Numerical Methods for Partial Differential Equations

Andrea Dal Prete

Abstract

The aim of this document is to provide some mathematical tools in the field of numerical mathematics useful to approach partial differential equations. Some code in Python will be provided as well in order to concretely show how these methods can be implemented in order to solve partial differential equations and build high accuracy numerical methods.

Contents

1 Introduction

Since the very beginning of time when human beings started to walk on earth they tried to create models able to describe the world surrounding them and to eventually make predictions on what could have happened in the future. Therefore, starting from the basic logical concepts they passed through mathematical models of different complexity following the purpose to describe the physical phenomena they observed every day. One sentence pronounced by the British statistician George E. P. Box highlights in a curious way the role of these mathematical methods used in engineering applications to make predictions: "All models are wrong, but some are useful". In this sentence is contained the meaning of all the tools used in scientific fields which, actually, are never able to perfectly describe the reality, but are sometimes useful to find approximate solutions leading to predictions with the needed accuracy level in specific applications. As a matter of fact, the affinity grade of the solution with the real behavior of a system usually increases with the complexity of the used mathematical model, and along time, starting from the linear mathematics, passing through the ordinary differential equations, more detailed models have been formulated: the Partial Differential Equations (PDE). These analytical tools are widely used nowadays to make accurate predictions in many engineering fields (i.e. aerospace, fluidynamics, turbomachinery, the definition of the deformation field in extended bodies, etc) and nowadays they represent the fundamentals on which the whole numerical computer simulation theory is built up. Due to the fact that it is usually difficult to find an analytical solution for these equations, until now have been developed many numerical methods allowing to solve partial differential equations from a discrete point of view, and along this document the focus will be placed on two main of these: the *Galerkin Finite Elements Method*, and the *Finite Differences Method*.

2 The Finite Elements Method

Along the history of science the need to find more accurate mathematical models to reach precise solution in some engineering applications started to be more and more urgent, in order to better describe the engineering systems and to increase their efficiency and the forecasting of their working conditions. For this reason many models based on partial differential equations were developed, as these mathematical tools take into account the space extension of a mechanical body and its deformation in space, instead of considering it as a single point or a rigid body, increasing therefore the capacity of the model to describe the actual behaviour of the system. Unfortunately, to solve the PDE is usually a difficult challenge, since to find their analytical solutions represents a problem of higher complexity. Therefore, along the 19^{th} century many numerical methods able to find an approximate solution of these equations were developed, usually based on the studies carried on by the Russian engineer Boris Galerkin, which came out with a new mathematical concept allowing to approach PDE from a numerical point of view.

2.1 The Galerkin Problem (GP)

Find:

$$u_h \in V_h : \quad a(u_h, v_h) = F(v_h) \quad \forall v_h \in V_h$$

Where v_h is a test function defined such that u_h and v_h belong to the same space V_h. Therefore, passing through the approximation of the V space (infinite dimensional Hilbert space) with a V_h finite dimensional one such that $\dim(V_h) = N_h$, the following relations come out:

$$v_h(x) = \sum_{j=1}^{N_h} v_j \varphi_j(x)$$

and

$$u_h(x) = \sum_{j=1}^{N_h} u_j \varphi_j(x)$$

2.2 The Galerkin Finite Element Method

Given $a(u, v) = F(v)$ and having introduced a discretization of the Hilbert space passing from a continuous to a discrete one, we reach the following results:

with $i = 1, ...N_h$

$$a(\sum_{j=1}^{N_h} u_j \varphi_j(x), \varphi_i(x))v_i = F(\varphi_i(x))v_i$$

Hence:

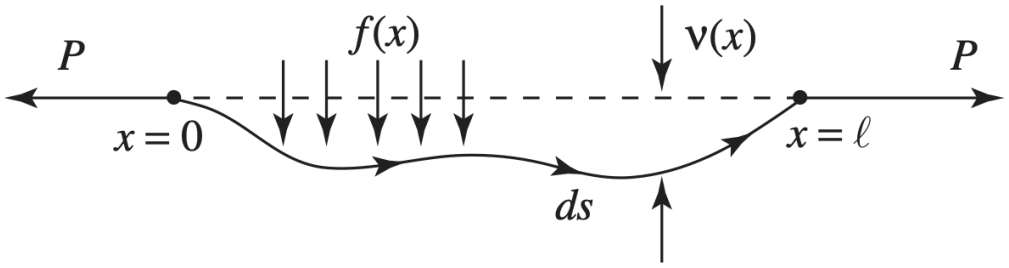
$$\sum_{j=1}^{N_h} u_j \; a(\varphi_j(x), \varphi_i(x)) = F(\varphi_i(x))$$

Therefore, $(A)_{ij}$ comes out to be the ij element of the $[A]$ matrix and $(f)_i$ the i element of the $\underline{F}$ vector; and is finally stated that the Galerkin Method reduces the starting problem (GP), representing the weak formulation of a partial differential equation, to the resolution of a linear system such that:

$$[A]\underline{u} = \underline{F}$$

2.3 Examples

2.3.1 The string Problem



The elastic string.

The string in figure might be well described by a second order partial differential equation as follows:

$$\frac{\partial^2 u}{\partial t^2} - \frac{1}{a^2}\frac{\partial^2 u}{\partial x^2} = f(t,x) \quad with \quad u = u(t,x)$$

The one reported above coincide with the wave equation in the variable $u = u(t,x)$, which represents the displacement along the y axis as a function of time t and space x. Thinking of the system as one which is exited by a distributed force constant in time and space and considering that has passed enough time to reach the steady-state equilibrium condition, the wave equation is reduced to a second order ordinary differential equation as follows:

$$-\frac{1}{a^2}\frac{\partial^2 u}{\partial x^2} = f(x)$$

which adding the boundary conditions comes out to be:

$$\begin{cases} \mu\ u''(x) = -f(x) \\ u = 0 \quad \forall x \in [a;b] \end{cases}$$

By making some considerations is now possible to formulate the weak problem which enables then to build the linear system making reference to the Galerkin Finite Element Method. Thus, starting by multiplying each member of the equation for a test function (v) such that:

$$u, v \in \ V = H^1\ (u, v = 0\ \forall x\ \ \in [a;b])$$

and making an integration on the domain of the equation:

$$\mu \int_\Omega u'' v\, dx = -\int_\Omega f v\, dx$$

$$\mu \int_\Omega u'v'\, dx - \mu\ u'v|_{\partial\Omega} = -\int_\Omega f(x)\ v\, dx$$

where since u, v have been chosen such that to be equal to zero at the boundaries:

$$u'v|_{\partial\Omega} = 0$$

Hence:

$$\mu \int_\Omega u'v'\, dx = -\int_\Omega f(x)\ v\, dx$$

The equation above represents the weak formulation of the stationary string problem, where the former part of the equation is called *bilinear form*, the latter *linear form*, such that:

$$A(u,v) = F(v)$$

Thanks to this weak formulation of the problem, the passage through a discretization of the spaces is now possible, allowing finally to formulate the Galerkin Method. Therefore the discrete domain is considered as:

$$x = [0,1], \quad N_h = 5,\ h = \frac{1-0}{N_h}$$

And the linear system is built as follows:

$$\begin{bmatrix} \int_0^h \varphi_1'\varphi_1'\,dx & \dots & \int_0^h \varphi_1'\varphi_{N_h}'\,dx \\ \dots & \dots & \dots \\ \int_0^h \varphi_{N_h}'\varphi_1'\,dx & \dots & \int_0^h \varphi_{N_h}'\varphi_{N_h}'\,dx \end{bmatrix} \begin{bmatrix} u_1 \\ \dots \\ u_{N_h} \end{bmatrix} = - \begin{bmatrix} \int_0^h f\varphi_1\,dx \\ \dots \\ \int_0^h f\varphi_{N_h}\,dx \end{bmatrix}$$

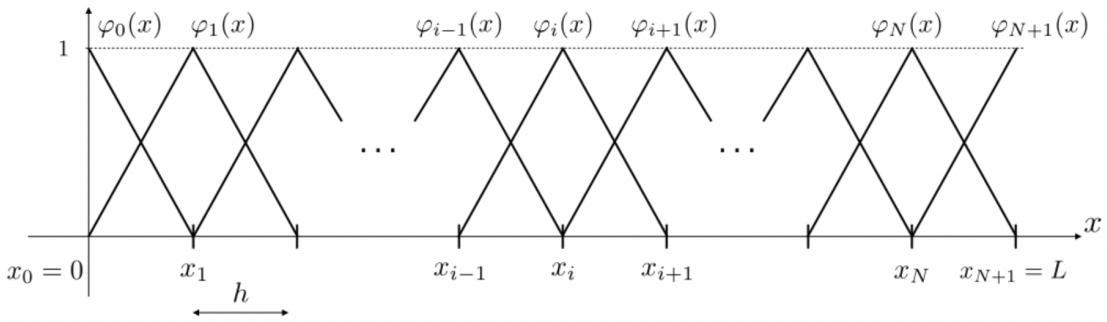
$$([A]\underline{u} = \underline{F})$$

where:

- $[A]$ is the stiffness matrix
- $\underline{u}$ is the approximated solution
- $\underline{F}$ is the discrete vector of the force

$$\varphi_{ij}$$

are the shape functions having the trend with which we choose to approximate the solution between the point of the discretized space. In this document the Lagrange polynomials of degree 1 have been chosen, but to reach more precise solutions one could even use either higher order polynomial functions or periodic functions as sinusoidal or cosinusoidal ones.



Linear shape functions.

Follows the analytical formulation of the linear shape functions:

$$\varphi_0(x) = \begin{cases} \frac{x_1 - x}{h} & x_0 < x < x_1 \\ 0 \quad else \end{cases}$$

$$\varphi_i(x) = \begin{cases} \frac{x - x_{i-1}}{h} & x_{i-1} < x < x_i \\ \frac{x_{i+1} - x}{h} & x_i < x < x_{i+1} \\ 0 \quad else \end{cases}$$

$$\varphi_{N_h+1}(x) = \begin{cases} \frac{x - x_{N_h}}{h} & x_n < x < x_{N_h+1} \\ 0 \quad else \end{cases}$$

And since in our case the domain has been defined in $[0, l] = [0, 1]$ it is stated that $x_0 = 0$, $x_1 = h$, ... $x_{N_h+1} = l = 1$, and the shape function derivatives come out to be:

$$\varphi_0'(x) = \begin{cases} -\frac{1}{h} \\ 0 \end{cases}$$

$$\varphi_i'(x) = \begin{cases} \frac{1}{h} \\ -\frac{1}{h} \\ 0 \end{cases}$$

$$\varphi_{N_h+1}'(x) = \begin{cases} \frac{1}{h} \\ 0 \end{cases}$$

Therefore, going up to the point where the resolution system was explained and considering its formulation, it is now possible to build the stiffness matrix and the vector of forces acting on the system, being:

$$[A]\underline{u} = \underline{F}$$

where:

$$[A] = \frac{\mu}{h} tridiag(-1, 2, -1) \quad and \quad F_i = hf(x_i)$$

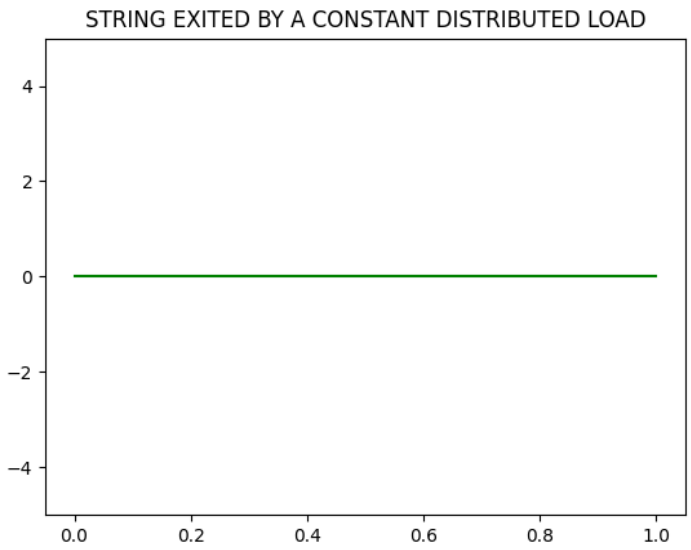
Consequently the string problem could be solved by means of a computational implementation, making use of Python for example, and in that case the script code would come out like this:

```
"""
@ author:   Andrea Dal Prete
@ copyright
"""

# Finite Element Method with linear shape functions
# Soluction of the static string problem exited by a constant distributed load

import numpy as np
import matplotlib.pyplot as plt

# DATA
L   = 1
mu  = 2
N   = 100
f   = -50

# BUILDING THE SYSTEM
h = (L-0)/(N+1)

# stiffness matrix
K = (mu/h)*(np.diag(np.ones(N-1)*2) + np.diag(-1*np.ones(N-2),1) + np.diag(-1*np.ones(
    N-2),-1))

# global matrix
A = K

# vector of forces
f = h*f*np.ones(N-1)

# building the vectors and the system's solution
uh = np.dot(np.linalg.inv(A),f)
u  = np.zeros(N+1)

for i in range(1,N):
```

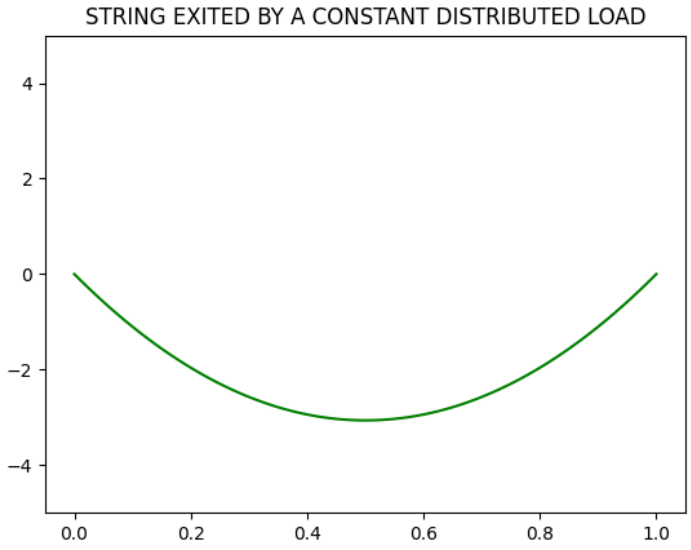
```
    u[i] = uh[i-1]

x  = np.linspace(0,L,N+1)

# PLOT
plt.figure(1)
plt.plot(x, u, color = 'green')
plt.ylim((-5,5))
plt.title("STRING EXITED BY A CONSTANT DISTRIBUTED LOAD")
plt.show()
```

Listing 1: Python example

And the results of the implementation are:



f=0



f≠0

2.3.2 Loaded plate

Moving now through a 2D space domain instead of staying in the 1D one, means to no longer deal with a string but with a plate, which is described by the Poisson equation reported below:

$$\begin{cases} \nabla^2 u = -\frac{f}{\mu} \\ u(x,y) = 0 \quad \forall (x,y) \in \partial\Omega \end{cases}$$

As made for the 1D case, the formulation of the weak problem is computed as follows:

$$\mu \int_\Omega \nabla^2 u \; v \, d\Omega = -\int_\Omega f \; v \, d\Omega$$

$$\mu \int_\Omega \nabla u \nabla v \, d\Omega - \mu \nabla u \; v|_{\partial\Omega} = -\int_\Omega f \; v \, d\Omega$$

where since u, v have been chosen such that to be equal to zero at the boundaries:

$$\nabla u \; v|_{\partial\Omega} = 0$$

Hence:

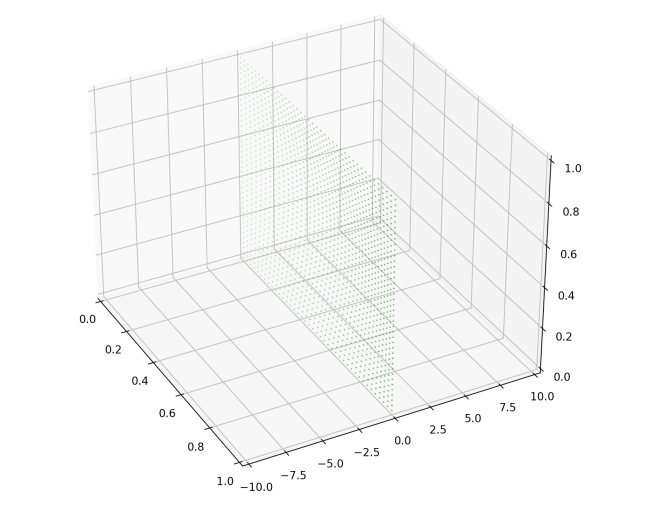
$$\mu \int_\Omega \nabla u \nabla v \, d\Omega = -\int_\Omega f \; v \, d\Omega$$

Same as previously indicated, the passage through discrete spaces such that u_h,$v_h \in \mathrm{V}_h$ is considered, and the final formulation comes out to be:

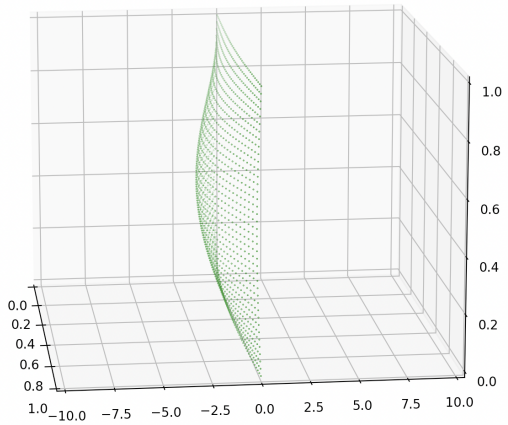
$$\begin{bmatrix} \int\int \nabla\varphi_1(x,y)\nabla\varphi_1(x,y)\,dxdy & \dots & \int\int \nabla\varphi_1(x,y)\nabla\varphi_{N_h}(x,y)\,dxdy \\ \dots & \dots & \dots \\ \int\int \nabla\varphi_{N_h}(x,y)\nabla\varphi_1(x,y)\,dxdy & \dots & \int\int \nabla\varphi_{N_h}(x,y)\nabla\varphi_{N_h}(x,y)\,dxdy \end{bmatrix} \begin{bmatrix} u_1 \\ \dots \\ u_{N_h} \end{bmatrix}$$

$$= -\begin{bmatrix} \int\int f_1\varphi_1(x,y)\,dxdy \\ \dots \\ \int\int f_{N_h}\varphi_{N_h}(x,y)\,dxdy \end{bmatrix}$$

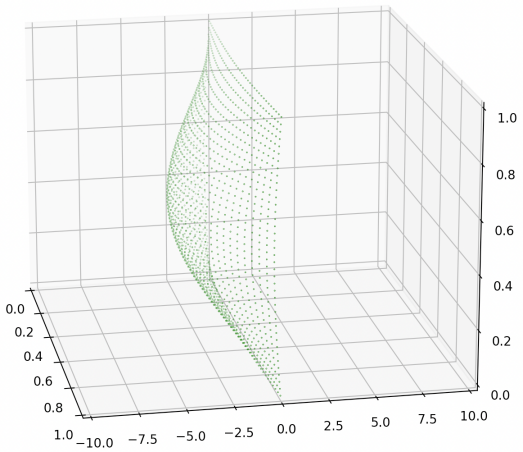
In order to find a numerical solution in the 2D case, it is necessary to make use of shape functions of the type $\varphi = \varphi(x,y)$ allowing to build the matrices and to pass from the weak to the numerical formulation. As a last step a Python code might be implemented to find the actual solution of the problem.
For the sake of simplicity only a plot of the numerical solution of the equation has been reported below to show the physical meaning of the Poisson equation and a possible numerical solution:



$f=0$



$f\neq 0$



$f\neq 0$

3 The Finite Differences Method

The *Finite Differences Method* is a good alternative to the *FEM* one, although in general it may lead to numerical instability-related problems, and it doesn't allow to improve the accuracy of the model, differently from the *FEM* case for which is possible instead, for instance by means of using higher order polynomials as shape functions. Anyway it represents a good tool to make a first good numerical approximation. The basis of this method consist in considering the derivative operator by means of a discrete difference instead of a continuous one at the limit.

3.1 Formulation of the method

Therefore, starting from the definition of derivative:

$$f'(x) = \lim_{dx \to 0} \frac{f(x+dx) - f(x)}{dx}$$
$$f''(x) = \lim_{dx \to 0} \frac{f'(x+dx) - f'(x)}{dx}$$

come out to be:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$
$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

Where for the sake of simplicity $f(x+h) = f_{i+1}$ and $f(x-h) = f_{i-1}$.

Therefore, since the function $u = u(x)$ is approximated with a discrete vector of values $\underline{u}_h = [u_0, u_1, u_2, ...u_{N_h}]^T$, the following steps come out as a consequence of the assumptions above:

$$\frac{\partial u}{\partial x} \approx [\frac{u_2 - u_0}{2h}, ..., \frac{u_{i+1} - u_{i-1}}{2h}, ..., \frac{u_{N_h+1} - u_{N_h-1}}{2h}]^T =$$
$$[\frac{u_2}{2h}, ..., \frac{u_{i+1} - u_{i-1}}{2h}, ..., \frac{-u_{N_h-1}}{2h}]^T - \frac{1}{2h}[u_0, ... - u_{N_h+1}]^T$$

Finally:

$$\frac{\partial u}{\partial x} \approx [\frac{u_2}{2h}, ..., \frac{u_{i+1} - u_{i-1}}{2h}, ..., \frac{-u_{N_h-1}}{2h}]^T - \underline{C} = [D_x]\underline{u}_h - \underline{C}$$

Which can be written in matrix form as follows:

$$[D_x]\underline{u_h} = \frac{1}{2h}\begin{bmatrix} 0 & 1 & 0 & ... & ... & ... & ... & 0 \\ -1 & 0 & 1 & 0 & ... & ... & ... & 0 \\ 0 & -1 & 0 & 1 & 0 & ... & ... & 0 \\ ... & ... & ... & ... & ... & ... & ... & ... \\ 0 & ... & ... & ... & ... & ... & -1 & 0 \end{bmatrix}\begin{bmatrix} u_1 \\ ... \\ ... \\ u_{N_h} \end{bmatrix}$$

The same passages might be followed to build the approximation of the second order derivative:

$$\frac{\partial^2 u}{\partial x^2} \approx [\frac{u_2 - 2u_1 + u_0}{h^2}, ..., \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2}, ..., \frac{u_{N_h+1} - 2u_{N_h} + u_{N_h-1}}{h^2}]^T =$$
$$[\frac{u_2 - 2u_1}{h^2}, ..., \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2}, ..., \frac{-2u_{N_h} + u_{N_h-1}}{h^2}]^T - \frac{1}{h^2}[-u_0, ... - u_{N_h+1}]^T$$

Finally:

$$\frac{\partial^2 u}{\partial x^2} \approx [\frac{u_2}{2h}, ..., \frac{u_2 - 2u_1}{h^2}, ..., \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2}, ..., \frac{-2u_{N_h} + u_{N_h-1}}{h^2}]^T - \underline{C} = [D_{xx}]\underline{u_h} - \underline{C}$$

Which can be written in matrix form as follows:

$$[D_{xx}]\underline{u_h} = \frac{1}{h^2}\begin{bmatrix} -2 & 1 & 0 & ... & ... & ... & ... & 0 \\ 1 & -2 & 1 & 0 & ... & ... & ... & 0 \\ 0 & 1 & -2 & 1 & 0 & ... & ... & 0 \\ 0 & ... & 1 & -2 & 1 & ... & ... & ... \\ ... & ... & ... & ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... & ... & 1 & -2 \end{bmatrix}\begin{bmatrix} u_1 \\ ... \\ ... \\ ... \\ ... \\ u_{N_h} \end{bmatrix}$$

Similar passages might be followed to build a matricial approximation of the fourth derivative of the u function:

$$\frac{\partial^4 u}{\partial x^4} \approx \frac{4}{h^4}[u_4 - 4u_3 + 6u_2 - 4u_1 + u_0, ...,$$
$$u_{i+2} - 4u_{i+1} + 6u_i - 4u_{i-1} + u_{i-2}, ...,$$
$$u_{N_h+2} - 4u_{N_h+1} + 6u_{N_h} - 4u_{N_h-1} + u_{N_h-2}]^T$$
$$...$$

Reaching after the same passages as before the matrix form as follows:

$$[D_{4x}]\underline{u_h} = \frac{4}{h^4}\begin{bmatrix} 6 & -4 & 1 & ... & ... & ... & ... & 0 \\ -4 & 6 & -4 & 1 & ... & ... & ... & 0 \\ 1 & -4 & 6 & -4 & 1 & ... & ... & 0 \\ 0 & ... & -4 & 6 & -4 & ... & ... & ... \\ ... & ... & ... & ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... & ... & -4 & 6 \end{bmatrix}\begin{bmatrix} u_1 \\ ... \\ ... \\ ... \\ ... \\ u_{N_h} \end{bmatrix}$$

And so on.

Finding out these matrix formulations to write the approximations of the first, second, ... n^{th} order derivatives by means of finite differences enables to build a numerical approximation of an analytical problem based on partial differential equations. For instance, consider the following *diffusion-transport-reaction* problem in the function $u = u(t, x)$:

$$\frac{\partial u}{\partial t} + \mu \frac{\partial^2 u}{\partial x^2} + \alpha \frac{\partial u}{\partial x} + \beta u = 0$$

By means of finite differences the problem is formulated as follows:

$$\frac{\underline{u}^{t+1} - \underline{u}^t}{ht} + \mu [D_{xx}]\underline{u}^t + \alpha [D_x]\underline{u}^t + \beta \underline{u}^t - \underline{C} = 0$$

$$\underline{u}^{t+1} = [[I] - h_t\ \mu[D_{xx}] - h_t\ \alpha[D_x] - h_t\ \beta[I]]\underline{u}^t + h_t\ \underline{C}$$

3.2 Examples

3.2.1 The Beam Equation

Similarly to the string case, by means of mathematical formulations a partial differential equation describing the behaviour of the slender beam might be formulated as follows:

$$m \frac{\partial^2 u}{\partial t^2} + EJ \frac{\partial^4 u}{\partial x^4} = -f$$

This equation describes the dynamic of a slender beam characterized by a Young modulus E and an inertial momentum J, where the fourth derivative with respect to space represents the fact that four boundary conditions are needed to completely describe the behavior of the body, corresponding to the two displacements and the two rotations at the boundaries. If we wanted to study instead how the beam behaves in steady-state condition under the effect of a constant load, the previous equation reduces to:

$$EJ \frac{\partial^4 u}{\partial x^4} = -f$$

Which by means of finite differences might be written as:

$$EJ[D_{4x}]\underline{u} = -\underline{f}$$

Where:

$$[D_{4x}] = \frac{4}{h^4} \begin{bmatrix} 6 & -4 & 1 & ... & ... & ... & ... & 0 \\ -4 & 6 & -4 & 1 & ... & ... & ... & 0 \\ 1 & -4 & 6 & -4 & 1 & ... & ... & 0 \\ 0 & ... & -4 & 6 & -4 & ... & ... & ... \\ ... & ... & ... & ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... & ... & -4 & 6 \end{bmatrix}$$

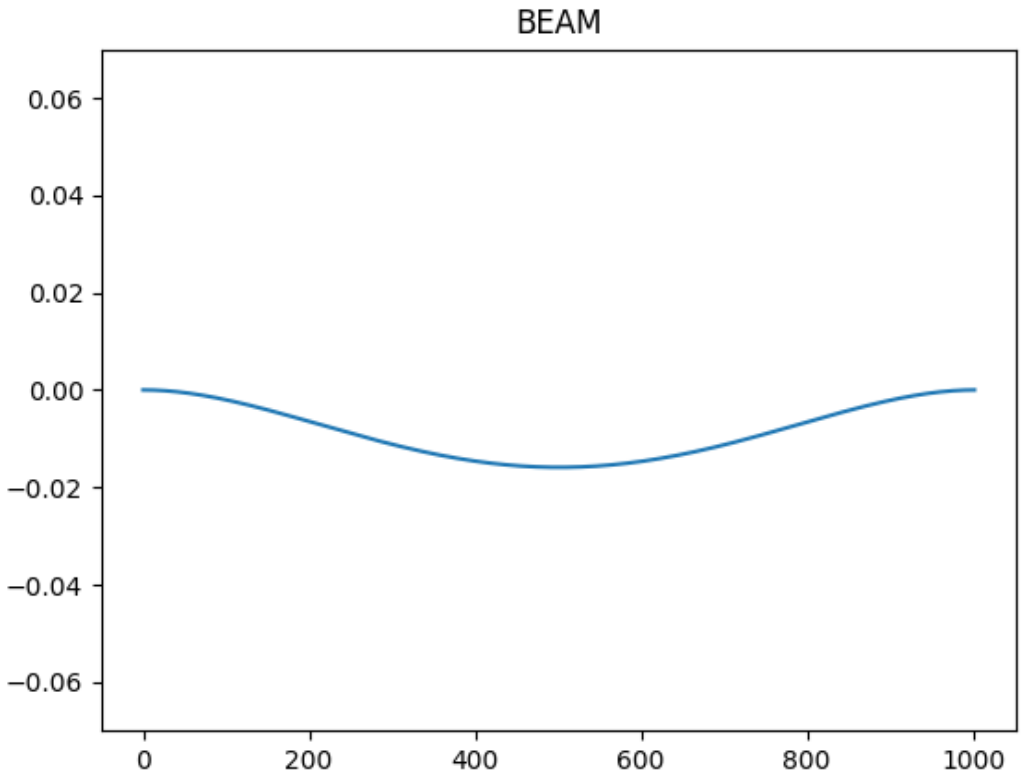
As a matter of fact, taking into account the formulation of the equation describing the system behavior by means of Finite Differences, a Python code can be implemented to find an approximated numerical solution of the problem as follows:

```
"""
@ author:   Andrea Dal Prete
@ copyright
"""

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# definition of the approximated fourth derivative of u with respect of x by means of
    finite differences

def der_x(n):
    vn     = np.ones(n)
    vd     = -4*vn
    A      = np.diag(vn[0:-2],-2) + np.diag(vd[0:-1],-1) + 6*np.diag(vn) + np.diag(vd
    [0:-1],1) + np.diag(vn[0:-2],2)
    return A

# definition of the constant force acting on the system

def vect_g_D(n):                        # distributed force
    return pow(10,4.7)*np.ones(n)

# definition of the boundary conditions

def vect_C(n,a,h,uf,ui,u_ii,u_ff):
    C     = np.zeros(n)
    C[0]  = -ui*a*4/(h**4) + 4*u_ii*a*4/(h**4)
    C[-1] = -uf*a*4/(h**4) + 4*u_ff*a*4/(h**4)
    return C

# solver definition

def solve(h,A,n,a,ui,uf,u_ii,u_ff):
    u = np.zeros(n)
    u = np.dot(np.linalg.inv(A),vect_C(n,a,h,uf,ui,u_ii,u_ff) - vect_g_D(n))

    return u

# DATA

n     = 1000
L     = 1
h     = L/(n+1)
ui    = 0
u_x_i = 0
uf    = 0
u_x_f = 0
u_ii  = ui + u_x_i*h
u_ff  = uf - u_x_f*h
a     = 206*10 #(EJ)

# RESOLUTION

u  = np.zeros(n+3)
u[0]   = ui
u[1]   = u_ii
u[n]   = uf
u[n+1] = u_ff

A = a*4*(der_x(n)/(h**4))
uh = solve(h,A,n,a,ui,uf,u_ii,u_ff)

```

```
for i in range(2,n+1):
    u[i] = uh[i-2]

# plot
plt.title("BEAM")
plt.plot(u)
plt.ylim((-0.07,0.07))
plt.show()
```

Listing 2: Python example

Giving as an output the following result:



3.2.2 The Navier-Stokes Equations

The Navier-Stokes equations represents a system of 4 nonlinear, elliptic and chaotic partial differential equations, and are reported below:

$$\begin{cases} \rho \frac{\partial \vec{v}}{\partial t} + \rho\ \vec{v} \cdot \nabla \vec{v} = -\nabla p_e + \mu \nabla^2 \vec{v} \\ \nabla \cdot \vec{v} = 0 \end{cases}$$

where:

- $\vec{v} \cdot \nabla \vec{v}$ represents the non-linearity
- $\nabla^2 \vec{v}$ represents the chaos
- and since the boundary conditions are function of the problem's solution, the equations come out to be implicit.

As always it is possible to make some considerations on the equations in order to understand whether to consider or not specific terms and to possibly find particular solutions, starting for instance by considering their non-dimensional formulation:

$$\frac{\partial \widehat{\vec{v}}}{\partial \widehat{t}} + \widehat{\vec{v}} \cdot \widehat{\nabla} \widehat{\vec{v}} = -\widehat{\nabla} \widehat{p_e} + \frac{1}{R_e} \widehat{\nabla}^2 \widehat{\vec{v}}$$

Where R_e is the Reynolds number, defined as $R_e = \frac{\rho v D}{\mu}$. Therefore it can be noticed that for high number of R_e the term responsible for the chaotic part of the equations can be neglected. Moreover in applications where the Reynolds number is very high and the flow can be assumed to be close to a laminar one, in that specific domain the Euler equations represent a good approximation of the air flow behavior:

$$\rho \frac{\partial \vec{v}}{\partial t} + \rho \ \vec{v} \cdot \nabla \vec{v} = -\nabla p_e$$

$$\frac{\partial \vec{v}}{\partial t} + \vec{v} \cdot \nabla \vec{v} = -\frac{1}{\rho} \nabla p_e$$

In order to proceed with the analysis, the components of the equation are made explicit:

$$\frac{\partial \vec{v}}{\partial t} = \begin{bmatrix} \frac{\partial v_x}{\partial t} \\ \frac{\partial v_y}{\partial t} \\ \frac{\partial v_z}{\partial t} \end{bmatrix}$$

$$\nabla \vec{v} = \begin{bmatrix} \frac{\partial v_x}{\partial x} & \frac{\partial v_x}{\partial y} & \frac{\partial v_x}{\partial z} \\ \frac{\partial v_y}{\partial x} & \frac{\partial v_y}{\partial y} & \frac{\partial v_y}{\partial z} \\ \frac{\partial v_z}{\partial x} & \frac{\partial v_z}{\partial y} & \frac{\partial v_z}{\partial z} \end{bmatrix}$$

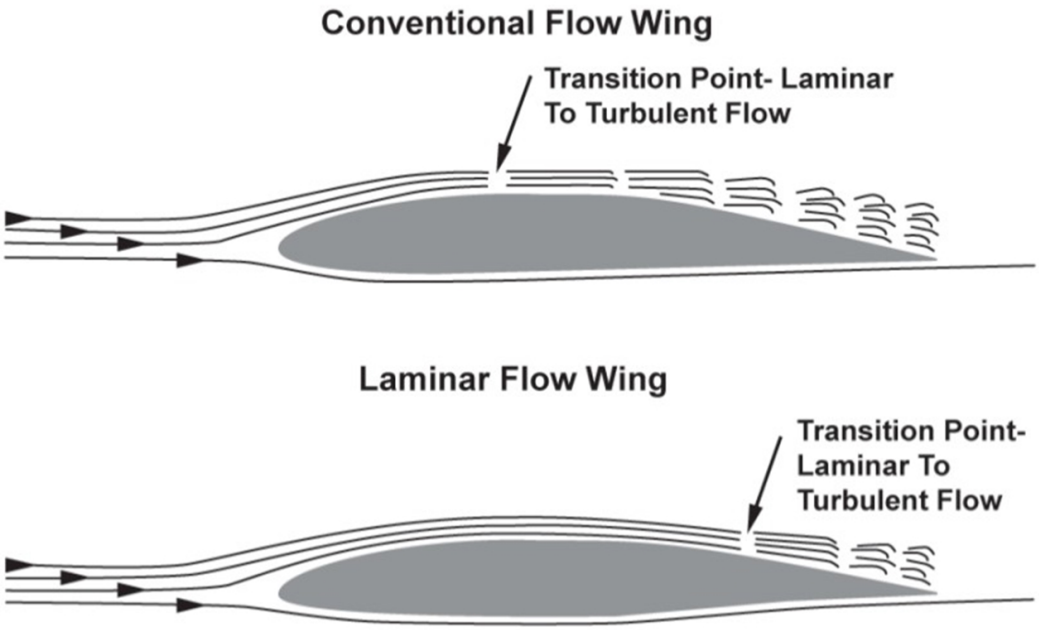
$$\vec{v} \cdot \nabla \vec{v} = \begin{bmatrix} v_x \frac{\partial v_x}{\partial x} + v_y \frac{\partial v_x}{\partial y} + v_z \frac{\partial v_x}{\partial z} \\ v_x \frac{\partial v_y}{\partial x} + v_y \frac{\partial v_y}{\partial y} + v_z \frac{\partial v_y}{\partial z} \\ v_x \frac{\partial v_z}{\partial x} + v_y \frac{\partial v_z}{\partial y} + v_z \frac{\partial v_z}{\partial z} \end{bmatrix}$$

$$\nabla^2 \vec{v} = \nabla \cdot \nabla \vec{v} = \begin{bmatrix} \nabla^2 v_x \\ \nabla^2 v_y \\ \nabla^2 v_z \end{bmatrix}$$

Therefore it is possible now to make explicit all the 4 equations of the general Navier-Stokes equations' system:

$$\begin{cases} \rho \frac{\partial v_x}{\partial t} + \rho (v_x \frac{\partial v_x}{\partial x} + v_y \frac{\partial v_x}{\partial y} + v_z \frac{\partial v_x}{\partial z}) = -\frac{\partial p_e}{\partial x} + \mu \nabla^2 v_x \\ \rho \frac{\partial v_y}{\partial t} + \rho (v_x \frac{\partial v_y}{\partial x} + v_y \frac{\partial v_y}{\partial y} + v_z \frac{\partial v_y}{\partial z}) = -\frac{\partial p_e}{\partial y} + \mu \nabla^2 v_y \\ \rho \frac{\partial v_z}{\partial t} + \rho (v_x \frac{\partial v_z}{\partial x} + v_y \frac{\partial v_z}{\partial y} + v_z \frac{\partial v_z}{\partial z}) = -\frac{\partial p_e}{\partial z} + \mu \nabla^2 v_z \\ \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z} = 0 \end{cases}$$

The Wing Profile



Wing profile hit by a laminar air flow.

It should be noticed that in many applications it is possible to make several considerations and reasonable hypothesis on the equations. For instance, when considering a very large wing profile, as could be the one of the line airplane reported in the figure above, is generally reasonable to make the hypothesis that the flux of air flowing through the profile doesn't change a lot with respect to the z direction. Furthermore, it is considered that in this application in operating conditions the flux of air can be considered as a laminar flow at least in the boundary layer, and for this reason the absence of turbulence leads to other assumptions such as v_y speed constant with respect to the y coordinate, and velocity vector with y and z components equal to zero.
All these considerations are coupled with the continuity equation ($\nabla \cdot \vec{v} = 0$) to obtain the following mathematical relations:

$$\frac{\partial}{\partial z} = 0 \ , \ \ v_y = v_z = 0 \ , \ \ \frac{\partial v_y}{\partial y} = \frac{\partial v_z}{\partial z} = 0$$

Moreover, considering the continuity equation:

$$\nabla \cdot \vec{v} = 0 \quad and \quad \frac{\partial v_y}{\partial y} = \frac{\partial v_z}{\partial z} = 0 \ \longrightarrow \ \frac{\partial v_x}{\partial x} = 0$$

And from these the following come out:

$$\frac{\partial v_x}{\partial x} = \frac{\partial^2}{\partial^2 z} = \frac{\partial^2 v_y}{\partial^2 y} = \frac{\partial^2 v_x}{\partial^2 x} = 0$$

With these considerations translated in mathematical equations the system of differential equations previously reported reduces to the following single equation:

$$\rho\frac{\partial v_x}{\partial t} - \mu\frac{\partial^2 v_x}{\partial^2 y} = -\frac{\partial p_e}{\partial x}$$

Which represents a partial differential equation in the variable v_x with respect to space (y) and time (t).
Applying now the *Finite Differences Method* as explained before is possible to find a numerical solution of the Navier-Stokes equations in the particular case characterized by the assumptions exploited before:

$$\frac{\underline{v}_{x_h}^{j+1} - \underline{v}_{x_h}^{j}}{h_t} - c[D_{yy}]\underline{v}_h^j - \underline{C} = \underline{f}$$

$$\underline{v}_{x_h}^{j+1} = [[I] + ch_t[D_{yy}]]\underline{v}_{x_h}^{j} + h_t\ \underline{f} + h_t\ \underline{C}$$

With:

$$[D_{yy}] = \frac{1}{h^2}\begin{bmatrix} -2 & 1 & ... & ... \\ 1 & -2 & 1 & ... \\ ... & 1 & -2 & ... \\ ... & ... & ... & ... \end{bmatrix}$$

Consider now that the flow moves at a constant speed v_{y0} sufficiently far away along the y direction, and that at the initial time t_0 the speed profile follows a straight line. Furthermore, at a certain transition point (called A) is registered a periodic vortex detachment, similar to the Karman vortex street phenomenon, leading to a sinusoidal variation of the pressure which will represent the force applied to the system $(-\frac{\partial p_e}{\partial x} = B\ cos(2\pi f_{req} + \varphi))$; the velocity profile is studied just before the point A.
As a matter of fact, taking into account all these considerations and the formulation of the equation describing the system behavior, a Python code can be implemented by means of *Finite Differences Method* to find an approximated numerical solution of the system, either in steady-state or in non-stationary condition.

Here follows the code for the second case:

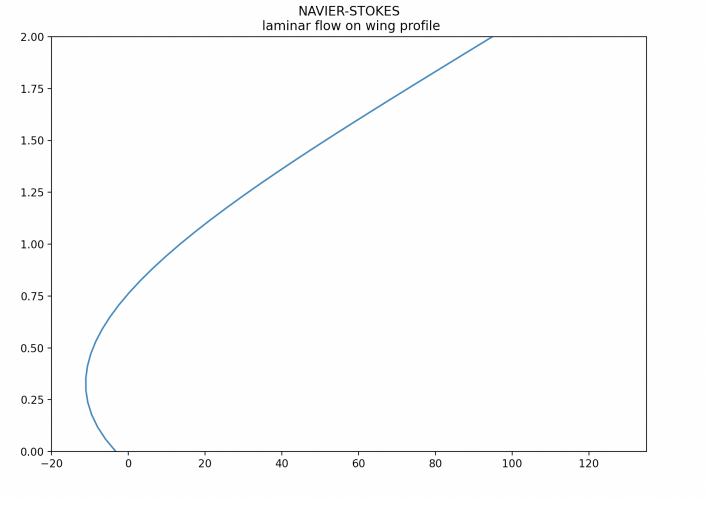
```
"""
@ author:   Andrea Dal Prete
@ copyright
"""

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def der_x(n):
    vn     = np.ones(n)
    vd     = -2*vn
    A      = np.diag(vn[0:-1],1) + np.diag(vn[0:-1],-1) + np.diag(vd)

    return A

# BOUNDARY CONDITIONS

def vect_C(n,a,ht,h,uf,t):
    C     = np.zeros(n)
    C[0]  = 0*a*(ht)/(h*h)
    C[-1] = uf*a*(ht)/(h*h)
```

```
    return C

# FORCE APPLIED ON THE SYSTEM

def vect_g(n,t,ht,f,f0):
    g = ht*fun_ui(t,f,f0)*np.linspace(L,0,n)
    return g

def fun_ui(t,f,f0):
    return np.cos(2*np.pi*f*t)*f0

def solve(t0,tf,ht,h,A,u0,n,a,u_inf,f,f0):
    t = np.arange(t0+ht,tf+ht,ht)
    u = np.zeros((len(u0),len(t)))
    u[:,0] = u0
    for i in range(2,len(t)):
        u[:,i] = np.dot(A,u[:,i-1]) + vect_C(n,a,ht,h,u_inf,t[i-1]) + vect_g(n,t[i-1],
    ht,f,f0)

    return u

# DATA

mu    = 1.8             # dynamic viscosity of the incomplressible fluid
rho   = 1.5             # density of the incompressible fluid
p     = 2*1e2           # pressure gradient along x
n     = 35
L     = 2
h     = L/(n+1)
u_inf = 100
u0    = np.linspace(0,u_inf,n)
c     = mu/rho
f0    = p/rho
t0    = 0
tf    = 40
ht    = 0.001
f     = 0.5             # vortex detachment frequency (n vortexes/second)

# RESOLUTION

A = (np.diag(np.ones(n)) + (ht*c)*der_x(n)/(h**2))
u = solve(t0,tf,ht,h,A,u0,n,c,u_inf,f,f0)

# PLOT

X = np.linspace(0,L,len(u[:,0]))

plt.figure(figsize=(10,7))
my_plot, =plt.plot([],[])
plt.title("NAVIER-STOKES\n laminar flow on wing profile")
plt.xlim((-u_inf/5,u_inf + n))
plt.ylim((0,L))

t= 0
tmp = 0
U= np.zeros(len(u[:,0]))
i = 0
while t<tf:
    U = u[:,i]
    t = t + ht
    i = i + 1
    tmp = tmp +1

    if tmp==30 :
        tmp=0
        my_plot.set_data(U,X)
        plt.pause(0.01)
```

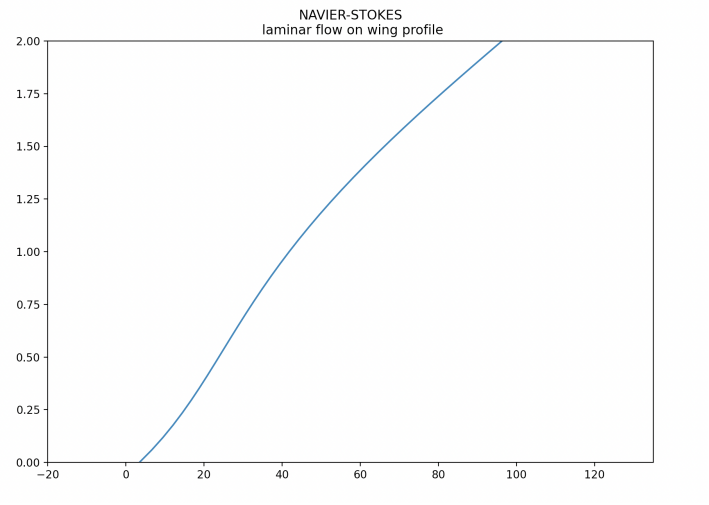
Listing 3: Python example

As a result this Python code provides the simulation in time of the v_x component

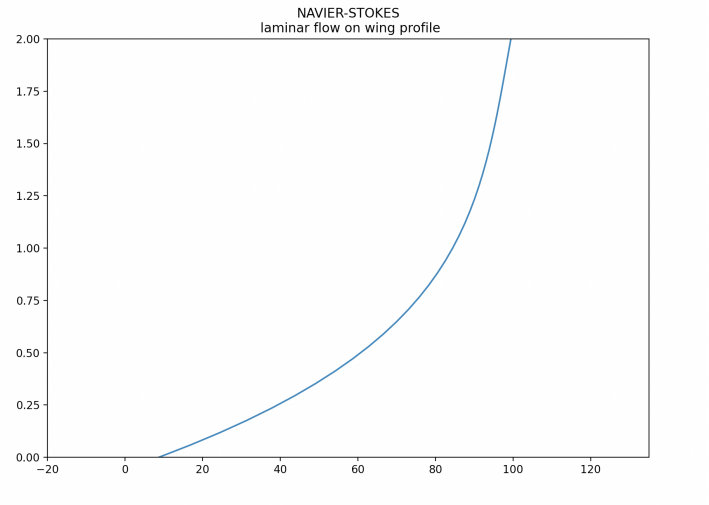
(the only component different from zero) of the velocity vector of the air flow around the wing profile; but for the sake of simplicity, below are reported the plot of the results at three different time steps only:



Step time t_1.



Step time t_2.



Step time t_3.

3.2.3 Heat Transfer Equation

By making reference to the Fourier's thermal conduction law, which states that:

$$\dot{q}_n = -k_n \frac{\partial T}{\partial n}$$

And the conservation of energy per unit time:

$$U_{in} - U_{out} = \dot{U}_{int}$$

considering an infinitesimal portion of material the heat transfer equation might be formulated by passing through the following steps:

$$U_{in} = \dot{q}_x dydz + \dot{q}_y dxdz + \dot{q}_z dxdy$$

$$U_{out} = \dot{q}_{x+dx} dydz + \dot{q}_{y+dy} dxdz + \dot{q}_{z+dz} dxdy =$$

$$(\dot{q}_x + \frac{\partial \dot{q}_x}{\partial x} dx) dydz + (\dot{q}_y + \frac{\partial \dot{q}_y}{\partial y} dy) dxdz + (\dot{q}_z + \frac{\partial \dot{q}_z}{\partial z} dz) dxdy$$

and:

$$\dot{U}_{int} = \rho c_p \frac{\partial T}{\partial t} dxdydz$$

thus:

$$\dot{q}_x dydz + \dot{q}_y dxdz + \dot{q}_z dxdy -$$

$$(\dot{q}_x + \frac{\partial \dot{q}_x}{\partial x} dx) dydz + (\dot{q}_y + \frac{\partial \dot{q}_y}{\partial y} dy) dxdz + (\dot{q}_z + \frac{\partial \dot{q}_z}{\partial z} dz) dxdy =$$

$$\rho c_p \frac{\partial T}{\partial t} dxdydz$$

hence:

$$-\frac{\partial \dot{q}_x}{\partial x} - \frac{\partial \dot{q}_y}{\partial y} - \frac{\partial \dot{q}_z}{\partial z} = \rho c_p \frac{\partial T}{\partial t}$$

which considering the Fourier's law comes out to be:

$$k_x \frac{\partial^2 T}{\partial x^2} + k_y \frac{\partial^2 T}{\partial y^2} + k_z \frac{\partial^2 T}{\partial z^2} = \rho c_p \frac{\partial T}{\partial t}$$

if $k_x = k_y = k_z = k$ then:

$$k \nabla^2 T = \rho c_p \frac{\partial T}{\partial t}$$

in the presence of an external source of thermal power:

$$\frac{1}{\alpha} \frac{\partial T}{\partial t} - \nabla^2 T = \frac{\dot{q}_{ext}}{k}$$

being α the thermal diffusivity of the material.

In light of the problem previously formulated, and by proceeding with the discretization of the partial differential equation, the problem might be numerically solved. Therefore, after the formulation of the problem by means of finite differences it assumes the following shape:

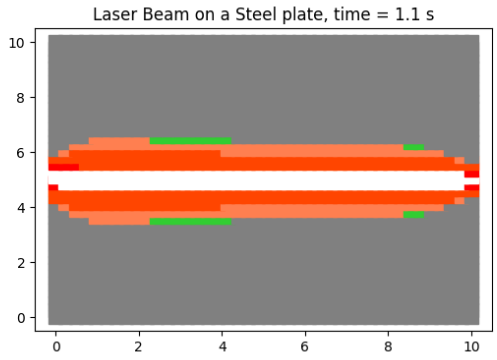
$$\frac{\underline{T}^{t+1} - \underline{T}^t}{h_t} = \alpha \left[\Delta\right] \underline{T}^t + \underline{q}_{eq}$$

Considering now a 2D case in which a steel plate is hit by a heat source, for example a laser for cutting applications, and looking for the way the heat diffuses through it, the solution of the mathematical model previously formulated might be a good representation of the system's behavior. Thus as always a Python script might be implemented to find an approximated solution:

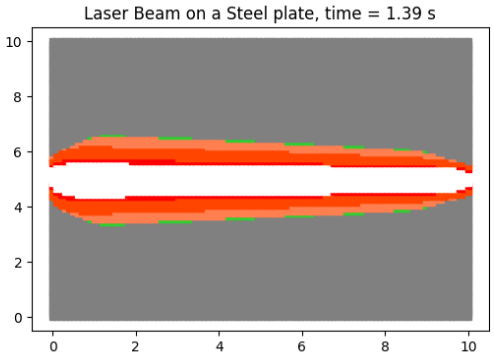
```
"""
@ author:   Andrea Dal Prete
@ copyright
"""

import numpy as np
import matplotlib.pyplot as plt
import os
import time

def grad(n):
    vn     = np.ones(n)
    vn[-1] = 0
    vn     = np.tile(vn,n)
    vn = vn[:-1]
    return (-4*np.diag(np.ones(n*n)) + np.diag(vn,1) + np.diag(vn,-1) +
            np.diag(np.ones(n*n-(n)), n)+ np.diag(np.ones(n*n-(n)), -n))

def solve(t_0,t_f,ht,n,T_0,a):
    start_time = time.time()
    t      = np.arange(t_0+ht,t_f,ht)
    T      = np.zeros((len(t), n*n))
    T[0,:] = T_0
    for i in range(1,len(t)):
        q = np.zeros(n*n)
        if t[i] < 0.67:
            q[(i-1)+int(n*n/2)] = 10**6 # best 10^7
        T[i,:] = np.matmul((np.eye(n*n) + (ht/a)*grad(n)/(h**2)),T[i-1,:]) + ht*q
    tim = time.time() - start_time
    print("Needed time for the calculation [s]:" + str(tim))
    return T,t,tim


# DATA

t_0 = 0
t_f = 1.1                       # time [s]
ht  = 0.015                     # delta t
L   = 10                        # plate length [mm]
n   = 42                        # number of elements
h   = L/n
a   = 18.8                      # thermal diffusivity [mm^2/s]
Ti  = 20 + 273.15               # initial temperature
T_0 = Ti*np.ones(n*n)
T_m = 1400 + 273.15             # melting temperature
T_e = 2700 + 273.15             # evaporation temperature

# SYSTEM SOLUTION

st_time = time.time()

u,t,tempo = solve(t_0,t_f,ht,n,T_0,a)

# PLOT

for j in range(0,len(u[:,0])):
    plt.figure(j, figsize=(9,8))
    plt.title('Laser Beam on a Steel plate, ' + 'time = ' + str(round(t[j],2)) + " s")
    u1 = u[j,:]
    x = np.linspace(0,L,n)
    y = np.linspace(0,L,n)

    for k in range(0,n):
        u_s = u1[k*n:(k+1)*n]
```

```
        for i in range(0,n):
            if u_s[i]<Ti+0.8:
                col = 'gray'
            elif u_s[i]>Ti+0.8 and u_s[i]<Ti+1.5:
                col = 'limegreen'
            elif u_s[i]>Ti+1.5 and u_s[i]<Ti+50:
                col = 'coral'
            elif u_s[i]>Ti+50 and u_s[i]<T_m:
                col = 'orangered'
            elif u_s[i]>T_m and u_s[i]<T_e:
                col = 'red'
            else:
                col = 'white'
            plt.plot(x[i],y[k], marker='s', markersize=9.8, color=col)
    plt.show(block=False)
    plt.pause(ht/(10**20))
    if j<len(u[:,0])-1:
        plt.close()
```

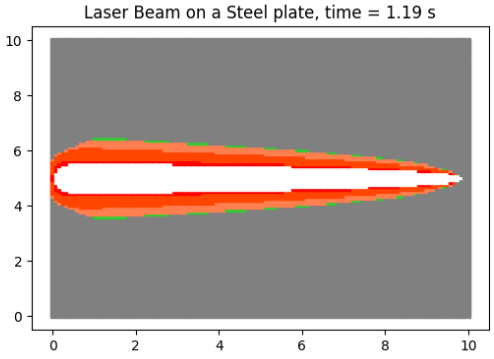
Listing 4: Python example



number of elements 42.



number of elements 100.



number of elements 120.

The figures above report the output provided by the script, which is a rough simulation of the discretized solution of the Heat Transfer Equation previously mentioned. The pictures represent a steel plate (gray) which is warmed up to the vaporization point in the white zone (where the cutting operation actually occurs), to the melting point in the red one, and to lower temperatures in the orange and green zones. As expected, by increasing the number of elements (n in the script) a more and more accurate solution might be reached, with the expectation to reach the convergence at a certain point, and obtaining at the end of the simulations the results represented in the figures.

4 Conclusions

The *Galerkin Finite Elements Method* and the *Finite Differences Method* are just two of the manifold methods nowadays used to find approximated solutions of Partial Differential Equations, (i.e. the reader is referred to the *Finite Volumes Method*, *Gradient Discretization Method*, *Spectral Method*, *Method of lines*, etc), but they represent the starting point from which all the others have been developed. The extreme potential of this numerical tools allow scientists of several fields to make precise prediction and increase the capability of understanding what actually happens on a system along its working conditions, and as knowledge goes deeper in understanding what actually happens, the human being is more and more capable to find solutions so to decrease the waste and increase the efficiency of the working conditions of an engineering system. Moreover, if properly coupled with an appropriate graphic interface, these methods are widely used to perform numerical simulations and to have a concrete visualization of the studied case of the system, and this allows to make virtual experiments and possibly design a whole engineering system without making empirical experiments, therefore drastically reducing the waste of time and resources and inexorably increasing the safety of the design conditions.
For a practical example of an advanced application of numerical simulation, the reader is referred to the following link:

$$https://youtu.be/4EQCMR2LROM$$