_0, \ldots, p_d$ inside our interval $[a, b]$ and we define $\ell_i(x)$ as the only polynomial of degree $d$ such that $\ell_i(p_i) = 1$ and $\ell_i(p_j) = 0$ for $j \neq i$.

For simplicity (but this is not mandatory) we will assume that $p_0 = a$ and $p_d = b$, i.e. the first and the last node are the boundary of the interval. The reason why we do that will be clear later.

The vector basis of the space $S_h$ will be $\ell_0, \ldots, \ell_d$.

For example, if our domain is $[0, 1]$ and we choose degree 3, we obtain the following basis



As you should already know, a good choice for the nodes are the Chebychev points. In this case, because we have few nodes and because I want to put the first and the last node on the boundary, I have choose equally space ones (the orange dots). But if you want to use many nodes, this choice is not optimal.

What happens if we have more than one element (subinterval)? In this case, choosing a basis is a little bit more complicated.

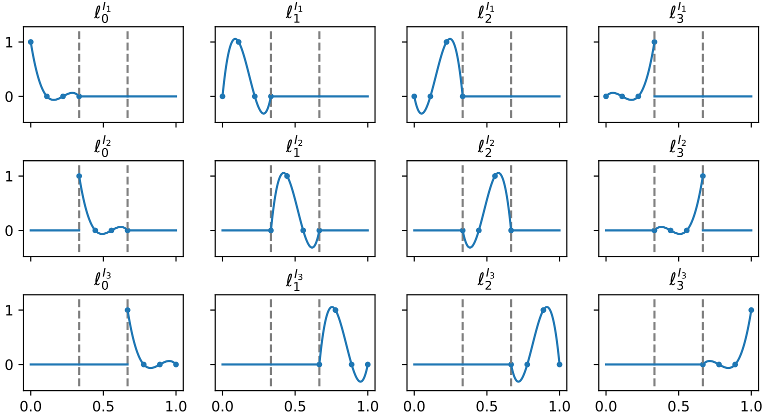
Let us recall that we have chosen that

$$S_h \overset{\text{def}}{=} \left\{ \begin{array}{c} \text{continuous } f \in C([a,b]) \text{ such that their restriction} \\ \text{on each interval } I_h \text{ is a polynomial of degree } \leq d \end{array} \right\}$$

The main idea is to go interval by interval and build, for each interval $I_h$, the lagrangian basis $\ell_0, \ldots, \ell_d$ we have seen before. Of course, these functions are defined only on $I_h$, but we can extend them imposing that, if $x$ is outside $I_h$,

$$\ell_i(x) = 0$$

For example, if we split the interval $[0, 1]$ in three parts and we use again degree $d = 3$, we have the following functions



Unfortunately, there is a problem! Some functions are not continuous! How can we fix this problem?

Let us change prospective. Before, we have seen that, with degree 3, we have 4 lagrangian polynomials for each interval $I_h$. Therefore, we have a total of 12 lagrangian polynomials:

$$\ell_0^{I_1}, \ell_1^{I_1}, \ell_2^{I_1}, \ell_3^{I_1}, \quad \ell_0^{I_2}, \ell_1^{I_2}, \ell_2^{I_2}, \ell_3^{I_2}, \quad \ell_0^{I_3}, \ell_1^{I_3}, \ell_2^{I_3}, \ell_3^{I_3}$$

A linear combination of these polynomials is an expression like

$$f \overset{\text{def}}{=} \lambda_1 \ell_0^{I_1} + \lambda_2 \ell_1^{I_1} + \lambda_3 \ell_2^{I_1} + \lambda_4 \ell_3^{I_1} + \lambda_5 \ell_0^{I_2} + \ldots + \lambda_{11} \ell_2^{I_3} + \lambda_{12} \ell_3^{I_3}$$

How can we ensure that this expression is continuous? Of course, the problem is only between the intervals, on the points $p_1 \overset{\text{def}}{=} I_1 \cap I_2$ and $p_2 \overset{\text{def}}{=} I_2 \cap I_3$ because, inside an interval $I_h$, our function $f$ is just a sum of 4 polynomials.

It is also easy to show that

$$\lim_{x \to p_1^-} f(x) = \lim_{x \to p_1^-} \left( \lambda_1 \ell_0^{I_1} + \lambda_2 \ell_1^{I_1} + \lambda_3 \ell_2^{I_1} + \lambda_4 \ell_3^{I_1} \right) = \lambda_4$$

and, on the other side,

$$\lim_{x \to p_1^+} f(x) = \lim_{x \to p_1^+} \left( \lambda_5 \ell_0^{I_2} + \lambda_6 \ell_1^{I_2} + \lambda_7 \ell_2^{I_2} + \lambda_8 \ell_3^{I_2} \right) = \lambda_5$$

To have a continuous function, we have to impose that

$$\lambda_4 = \lambda_5$$

On the very same way, for the point $p_2$ we have to impose that

$$\lambda_8 = \lambda_9$$

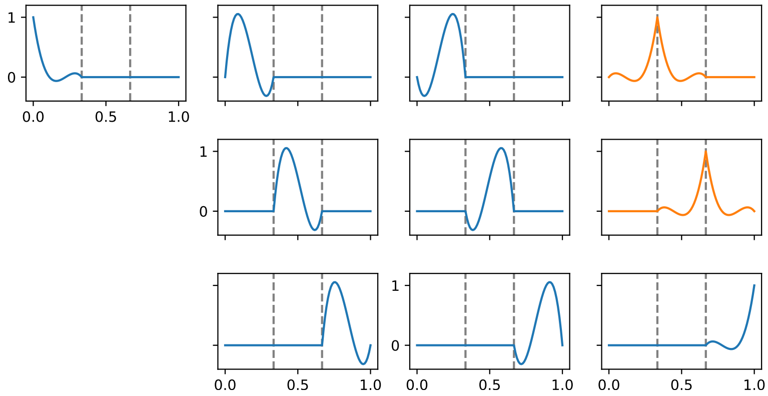
Therefore, if a linear combination $f$ is continuous, it will have the following expression

$$f = \lambda_1 \ell_0^{I_1} + \ldots + \lambda_4(\ell_3^{I_1} + \ell_0^{I_2}) + \lambda_6 \ell_1^{I_2} + \ldots + \lambda_8(\ell_3^{I_2} + \ell_0^{I_3}) + \ldots + \lambda_{12} \ell_3^{I_3}$$

We have found our new base, made of 10 elements! Indeed, we will consider the function $\ell_3^{I_1} + \ell_0^{I_2}$ as a unique element of our basis (and the same for $\ell_3^{I_2} + \ell_0^{I_3}$).

Of course, we will slightly modify the function $\ell_3^{I_1} + \ell_0^{I_2}$ on the point $p_1$ so that it evaluates to 1 on that point (and not 2). In other words, we modify the only point where, otherwise, the continuity will be lost.

The following picture shows the elements of the new basis we have just found



Here, the two lagrangian polynomials that have been "glued" together are shown in orange. Now all the functions are continuous.

Starting by the (hopefully, well known) fact that $\ell_1, \dots, \ell_d$ are a basis for the space of polynomials of degree $d$, it is easy to show that the previous functions are a basis for the space $S_h$ (if you want, you can do that as an exercise)

The last point we have to deal with is how to implement all this stuff in a reasonable algorithm.

Let us define the elements of our basis

$$\begin{aligned} e_1 &\overset{\text{def}}{=} \ell_0^{I_1} & e_6 &\overset{\text{def}}{=} \ell_2^{I_2} \\ e_2 &\overset{\text{def}}{=} \ell_1^{I_1} & e_7 &\overset{\text{def}}{=} \ell_3^{I_2} + \ell_0^{I_3} \\ e_3 &\overset{\text{def}}{=} \ell_2^{I_1} & e_8 &\overset{\text{def}}{=} \ell_1^{I_3} \\ e_4 &\overset{\text{def}}{=} \ell_3^{I_1} + \ell_0^{I_2} & e_9 &\overset{\text{def}}{=} \ell_2^{I_3} \\ e_5 &\overset{\text{def}}{=} \ell_1^{I_2} & e_{10} &\overset{\text{def}}{=} \ell_3^{I_3} \end{aligned}$$

We know that we have to compute $\mathcal{A}(e_i, e_j)$ in order to fill the matrix that represents the linear system we have to solve.

Luckily, a lot of elements will be 0. For example, $\mathcal{A}(e_1, e_9)$ is 0 because

$$\mathcal{A}(e_1, e_9) = \int_a^b e_1' e_9' \, \mathrm{d}x + \int_a^b \sigma e_1 e_9 \, \mathrm{d}x$$

and this is 0 because $e_9$ is 0 on $I_1$ and $e_1$ is 0 on $I_3$ (and both are zero on $I_2$).

We want to understand the sparsity pattern of the matrix $\mathbf{A}$.

We can use the knowledge about where the functions of the basis are not zero to understand where $\mathcal{A}(e_i, e_j)$ is zero.

For example, $e_1$ is different from 0 only the first subinterval $I_1$. This means that, for every function $e_i$ such that $e_i(x) = 0$ for all $x \in I_1$, we have

$$\mathcal{A}(e_1, e_i) = 0$$

and, for symmetry,

$$\mathcal{A}(e_i, e_1) = 0$$

This is true for every $i \in \{5, 6, \dots, 10\}$.

Moreover, it is interesting to note that $\mathcal{A}(e_1, e_3)$ (which, in principle, is not 0) can be computed integrating only on the interval $I_1$. Indeed, we have that

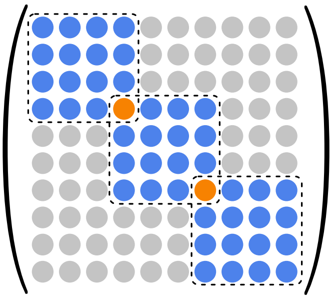
$$\int_a^b e_1' e_3' \,\mathrm{d}x = \int_{I_1} e_1' e_3' \,\mathrm{d}x$$

and

$$\int_a^b \sigma e_1 e_3 \,\mathrm{d}x = \int_{I_1} \sigma e_1 e_3 \,\mathrm{d}x$$

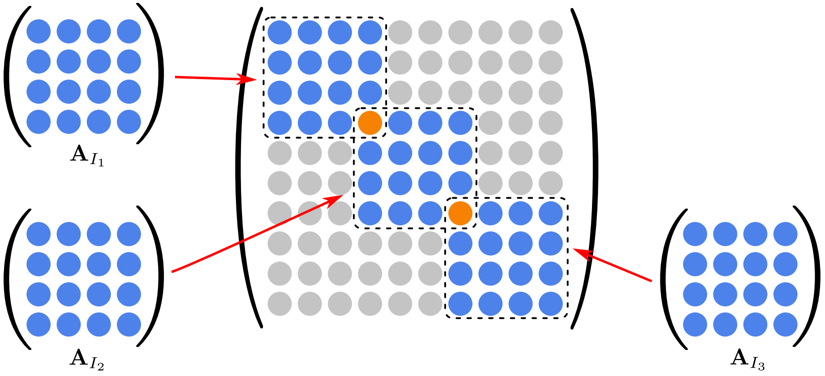
but this is not true for $\mathcal{A}(e_4, e_4)$ because, in that case, we have to integrate on both $I_1$ and $I_2$.

Therefore, we have the following situation



- The gray dots are zeros, obtained integrating two functions that are non–zero on two different intervals;
- The blue dots are values that can be computed integrating our basis functions on one subinterval ($I_1$ if in the first square, $I_2$ if in the second and $I_3$ if in the third);
- The orange dots are values that must be computed integrating on more than one subinterval

All these considerations lead us to implement the following algorithm: instead of building the entire basis $e_1, \ldots, e_n$ we could consider each element (subinterval) and build the local basis $\ell_0, \ldots, \ell_d$ for that element. Then we can compute $\mathcal{A}(\ell_i, \ell_j)$ integrating only on the current subinterval. In some sense, this means that we are building a smaller matrix $\mathbf{A}_{I_h}$ for each element.



The most interesting part is what happens for the orange dots. Indeed, the element $(4,4)$ of the matrix $\mathbf{A}_{I_1}$ is

$$k_1 = \int_{I_1} \left( ({\ell_3}^{I_1})' \right)^2 \, \mathrm{d}x + \int_{I_1} \sigma (\ell_3^{I_1})^2 \, \mathrm{d}x$$

or (because $\ell_0^{I_2}$ is 0 on $I_1$)

$$k_1 = \int_{I_1} (e_4')^2 \, \mathrm{d}x + \int_{I_1} \sigma e_4^2 \, \mathrm{d}x$$

On the other side, the first element of $\mathbf{A}_{I_2}$ is

$$k_2 = \int_{I_2} (e_4')^2 \, \mathrm{d}x + \int_{I_2} \sigma e_4^2 \, \mathrm{d}x$$

and, therefore (taking into account that $e_4$ is zero on $I_3$)

$$k_1 + k_2 = \int_a^b (e_4')^2 \, \mathrm{d}x + \int_a^b \sigma e_4^2 \, \mathrm{d}x = \mathcal{A}(e_4, e_4)$$

We have shown that, if we have built the matrices $\mathbf{A}_{I_h}$, the values of the orange dots inside $\mathbf{A}$ can be obtained summing the corresponding values of the matrices $\mathbf{A}_{I_h}$.

Let us summarize again our algorithm:

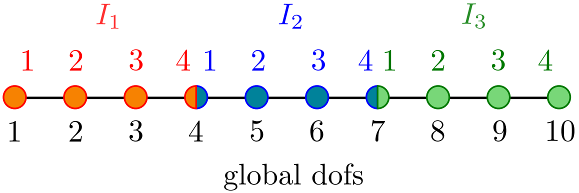
- Allocate a matrix $\mathbf{A}$ (the global matrix) with all entries set to 0;
- Go element by element (subinterval by subinteval, in our case) and assemble a small matrix $\mathbf{A}_{I_h}$ (the local matrix) using the lagrangian basis of the element;
- Add the values of the local matrix in the corresponding entries of the global one

Of course, the same approach is also valid to compute $\mathcal{F}(e_i)$. Assemble a local vector for each element and copy its values inside the corresponding entries of a global one.

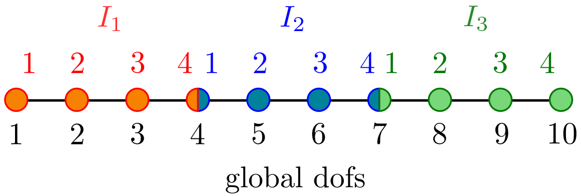
But how do we recognize the “corresponding” entries?

We need to perform an operation that is usually called "numbering the dofs".

We take all the lagrangian nodes that we have defined in each subinterval. Of course, the nodes that are on the boundaries are shared between two intervals.



These nodes are called *degrees of freedom* (DOFs) of the system. Indeed, we have constructed our basis so that, for each node $x_i$ there exists one function of the basis $e_i$ such that $e_i(x_i) = 1$ and $e_i(x_j) = 0$ for all the other nodes $x_j$. In other words, for each node there exists one element of the basis.



What we need to do is to store a map for each $I_h$. So, for example, for the subinterval $I_2$ we need to remember that the first node of the interval is the 4th degree of freedom of the global system (and so on). In this way, we can assemble the global matrix simply adding the values of the local matrix $\mathbf{A}_{I_h}$ to the corresponding indices of the global matrix (so, in this case, the index 1 of the local matrix is mapped to the index 4 of the global one).

Here I numbered starting from 1. Of course, in Python, you must start from 0.

For each element, the entries of the local matrix are

$$\left(\mathbf{A}_{I_h}\right)_{ij} = \mathcal{A}_{I_h}(\ell_j, \ell_i) = \int_{I_h} \ell_j' \ell_i' \,\mathrm{d}x + \int_{I_h} \sigma \ell_j \ell_i \,\mathrm{d}x$$

To compute these entries, we need to use a quadrature formula. Therefore, we need to define some quadrature nodes $q_1, \ldots, q_p$ and some weights $w_1, \ldots, w_p$ for each interval $I_h$. Then, the previous expression can be approximated by

$$\left(\mathbf{A}_{I_h}\right)_{ij} \approx \sum_{k=1}^{p} w_k(\ell_j(q_k)' \ell_i(q_k)' + \sigma(q_k)\ell_j(q_k)\ell_i(q_k))$$

For each interval $I_h$ we need to compute

- The position of the quadrature points $q_k$ and the weights $w_k$
- The value of each lagrangian polynomial $\ell_i$ evaluated on each quadrature point $q_k$: $\ell_i(q_k)$
- The value of the derivative of each lagrangian polynomial $\ell_i$ evaluated on each quadrature point $q_k$: $\ell_i'(q_k)$
- The value of $\sigma(q_k)$ on each quadrature point $q_k$

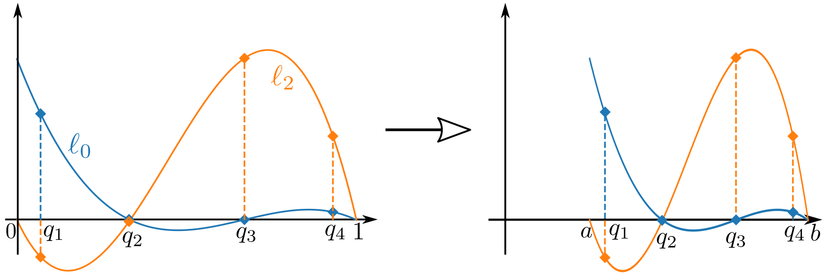
Luckily, there is a better way than recompute all these quantities for every element. Indeed, beside the last point (the one that requires to compute $\sigma(q_k)$), all the other quantities can be computed once and then be "translated" from one element to the other.

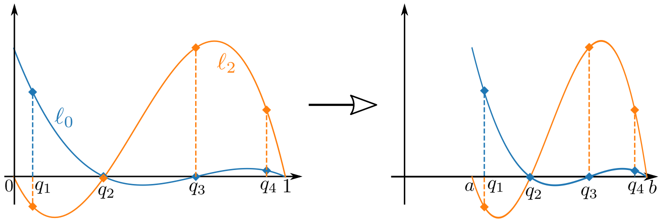
We will then use a “reference element”. This is, we compute the following quantities

- The position of the quadrature points $q_k$ and the weights $w_k$
- The value of each lagrangian polynomial $\ell_i$ evaluated on each quadrature point $q_k$: $\ell_i(q_k)$
- The value of the derivative of each lagrangian polynomial $\ell_i$ evaluated on each quadrature point $q_k$: $\ell_i'(q_k)$

on the interval $[0, 1]$ (for example), even if this interval is not one of the elements of our domain. Then, for every element, we can just move the previous quantities from the interval $[0, 1]$ to the actual one in the following way

The following picture shows the interval $[0, 1]$ with two basis functions (as an example, I chosed $\ell_0$ and $\ell_2$) that we have evaluated on the quadrature points (the dashed lines). On the right there is another interval $[a, b]$ for which we want to compute the same quantities



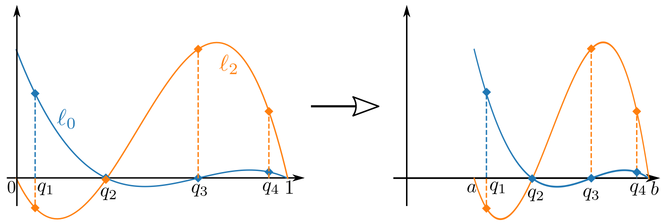


- The position of the quadrature points can be computed by an affine transformation

$$\tilde{q}_i = a + (b - a)q_i$$

- The new weights can be computed by multiply the old ones by the size of the new element

$$\tilde{w}_i = (b - a)w_i$$



- The value of the basis functions on the quadrature points is the same

$$\tilde{\ell}_j(\tilde{q}_i) = \ell_j(q_i)$$

- The value of the derivative of the basis functions on the quadrature points must be divided by the size of the element (can you tell why?)

$$\tilde{\ell}_j{}'(\tilde{q}_i) = \frac{\ell_j'(q_i)}{b-a}$$

Now we have all the ingredients to summarize the algorithm that we want to implement in Python. Starting from the weak formulation of your problem

- Define some elements on your domain, splitting it in several subintervals
- Decide some parameters of your methods: what degree are you going to use for the lagrangian basis? What quadrature rule?
- Take a reference element and compute all the quantities that you will need afterwards for your "real" elements
- Number the degrees of freedom
- Allocate the global matrix **A** and the global right hand side vector **F**
- Go element by element, assemble a local matrix and add its entries to the global matrix. To the same for the right hand side vector
- Solve the linear system

That’s all! Time for some Python!