AMCS 394E: FEM

Release 0.1.1

A.J.J. Lagerweij

Feb 18, 2021

HOMEWORK ASSIGNMENTS

CHAPTER

ONE

AMCS 394E: FEM

In this repository you can find my homework for Contemporary Topics in Computational Science: Computing with the Finite Element Method. The course is hosted from AMCS 394E: Computing with the Finite Element Method Git.

The folder */src/* contains the actual functions wherase the homework functions contain the homework assignments and the scripts that are used to run tha problems.

$$physics * argmin$$

1.1 Homework 1

Topic

Homework regarding the first week. The goal is to work with basic numerical approximation of PDE's' and functions.

Bram Lagerweij 18 Feb 2020

1.1.1 1 Method of Lines

Consider the one-dimensional advection diffusion equation:

$$u_t + cu_x - \mu u_{xx} = 0 \qquad \forall\, x \in \Omega = [0,1] \quad \& \quad t > 0$$

where $\mu > 0$ is the diffusion coefficient and c the wave speed. Consider periodic boundary conditions and the following initial condition:

$$u(x,0) = \sin(2\pi x)$$

What do we expect the exact solution to do? Due to the advective part, the initial condition travels at constant speed to the right. At the same time, due to the diffusive term, the initial condition is dissipated at a rate that depends on μ.

Consider the following discretization. Use second-order central finite differences to approximate u_x and u_{xx}. Use forward and backward Euler to obtain full discretization (write down the schemes). Consider a fixed mesh with of Δx.

1.1 Advective Diffusive PDE

Consider a final time of $t = 1$, $c = 1$ and $\mu = 0.01$. For each full discretization proceed as follows:

1. Experiment using the following time step sizes: $\Delta t = 10^4$, 10^3 and 10^2.
2. How do the explicit and implicit methods behave for these time steps?

Fig. 1.1: : The forward difference scheme is unstable for $dt = 10^{-3}$, the backward scheme behaves as expected. Click here for an animated version.

Fig. 1.2: : With a timestep of $dt = 10^{-3}$ both the forward and backward Euler scheme are stable. Click here for an animated version.

Fig. 1.3: : As expected with a timestep of $dt = 10^{-4}$ both time integrations behave stable. Click here for an animated version.

```
r"""
Solving an Advective and Diffusive PDE with finite differences.

The PDE described by

.. math::
    u_{t} + u_{x} = \mu u_{xx}  \quad \forall x \in\Omega = [0, 1]  \;\; \& \;\;  t >␣
↪0

With a periodic boundary condition. It will show a combination of diffusive
and advective behaviour. The approximation used is a second order finite
difference scheme in space with both a forward and backward Euler method of
lines implementation to handle the time direction.

The goal is to implement the code in python and not rely on existing solvers.

Bram Lagerweij
COHMAS Mechanical Engineering KAUST
2021
"""

# Importing External modules
import sys
import matplotlib.pyplot as plt
import numpy as np

# Importing my own scripts
sys.path.insert(1, '../src')
from pde import advectivediffusive
from time_integral import forwardEuler, backwardEuler


if __name__ == '__main__':
    # Define properties
    dx = 1e-2
    dt = 1e-4
    t_end = 1
```

(continues on next page)

(continued from previous page)

```
    mu = 0.01  # Diffusive term
    c = 1  # Advective term

    # Define discrete ranges
    dof = int(1 / dx) + 1
    x, dx = np.linspace(0, 1, dof, retstep=True)
    t = np.arange(0, t_end + dt, step=dt)

    # Prepare solver
    u0 = np.sin(2 * np.pi * x)  # Initial condition

    # Solve the problem using method of lines.
    u_forw = forwardEuler(advectivediffusive, u0, dt, t_end, args=(dof, dx, mu, c))
    u_back = backwardEuler(advectivediffusive, u0, dt, t_end, args=(dof, dx, mu, c))

    # Plotting plotting statically
    plt.xlim(0, 1)
    plt.ylim(-1, 1)
    plt.xlabel('$x$ location')
    plt.ylabel('$u(x)$')
    plt.annotate('time t={}'.format(t[-1]), xy=(0.5, 0.9), ha='center')
    plt.tight_layout()

    plt.plot(x, u_forw, label='forward')
    plt.plot(x, u_back, label='backward')

    plt.legend()
    plt.show()
```

1.2 Advective PDE

Consider $\mu = 0$ and $c = 2$ and solve the PDE using the explicit and the implicit methods. Use $\Delta t = 10^4$ and solve the problem for the following final times $t = 1,\ 5,\ 10,\ 15$ and 20. Comment on the behaviour of each full discretization as the final time increases.

Fig. 1.4: : Even with small time steps this type of hyperbolic like equation can become unstable when using a forward Euler method. Click here for an animated version.

There is a so called Courant-Friedrichs-Lewy condition that formulates a condition of stability on the model:

$$C = \frac{c\Delta t}{\Delta x} \leq C_{\max}$$

Where $C_{\max}$ is a constant, which for explicit schemes, such as forward Euler, is around 1. If the condition is violated the method becomes unstable, that does not mean that the results are unstable from the first iteration. In the animation the instabilities become only clear after 14 seconds. Nevertheless, even at $t = 1$ the method should be considered unstable. Similarly the backward Euler is inaccurate as well, it is too dissipative, after 20 seconds around 20% of our, wave magnitude has disappeared.

```
r"""
Solving an Advective PDE with finite differences.

The PDE described by

```

(continues on next page)

(continued from previous page)

```
.. math::
    u_{t} + u_{x} = 0  \quad \forall x \in\Omega = [0, 1]  \;\; \& \;\;  t > 0

With a periodic boundary condition. The approximation used is a second order
finite difference scheme in space with both a forward and backward Euler method
of lines implementation to handle the time direction.

The goal is to implement the code in python and not rely on existing solvers.

Bram Lagerweij
COHMAS Mechanical Engineering KAUST
2021
"""

# Importing External modules
import sys
import matplotlib.pyplot as plt
import numpy as np

# Importing my own scripts
sys.path.insert(1, '../src')
from pde import advective
from time_integral import forwardEuler, backwardEuler


if __name__ == '__main__':
    # Define properties
    dx = 1e-2
    dt = 1e-4
    t_end = 20
    c = 2  # Advective term

    # Define discrete ranges
    dof = int(1 / dx) + 1
    x, dx = np.linspace(0, 1, dof, retstep=True)
    t = np.arange(0, t_end + dt, step=dt)

    # Prepare solver
    u0 = np.sin(2 * np.pi * x)  # Initial condition

    # Solve the problem using method of lines.
    u_forw = forwardEuler(advective, u0, dt, t_end, args=(dof, dx, c))
    u_back = backwardEuler(advective, u0, dt, t_end, args=(dof, dx, c))

    # Plotting plotting statically
    plt.xlim(0, 1)
    plt.ylim(-1, 1)
    plt.annotate('time t={}'.format(t[-1]), xy=(0.5, 0.9), ha='center')
    plt.tight_layout()

    plt.plot(x, u_forw, label='forward')
    plt.plot(x, u_back, label='backward')

    plt.legend()
    plt.show()
```

1.1.2 2 Approximation of functions

Consider the function:

$$f(x) = \sin^4(2\pi x) \qquad \forall\, x \in \Omega = [0, 1]$$

for which we have to find multiple global and local approximations. Let $f_h(x)$ be such an approximation for a given grid. We consider the following errors:

$$E_1 := \int_\Omega |f(x) - f_h(x)| dx \quad \text{and} \quad E_2 := \int_\Omega \big(f(x) - f_h(x)\big)^2 dx$$

2.1 Global Approximations

Consider the following approximations all with N terms:

1. the Taylor series around $x = 0.5$,
2. the Fourier series,
3. a global polynomial interpolation on the closed interval given by:

$$f_h(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{N-1} x^{N-1}$$

Consider different levels of refinement, $N = 4,\ 5,\ 6,\ \ldots,\ 10$ and for each approximation report both E_1 and E_2.

2.1.1 Taylor series

The Taylor series till the order N is defined through:

$$f_h(x) = \sum_{n=0}^{N} \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n$$

Which immediately got me into problems, analyzing the n-th derivative of a function is a numerically a pain. Quickly the round off errors become significant, and from the 5th derivative onward the basic *scipy* Taylor series function became useless. As a result I decided to hardcode the weighting constants in our expansion, these are obtained from manual derivatives.

Fig. 1.5: : Approximating $f(x)$ with a Taylor series centered around $x_0 = 0.5$ till order 10.

From Fig. 1.5 it can be observed that the Taylor series is not a very efficient approximation. At the boundary of our domain the error is very high.

2.1.2 Fourier series

The Fourier series, which we assume to be real, approximates the equation with:

$$f_h(x) = \sum_{n=0}^{N} c_n \exp^{\frac{2\pi n x}{P} i} + \bar{c}_n \exp^{-\frac{2\pi n x}{P} i}$$

where P is the period of the function $f(x)$ and c_n are complex valued coefficients that can be found through a Fourier Transform. In our case I used a FFT algorithm to find these coefficients from our discrete dataset, essentially the real-FFH tries to solve:

$$c_n = \sum_{n=0}^{K} x_k \exp^{\frac{2\pi kn}{K-1}} \qquad n = 0, \ldots, N$$

in a highly efficient manner. Notice that for each unknown c_n consists of a real and imaginary part. This does mean that this approximation for any given N is more complex. The resulting approximation is shown in Fig. 1.6. Which show that this series is highly efficient in the approximation of our function. This is not to surprising, after all we are approximation a trigonometric functions with a series of trigonometric functions it is likely that we find the exact function somewhere in our series.

Fig. 1.6: : Approximating $f(x)$ with a Fourier series seems to be exact from the fourth order.

2.1.3 Polynomial series

The polynomial series

$$f_h(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{N-1} x^{N-1}$$

was to be found with a fitting through N evenly spaced points x_i throughout this interval. It should be noted that this type of fitting can be rewritten as an minimization:

$$_{a_0,\ldots,a_{N-1}} \sum_{i=0}^{N} \big(f(x_i) - f_h(x_i)\big)^2$$

$$thatmeans : finda_0, \ldots a_{N-1} suchthat f(x_i) - f_h(x_i) = 0 \quad \forall x_i$$

This minimization can efficiently be casted to a system of equations and subsequently be solved. This system of equations has N unknowns and N functions, and because each of these functions is linearly independent a solution exists. Simply said we construct a polynomial that goes exactly through these N points.

Fig. 1.7: : Approximating $f(x)$ with a polynomials of order $N-1$ using N sample points.

One can also choose to use more sample points to evaluate the minimization problem, lets consider that we use M sample points. It is not generally possible to find a $N-1$ order polynomial to fit exactly through more then N points. But we can find the best polynomial, to be specific one that minimizes:

$$_{a_0,\ldots,a_{N-1}} \sum_{i=0}^{M} \big(f(x_i) - f_h(x_i)\big)^2$$

Which is as if we are minimizing our error E_2 at only discrete points, instead of solving the integral itself. Anyway, Fig. 1.8 shows this fit would look like. The results seems closer, because we're not just minimizing the error at N points but at $5N$ points.

Fig. 1.8: : Approximating $f(x)$ with a polynomials of order $N-1$ using $M = 5N$ sample points.

2.1.4 Comparison

For the comparison of these different approximations I've plotted the errors on a log scale. Please do note that the Fourier series has 2 times as many unknowns for the N compared to the other methods.

Fig. 1.9: : The error E_1 for our different approximations where the approximation order ranges from 1 to 20.

Fig. 1.10: : The error E_2 for our different approximations where the approximation order ranges from 1 to 20.

I assume that the error of the Taylor series is increasing because the higher order terms will cause higher errors at the boundaries of our domain. But all in all it is my opinion that the Taylor series is a bad approximation for this purpose, it is difficult to calculate due to the derivatives and the result is inaccurate. This is not so surprising however, Taylor series are meant to approximate the behaviour of a function around a given point x_0 to characterize the local behaviour. We are here using it on a relatively large domain.

The script used for these computations can be found at 3 LocalApproximation.py.

2.2 Local Approximations

Split the domain Ω into N cells. For each cell K, compute linear and quadratic approximations $f_K(x)$ where $f_K(x_i) = f(x_i)$ where x_i are evenly spaced gridpoints, including the boundaries of the cell. Compute and report both E_1 and E_2 for a different numbers of cells $N = 4, 5, 6, \ldots, 10$.

The approximation by linear elements is created by scaling hat (shape) functions appropriately. These functions are chosen in such a way that:

1) The sum of all the shape functions together equals one, $\sum_{n=1}^{N} \varphi_i(x) = 1$ This is called the Partition of Unity Method.

2) There where a single function reaches its maximum all the other functions equal zero.

Then our approximation is defined by:

$$f_h(x) = \sum_{n=1}^{N} w_n \varphi_n(x)$$

where the weights w_n are unknown. But because the shape function where chosen smartly these weights are independent. After all at the point where a single shape function reaches its maximum (1) the other functions are zero. As a result the weight of this shape function equals the value of the function we are trying to approximate at the center point of the shape:

$$w_n = f(X_n)$$

where X_n denotes the point where shape function $\varphi_n(x)$ reaches its maximum.

2.2.1 Linear Elements

In the case of linear elements these shape functions are defined as:

$$\varphi_n(x) = \begin{cases} 0 & \forall \quad 0 \\ \leq x \leq & X_{n-1} \\ \frac{x-X_{n-1}}{X_n - X_{n-1}} & \forall \quad X_{n-1} \\ \leq x \leq & X_n \\ 1 - \frac{x-X_n}{X_{n+1}-X_n} & \forall \quad X_n \\ \leq x \leq & X_{n+1} \\ 0 & \forall \quad X_{n+1} \\ \leq x \leq & L \end{cases}$$

where X_n is the node of this shape function, X_{n-1} and X_{n+1} the nodes surrounding ours.

A more efficient formulation includes the creation of a unit function that is rescaled depending on the locations of the nodes. But I haven't yet implemented such an function yet.

Fig. 1.11: : The function $4\sin(\pi x) + 1$ approximated with four elements. The first element contain the orange and half of the green shape function.

Fig. 1.12: : The function $4\sin(\pi x) + 1$ approximated more and more linear elements.

Fig. 1.13: : The approximation of $f(x)$ with linear elements.

2.2.2 Quadratic Elements

In the case of quadratic elements there are two different types of shape function. One of these function extents into two elements, similar to what the linear element does. The second shape function is only inside a single element, and on an interior node. This node is placed exactly in the middle between the start and end of the element. I'll give these nodes the subscripts $n - \frac{1}{2}$ and $n + \frac{1}{2}$. Now the shape functions are defined by:

$$\varphi_n(x) = \begin{cases} 0 & \forall \quad 0 \\ \leq x \leq & X_{n-1} \\ \frac{2}{(X_n - X_{n-1})^2}(x - X_{n-1})(x - X_{n-\frac{1}{2}}) & \forall \quad X_{n-1} \\ \leq x \leq & X_n \\ \frac{2}{(X_{n+1} - X_n)^2}(x - X_{n+1})(x - X_{n+\frac{1}{2}}) & \forall \quad X_n \\ \leq x \leq & X_{n+1} \\ 0 & \forall \quad X_{n+1} \\ \leq x \leq & L \end{cases}$$

$$\varphi_{n-\frac{1}{2}}(x) = \begin{cases} 0 & \forall \quad 0 \\ \leq x \leq & X_{n-1} \\ -\frac{4}{(X_n - X_{n-1})^2}(x - X_{n-1})(x - X_n) & \forall \quad X_{n-1} \\ \leq x \leq & X_n \\ 0 & \forall \quad X_{n+1} \\ \leq x \leq & L \end{cases}$$

Again a more efficient formulation includes the creation of a unit function that is rescaled depending on the locations of the nodes. But I haven't yet implemented such an function yet.

Fig. 1.14: : The function $4\sin(\pi x) + 1$ approximated with four elements. The first element contain the orange and half of the green shape function.

Fig. 1.15: : The function $4\sin(\pi x) + 1$ approximated more and more quadratic elements.

Fig. 1.16: : The approximation of $f(x)$ with quadratic elements.

It is important to notice from Fig. 1.16 that the resulting curve is not smooth. for example at $x = 0.5$ one can see that the red approximation (6 elements) is non-smooth.

2.2.3 Comparison

For the comparison of these different approximations I've plotted the errors on a log scale. Please do note that the quadratic elements have $(N + 1)N$ unknowns where the linear elements have $N + 1$ weights to be determined. Nevertheless there is no interdependency between these weights, which as mentioned before means that these can be determined independently.

The script used for these computations can be found at 4 GlobalApproximation.py.

$$physics$$

1.2 About

Toppic

The reason for me is to solve classical problems in Solid Mechanics. This section, and those below will introduce the typical equations that are encountered in solid mechanics. This section is not exhaustive and it might be extended in the future to discuss more details.

Bram Lagerweij 11 Feb 2020

The examples will become gradually more complex. It starts with the simplest problem, the Laplace equation:

$$^2u(\vec{m}) = 0 \qquad \forall \vec{m} \in \Omega$$

In here one can imagine various levels of complication:

1. With a simple geometry, no sharp corners, and a combination of Neuman and Diriclet boundary conditions.
2. With a more complex geometry, sharp corners, cracks and inclusions.
3. With a 'non-linear' stiffness, $(\vec{C}\, u(\vec{m}))$ adding a non-constant variable $\vec{C}$ which is a function depending somehowe on u.
4. Where $\vec{C}$ is non-linear and history dependent, aka $\vec{C}^{(n+1)}$ is a function af all previous timesteps.
5. With softening in the non-linear stiffness C, that is the tangent of $\vec{C}\, u$ will become negative at some point.

Fig. 1.17: : The error E_1 for our element based approximations with 1 to 20 elements.

Fig. 1.18: : The error E_1 for our element based approximations with 1 to 20 elements.

6. Versions in 3D

Moving on to solids where we solve elasticity and plasticity equations:

$$\begin{aligned} \sigma + \vec{b} = 0 \qquad & \forall \vec{m} \in \Omega \\ where \sigma = C : \varepsilon & \\ \varepsilon = \frac{1}{2}\left(\vec{u} + (\vec{u})^T\right) & \end{aligned}$$

The simplest problem would be linear elasticity, but more complicated versions can be build as well.

1. With a simple geometry, no sharp corners, and a combination of Neuman and Diriclet boundary conditions.
2. With a more complex geometry, sharp corners, cracks and inclusions.
3. Large displacements (geometrically non-linear) and deformations (this might require a different strain measure).
4. Softening and possbily fracture.
5. Self Contact.

$$physics$$

1.3 Poisson Equation

Toppic

The Poisson equation is the simplest example of the PDE's considerd in Solid Mechanics. It is an eliptical PDE, and is simplified compared to linear elasticity in the sense that its solution is a scalar field, instead fo the vector field found in elasticity problems. This makes Poisson's equation a good start to explore numerical solving strategies for Solid Mechanics problems.

Bram Lagerweij 11 Feb 2020

1.3.1 Laplace Equation

The most basic description of the Laplace equation is given by:

$$\begin{aligned} {}^2u(\vec{m}) = [2]ux + [2]uy = 0 & \\ \forall \vec{m} \in \Omega & \\ \text{s.t.:} \quad u(\vec{m}) = \vec{\tilde{u}}(\vec{m}) & \\ \forall \vec{m} \in \mathcal{S}_u & \\ u(\vec{m}) = \vec{\tilde{t}}(\vec{m}) & \\ \forall \vec{m} \in \mathcal{S}_t & \end{aligned}$$

Where the entirety of the boundary $\partial\Omega$ is the union of these to boundary conditions that do not intersect.

$$\begin{aligned}\partial\Omega &= \mathcal{S}_u \cup \mathcal{S}_t \\ 0 &= \mathcal{S}_u \cap \mathcal{S}_t\end{aligned}$$

The following images summarizes this.

Fig. 1.19: A domain Ω subjected to the Laplace equation with combined boundary conditions.

1.3.2 Poisson equation

In case of nonhomogeneous formulations the Laplace equations is called the Poisson equation.

$$\begin{aligned}{}^2u(\vec{m}) = [2]ux + [2]uy &= \vec{b}(\vec{m}) \\ \forall \vec{m} \in \Omega & \\ \text{s.t.:} \quad u(\vec{m}) &= \vec{\tilde{u}}(\vec{m}) \\ \forall \vec{m} \in \mathcal{S}_u & \\ u(\vec{m}) &= \vec{\tilde{t}}(\vec{m}) \\ \forall \vec{m} \in \mathcal{S}_t &\end{aligned}$$

The boundary condition is still defined in the same way as in the Laplace equaiton.

1.3.3 Weak form

1.4 Derivatives

Storing various derivatives for the purpose of importing them into the partial derivative equations in another scrcipt.

Bram Lagerweij COHMAS Mechanical Engineering KAUST 2021

`derivative.`**`Dx`**(*dof*, *dx*, *bc='periodic'*)

Return the central differences matrix for the first derivatie. That is the matrix D_x represents the central difference approximation of ∂_x in 1D axis systems.

Parameters

- **dof** (*int*) – Number of spacial degrees of freedom.
- **dx** (*float*) – Spacial step size.
- **bc** (*string, optional*) – The type of boundary condition to be used. The default is 'periodic'.

Raises **ValueError** – Is raised when the requested boundary condition is not implemented.

Returns The central difference approximation of the first derivative.

Return type matrix (sparse csr format)

`derivative.`**`Dxx`**(*dof*, *dx*, *bc='periodic'*)

Return the central differences matrix for the second derivatie. That is the matrix D_{xx} represents the central difference approximation of ∂_{xx} in 1D axis systems.

Parameters

- **dof** (*int*) – Number of spacial degrees of freedom.
- **dx** (*float*) – Spacial step size.
- **bc** (*string, optional*) – The type of boundary condition to be used. The default is 'periodic'.

Raises **ValueError** – Is raised when the requested boundary condition is not implemented.

Returns The central difference approximation of the first derivative.

Return type matrix (sparse csr format)

1.5 Partial Differential Equations

Storing varios PDE's that can be will be solved in this course. This includes:

- Diffusive 1D

$$u_t = \mu u_{xx} \qquad \forall\, x \in \Omega = [0,1] \quad \& \quad t > 0$$

- Advective 1D

$$u_t + c u_x = 0 \qquad \forall\, x \in \Omega = [0,1] \quad \& \quad t > 0$$

- Diffusive-Advective 1D

$$u_t + c u_x = \mu u_{xx} \qquad \forall\, x \in \Omega = [0,1] \quad \& \quad t > 0$$

The goal is to implement the code in python and not rely on existing methods.

Bram Lagerweij COHMAS Mechanical Engineering KAUST 2021

pde.**advective**(*dof*, *dx*, *c*)

Time derivative of the PDE for advective diffusive problems.

$$u_t + c u_x = 0 \qquad \forall\, x \in \Omega = [0,1] \quad \& \quad t > 0$$

Thus this returns:

$$u_t = -c u_x$$

Because we use finite difference based matrix products we can convert this into a matrix vector product, where D_x is the central difference approximation of ∂_x:

$$u_t = -c D_x u = K u$$

This function calculates the matrix K. Because it should be compatible with general, non-homogeneous formulation, a part that is independent of u is also included.

Parameters

- **dof** (*int*) – Number of degrees of freedom.
- **dx** (*float*) – Stepsize in the of spatial discretisation.
- **c** (*float*) – The avective coefficient.

Returns

- **K** (*matrix (sparse csr format)*) – The time derivative part of the pde obtained from the spatial part.

- **b** (*vector (dense array)*) – The remaining term, in this homeneous case it is a zero array.

pde.**advectivediffusive**(*dof*, *dx*, *mu*, *c*)

Time derivative of the PDE for advective diffusive problems.

$$u_t + cu_x = \mu u_{xx} \qquad \forall\, x \in \Omega = [0,1] \quad \& \quad t > 0$$

Thus this returns:

$$u_t = -cu_x + \mu u_{xx}$$

Because we use finite difference based matrix products we can convert this into a matrix vector product, where D_x is the central difference approximation of ∂_x and similarly D_{xx} the central difference apprximation of ∂_{xx}:

$$u_t = -cD_x u + \mu D_{xx} u = \left(-cD_x + \mu D_{xx}\right) u = Ku$$

This function calculates the matrix K. Because it should be compatible with general, non-homogeneous formulation, a part that is independent of u is also included.

Parameters

- **dof** (*int*) – Number of degrees of freedom.
- **dx** (*float*) – Stepsize in the of spatial discretisation.
- **mu** (*float*) – The diffusive coefficient.
- **c** (*float*) – The avective coefficient.

Returns

- **K** (*matrix (sparse csr format)*) – The time derivative part of the pde obtained from the spatial part.
- **b** (*vector (dense array)*) – The remaining term, in this homeneous case it is a zero array.

pde.**diffusive**(*dof*, *dx*, *mu*)

Time derivative of the PDE for advective diffusive problems.

$$u_t = \mu u_{xx} \qquad \forall\, x \in \Omega = [0,1] \quad \& \quad t > 0$$

Thus this returns:

$$u_t = \mu u_{xx}$$

Because we use finite difference based matrix products we can convert this into a matrix vector product, where $D_x x$ is the central difference approximation of ∂_{xx}:

$$u_t = \mu D_{xx} u = Ku$$

This function calculates the matrix K. Because it should be compatible with general, non-homogeneous formulation, a part that is independent of u is also included.

Parameters

- **dof** (*int*) – Number of degrees of freedom.
- **dx** (*float*) – Stepsize in the of spatial discretisation.
- **mu** (*float*) – The defusive coefficient.

Returns

- **K** (*matrix (sparse csr format)*) – The time derivative part of the pde obtained from the spatial part.
- **b** (*vector (dense array)*) – The remaining term, in this homeneous case it is a zero array.

1.6 Time Integration

Various implementations of the method of lines to progress through time. The goal is to implement the code in python and not rely on existing solvers.

Bram Lagerweij COHMAS Mechanical Engineering KAUST 2021

time_integral.**backwardEuler**(*func*, *u*, *dt*, *t_end*, *args=()*)

Itterate a through time with the backward Eurler method.

The backward Euler method predicts the field of our function based upon information of the previous timestep only. Imagine that we are at timestep n and want to predict our field at timestep $u^{(n+1)}$. Now a backward finite difference approximation used the time derivative of the next timestep, wich is not yet known:

$$u_t^{(n+1)} = \frac{-u^{(n)} + u^{(n+1)}}{dt}$$

That is we can predict our field in the future timestep as:

$$u^{(n+1)} = u^{(n)} + dt\, u_t^{(n+1)}$$

It is important to notic that there is a term with an unknown, as that is at time step :math:`n+1' on both sides of the equation. Our time derivative is obtained with an approximation equation:

$$u_t = Ku + b$$

where matrix K and vector b stem from approximations of our spatial derivatives defined by the functien provided to *func*. This results in:

$$u^{(n+1)} = u^{(n)} + dt\,(Ku^{(n+1)} + b)$$

Now we rewrite it into a system of equations where we find all unknowns on the left hand side and all knownn on the right hand side.

$$(I - dt\,K)\,u^{(n+1)} = u^{(n)} + dt\,b$$

Parameters

- **func** (*callable*) – The time derivative of the pde to be solved such that $u_t = K\,u + b$.
- **u** (*array_like*) – The field at the start $u(t = 0)$.
- **dt** (*float*) – The size of the time step.
- **t_end** (*float*) – Time at termination.
- **args** (*tuple, optional*) – The parameters into the PDE approximation. Defealts to an empty tuple.

Returns The function for all time steps.

Return type array_like

time_integral.**forwardEuler**(*func*, *u*, *dt*, *t_end*, *args=()*)

Itterate a through time with the forward Eurler method.

The backward Euler method predicts the field of our function based upon information of the previous timestep only. Imagine that we are at timestep n and want to predict our field at timestep $u^{(n+1)}$. Now a forward finite difference approximation is used:

$$u_t^{(n)} = \frac{-u^{(n)} + u^{(n+1)}}{dt}$$

That is we can predict our field in the future timestep as:

$$u^{(n+1)} = u^{(n)} + dt\, u_t^{(n)}$$

Our time derivative at the current timestep, $u_t^{(n)}$ is obtained with:

$$u_t = Ku + b$$

where matrix K and vector b stem from approximations of our spatial derivatives defined by the functien provided to *func*. Resulting in the following update scheme:

$$u^{(n+1)} = u^{(n)} + dt\,(Ku^{(n)} + b)$$

most important of all is to see that everything on the right hand side is exactly known. Thus the updated field can be calculated directly.

Parameters

- **func** (*callable*) – The time derivative of the pde to be solved such that $u_t = K\,u + b$.
- **u** (*array_like*) – The field at the start $u(t = 0)$.
- **dt** (*float*) – The size of the step.
- **t_end** (*float*) – Time at termination.
- **args** (*tuple, optional*) – The parameters into the PDE approximation. Defealts to an empty tuple.

Returns The function for all time steps.

Return type array_like

1.7 Mozilla Public License Version 2.0

1.7.1 1. Definitions

1.1. "Contributor" means each individual or legal entity that creates, contributes to the creation of, or owns Covered Software.

1.2. "Contributor Version" means the combination of the Contributions of others (if any) used by a Contributor and that particular Contributor's Contribution.

1.3. "Contribution" means Covered Software of a particular Contributor.

1.4. "Covered Software" means Source Code Form to which the initial Contributor has attached the notice in Exhibit A, the Executable Form of such Source Code Form, and Modifications of such Source Code Form, in each case including portions thereof.

1.5. "Incompatible With Secondary Licenses" means

(a) that the initial Contributor has attached the notice described in Exhibit B to the Covered Software; or

(b) that the Covered Software was made available under the terms of version 1.1 or earlier of the License, but not also under the terms of a Secondary License.

1.6. "Executable Form" means any form of the work other than Source Code Form.

1.7. "Larger Work" means a work that combines Covered Software with other material, in a separate file or files, that is not Covered Software.

1.8. "License" means this document.

1.9. "Licensable" means having the right to grant, to the maximum extent possible, whether at the time of the initial grant or subsequently, any and all of the rights conveyed by this License.

1.10. "Modifications" means any of the following:

(a) any file in Source Code Form that results from an addition to, deletion from, or modification of the contents of Covered Software; or

(b) any new file in Source Code Form that contains any Covered Software.

1.11. "Patent Claims" of a Contributor means any patent claim(s), including without limitation, method, process, and apparatus claims, in any patent Licensable by such Contributor that would be infringed, but for the grant of the License, by the making, using, selling, offering for sale, having made, import, or transfer of either its Contributions or its Contributor Version.

1.12. "Secondary License" means either the GNU General Public License, Version 2.0, the GNU Lesser General Public License, Version 2.1, the GNU Affero General Public License, Version 3.0, or any later versions of those licenses.

1.13. "Source Code Form" means the form of the work preferred for making modifications.

1.14. "You" (or "Your") means an individual or a legal entity exercising rights under this License. For legal entities, "You" includes any entity that controls, is controlled by, or is under common control with You. For purposes of this definition, "control" means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50%) of the outstanding shares or beneficial ownership of such entity.

1.7.2 2. License Grants and Conditions

2.1. Grants

Each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by such Contributor to use, reproduce, make available, modify, display, perform, distribute, and otherwise exploit its Contributions, either on an unmodified basis, with Modifications, or as part of a Larger Work; and

(b) under Patent Claims of such Contributor to make, use, sell, offer for sale, have made, import, and otherwise transfer either its Contributions or its Contributor Version.

2.2. Effective Date

The licenses granted in Section 2.1 with respect to any Contribution become effective for each Contribution on the date the Contributor first distributes such Contribution.

2.3. Limitations on Grant Scope

The licenses granted in this Section 2 are the only rights granted under this License. No additional rights or licenses will be implied from the distribution or licensing of Covered Software under this License. Notwithstanding Section 2.1(b) above, no patent license is granted by a Contributor:

(a) for any code that a Contributor has removed from Covered Software; or

(b) for infringements caused by: (i) Your and any other third party's modifications of Covered Software, or (ii) the combination of its Contributions with other software (except as part of its Contributor Version); or

(c) under Patent Claims infringed by Covered Software in the absence of its Contributions.

This License does not grant any rights in the trademarks, service marks, or logos of any Contributor (except as may be necessary to comply with the notice requirements in Section 3.4).

2.4. Subsequent Licenses

No Contributor makes additional grants as a result of Your choice to distribute the Covered Software under a subsequent version of this License (see Section 10.2) or under the terms of a Secondary License (if permitted under the terms of Section 3.3).

2.5. Representation

Each Contributor represents that the Contributor believes its Contributions are its original creation(s) or it has sufficient rights to grant the rights to its Contributions conveyed by this License.

2.6. Fair Use

This License is not intended to limit any rights You have under applicable copyright doctrines of fair use, fair dealing, or other equivalents.

2.7. Conditions

Sections 3.1, 3.2, 3.3, and 3.4 are conditions of the licenses granted in Section 2.1.

1.7.3 3. Responsibilities

3.1. Distribution of Source Form

All distribution of Covered Software in Source Code Form, including any Modifications that You create or to which You contribute, must be under the terms of this License. You must inform recipients that the Source Code Form of the Covered Software is governed by the terms of this License, and how they can obtain a copy of this License. You may not attempt to alter or restrict the recipients' rights in the Source Code Form.

3.2. Distribution of Executable Form

If You distribute Covered Software in Executable Form then:

(a) such Covered Software must also be made available in Source Code Form, as described in Section 3.1, and You must inform recipients of the Executable Form how they can obtain a copy of such Source Code Form by reasonable means in a timely manner, at a charge no more than the cost of distribution to the recipient; and

(b) You may distribute such Executable Form under the terms of this License, or sublicense it under different terms, provided that the license for the Executable Form does not attempt to limit or alter the recipients' rights in the Source Code Form under this License.

3.3. Distribution of a Larger Work

You may create and distribute a Larger Work under terms of Your choice, provided that You also comply with the requirements of this License for the Covered Software. If the Larger Work is a combination of Covered Software with a work governed by one or more Secondary Licenses, and the Covered Software is not Incompatible With Secondary Licenses, this License permits You to additionally distribute such Covered Software under the terms of such Secondary License(s), so that the recipient of the Larger Work may, at their option, further distribute the Covered Software under the terms of either this License or such Secondary License(s).

3.4. Notices

You may not remove or alter the substance of any license notices (including copyright notices, patent notices, disclaimers of warranty, or limitations of liability) contained within the Source Code Form of the Covered Software, except that You may alter any license notices to the extent required to remedy known factual inaccuracies.

3.5. Application of Additional Terms

You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Software. However, You may do so only on Your own behalf, and not on behalf of any Contributor. You must make it absolutely clear that any such warranty, support, indemnity, or liability obligation is offered by You alone, and You hereby agree to indemnify every Contributor for any liability incurred by such

Contributor as a result of warranty, support, indemnity or liability terms You offer. You may include additional disclaimers of warranty and limitations of liability specific to any jurisdiction.

1.7.4 4. Inability to Comply Due to Statute or Regulation

If it is impossible for You to comply with any of the terms of this License with respect to some or all of the Covered Software due to statute, judicial order, or regulation then You must: (a) comply with the terms of this License to the maximum extent possible; and (b) describe the limitations and the code they affect. Such description must be placed in a text file included with all distributions of the Covered Software under this License. Except to the extent prohibited by statute or regulation, such description must be sufficiently detailed for a recipient of ordinary skill to be able to understand it.

1.7.5 5. Termination

5.1. The rights granted under this License will terminate automatically if You fail to comply with any of its terms. However, if You become compliant, then the rights granted under this License from a particular Contributor are reinstated (a) provisionally, unless and until such Contributor explicitly and finally terminates Your grants, and (b) on an ongoing basis, if such Contributor fails to notify You of the non-compliance by some reasonable means prior to 60 days after You have come back into compliance. Moreover, Your grants from a particular Contributor are reinstated on an ongoing basis if such Contributor notifies You of the non-compliance by some reasonable means, this is the first time You have received notice of non-compliance with this License from such Contributor, and You become compliant prior to 30 days after Your receipt of the notice.

5.2. If You initiate litigation against any entity by asserting a patent infringement claim (excluding declaratory judgment actions, counter-claims, and cross-claims) alleging that a Contributor Version directly or indirectly infringes any patent, then the rights granted to You by any and all Contributors for the Covered Software under Section 2.1 of this License shall terminate.

5.3. In the event of termination under Sections 5.1 or 5.2 above, all end user license agreements (excluding distributors and resellers) which have been validly granted by You or Your distributors under this License prior to termination shall survive termination.

1.7.6 6. Disclaimer of Warranty

Covered Software is provided under this License on an "as is" basis, without warranty of any kind, either expressed, implied, or statutory, including, without limitation, warranties that the Covered Software is free of defects, merchantable, fit for a particular purpose or non-infringing. The entire risk as to the quality and performance of the Covered Software is with You. Should any Covered Software prove defective in any respect, You (not any Contributor) assume the cost of any necessary servicing, repair, or correction. This disclaimer of warranty constitutes an essential part of this License. No use of any Covered Software is authorized under this License except under this disclaimer.

1.7.7 7. Limitation of Liability

Under no circumstances and under no legal theory, whether tort (including negligence), contract, or otherwise, shall any Contributor, or anyone who distributes Covered Software as permitted above, be liable to You for any direct, indirect, special, incidental, or consequential damages of any character including, without limitation, damages for lost profits, loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses, even if such party shall have been informed of the possibility of such damages. This limitation of liability shall not apply to liability for death or personal injury resulting from such party's negligence to the extent applicable law prohibits such limitation. Some jurisdictions do not allow the exclusion or limitation of incidental or consequential damages, so this exclusion and limitation may not apply to You.

1.7.8 8. Litigation

Any litigation relating to this License may be brought only in the courts of a jurisdiction where the defendant maintains its principal place of business and such litigation shall be governed by laws of that jurisdiction, without reference to its conflict-of-law provisions. Nothing in this Section shall prevent a party's ability to bring cross-claims or counter-claims.

1.7.9 9. Miscellaneous

This License represents the complete agreement concerning the subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not be used to construe this License against a Contributor.

1.7.10 10. Versions of the License

10.1. New Versions

Mozilla Foundation is the license steward. Except as provided in Section 10.3, no one other than the license steward has the right to modify or publish new versions of this License. Each version will be given a distinguishing version number.

10.2. Effect of New Versions

You may distribute the Covered Software under the terms of the version of the License under which You originally received the Covered Software, or under the terms of any subsequent version published by the license steward.

10.3. Modified Versions

If you create software not governed by this License, and you want to create a new license for such software, you may create and use a modified version of this License if you rename the license and remove any references to the name of the license steward (except to note that such modified license differs from this License).

10.4. Distributing Source Code Form that is Incompatible With Secondary Licenses

If You choose to distribute Source Code Form that is Incompatible With Secondary Licenses under the terms of this version of the License, the notice described in Exhibit B of this License must be attached.

1.7.11 Exhibit A - Source Code Form License Notice

> This Source Code Form is subject to the terms of the Mozilla Public License, v. 2.0. If a copy of the MPL was not distributed with this file, You can obtain one at http://mozilla.org/MPL/2.0/.

If it is not possible or desirable to put the notice in a particular file, then You may include the notice in a location (such as a LICENSE file in a relevant directory) where a recipient would be likely to look for such a notice.

You may add additional accurate notices of copyright ownership.

1.7.12 Exhibit B - "Incompatible With Secondary Licenses" Notice

This Source Code Form is "Incompatible With Secondary Licenses", as defined by the Mozilla Public License, v. 2.0.

1.8 Indices and Tables

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX