the Computational Geophysics course (GEO4-1427)
Utrecht University

Polina Guseva
(student number: 4951581)

July 2022

Abstract

The current report reflects my understanding of the obtained results with some discussion parts. For each dimension and energy-transferring process FDM (Finite Difference Method) and FEM (Finite Element Method) would be compared together as well as some needed methods of applied math.

Overall,

- numerical approximations
  - for derivatives central (Crank-Nicolson) approach works better than forward and backward since it contains an error smaller in order;
  - for integrals Gaussian quadrature is more precise and flexible for more functions than the mid-point and the trapezoidal rules since there is a polynomial approach. Even 2 quadrature point is already enough for double integrals;
  - for solving a system of linear equations the Gauss-Seidel or the Crank-Nicolson methods are both stable for the researched conditions and the fastest ones.
- solving approaches
  - the explicit method explodes very often already in early stages of computations;
  - implicit methods are more accurate than explicit ones, but their combination (central, Crank-Nicolson) is even better and does not always require more time;
  - the smaller time steps prevent code explosions since in performed exercises a temperature derivative is assumed just as first (backward, forward approaches) or second (central) order accuracy based on Taylor-series expansion. All mentioned above methods require boundary conditions where with increasing derivative approximation accuracy and/or dimension number amount of must-known borders will rise too.
- FDM vs FEM
  - FDM works way faster, especially for a big mesh;
  - FEM can proceed with more flexible shapes of mesh-grid, whereas FDM is applicable only for Cartesian coordinates;
  - FEM is more stable yet it still could explode;
  - accuracy-wise FEM is more precise, especially with some tuning parameters such as SUPG.

In geophysics, the listed above methods are used for all kinds of geological modelling. For example, it is applied for recreating the processes deep down below the Earth's surface (magma movement, crust shifting) and underneath too. For oceanic research water flow could be represented as a bunch of particles. Air string could behave similarly to magma just with a different density. The revised methods could be used also for the Navier-Stokes equations and much more. Furthermore, computational geophysics could be applied for industrial purposes such as gas and oil extraction as well as potential problems caused by that.

Contents

Chapter 1

Numerical approximations

Such mathematical operations as derivation and integration require continuous functions where x is a point and $\lim_{len(x)\to\infty} \Delta x = 0$. Obviously, it is not a case for a real world where $\lim \Delta x \neq 0$. In that case, some mathematical approximation could be used.

1.1 Derivative approximation

If a vicinity around a point is small enough then any function could be approximated as linear. Therefore, any derivative could be calculated as an approximation via its neighbouring point (eq.(11.8-11.9) [1]). The same procedure could be implemented in the second derivative. This approach could be applied to a derivative of any order. However, it could be easier extracted from the Taylor series expansion (eq.(11.5) [1]). However, for this, the price of accuracy should be paid since some values are not counted.

1.1.1 bonus: Higher-order derivative approximation

To reach a more accurate approximation of the derivative more members of the infinite polynomial Taylor series should be included into. However, there is a price for that as more neighbour points are involved. That increases the number of calculations and must-known points. Let's prove the statement made.

Taylor expansion for FD could be calculated backward (1.1), forward (1.2), or central (1.3).

$$f_{i-1} = f_i - hf_i' + \frac{h^2}{2!}f_i'' - \frac{h^3}{3!}f_i''' + \frac{h^4}{4!}f_i^{(4)} - \frac{h^5}{5!}f_i^{(5)} + \frac{h^6}{6!}f_i^{(6)} - \frac{h^7}{7!}f_i^{(7)} + ... \tag{1.1}$$

$$f_{i+1} = f_i + hf_i' + \frac{h^2}{2!}f_i'' + \frac{h^3}{3!}f_i''' + \frac{h^4}{4!}f_i^{(4)} + \frac{h^5}{5!}f_i^{(5)} + \frac{h^6}{6!}f_i^{(6)} + \frac{h^7}{7!}f_i^{(7)} + ... \tag{1.2}$$

By combining backward and forward FD approaches central difference could be achieved (1.3):

$$f_{i+1} - f_{i-1} = 2hf_i' + \frac{2h^3}{3!}f_i''' + o(h^5) \tag{1.3}$$

For backward difference with third-order accuracy let's take the Taylor series before the fourth derivative:

$$f_{i-1} = f_i - hf_i' + \frac{h^2}{2!}f_i'' - \frac{h^3}{3!}f_i''' + o(h^4) \longrightarrow$$

$$f_i' = \frac{f_i - f_{i-1}}{h} + \frac{h}{2}f_i'' - \frac{h^2}{6}f_i''' + o(h^3) = \frac{f_i - f_{i-1}}{h} + \frac{h}{2}\left(\frac{f_{i+1} - 2f_i + f_{i-1}}{h^2}\right) - \frac{h^2}{6}\left(\frac{f_i'' - f_{i-1}''}{h}\right) + o(h^3) =$$

$$= \frac{f_i - f_{i-1}}{h} + \frac{f_{i+1} - 2f_i + f_{i-1}}{2h} - \frac{h}{6}\left(\frac{f_{i+1} - 2f_i + f_{i-1}}{h^2} - \frac{f_i - 2f_{i-1} + f_{i-2}}{h^2}\right) + o(h^3) =$$

$$= \frac{6f_i - 6f_{i-1} + 3f_{i+1} - 6f_i + 3f_{i-1} - f_{i+1} + 2f_i - f_{i-1} + f_i - 2f_{i-1} + f_{i-2}}{6h} + o(h^3) =$$

$$= \boxed{\frac{f_{i-2} - 6f_{i-1} + 3f_i + 2f_{i+1}}{6h} + o(h^3)}$$

Implement the same procedure for forward difference with third-order accuracy:

$$f_{i+1} = f_i + hf'_i + \frac{h^2}{2!}f''_i + \frac{h^3}{3!}f'''_i + o(h^4) \longrightarrow$$

$$f'_i = \frac{f_{i+1} - f_i}{h} - \frac{h}{2}f''_i - \frac{h^2}{6}f'''_i + o(h^3) = \frac{f_{i+1} - f_i}{h} - \frac{h}{2}\left(\frac{f_{i+1} - 2f_i + f_{i-1}}{h^2}\right) - \frac{h^2}{6}\left(\frac{f''_{i+1} - f''_i}{h}\right) + o(h^3) =$$

$$= \frac{f_{i+1} - f_i}{h} - \frac{f_{i+1} - 2f_i + f_{i-1}}{2h} - \frac{h}{6}\left(\frac{f_{i+2} - 2f_{i+1} + f_i}{h^2} - \frac{f_{i+1} - 2f_i + f_{i-1}}{h^2}\right) + o(h^3) =$$

$$= \frac{6f_{i+1} - 6f_i - 3f_{i+1} + 6f_i - 3f_{i-1} - f_{i+2} + 2f_{i+1} - f_i + f_{i+1} - 2f_i + f_{i-1}}{6h} + o(h^3) =$$

$$= \boxed{\frac{-2f_{i-1} - 3f_i + 6f_{i+1} - f_{i+2}}{6h} + o(h^3)}$$

Finally, let's prove the formula for the central difference (1.3):

$$f'_i = \frac{f_{i+1} - f_{i-1}}{2h} - \frac{h^2}{6}f'''_i + o(h^4) = \frac{f_{i+1} - f_{i-1}}{2h} - \frac{h^2}{6}\left(\frac{f''_{i+1} - f''_{i-1}}{2h}\right) + o(h^4) =$$

$$= \frac{f_{i+1} - f_{i-1}}{2h} - \frac{h}{12}\left(\frac{f_{i+2} - 2f_{i+1} + f_i}{h^2} - \frac{f_i - 2f_{i-1} + f_{i-2}}{h^2}\right) + o(h^4) =$$

$$= \frac{6f_{i+1} - 6f_{i-1} - f_{i+2} + 2f_{i+1} - f_i + f_i - 2f_{i-1} + f_{i-2}}{12h} + o(h^4) = \boxed{\frac{f_{i-2} - 8f_{i-1} + 8f_{i+1} - f_{i+2}}{12h} + o(h^4)}$$

1.2 Integration approximation

As it was described for a derivative, integral should be approximated via different workarounds too. Since an integral represents the area (1D) or volume (2D) under the graph, it could be replaced by a number of rectangles (rectangular cuboids). Ideally, their amounts should be $\to \infty$, yet this is not achievable.

1.2.1 The midpoint rule (or rectangle rule) and the trapezoidal rule

The idea behind the midpoint rule (or rectangle rule) approach is to have a height of approximate rectangular at the height of function at the middle of its region ($h = f(x + \Delta x/2)$) (fig.1.1). Thus it would be some part under the line and almost the same part under will be empty, so their areas balance the total (fig.1.1B).

As it was for a derivative approximation, for the trapezoidal rule the best fit to the function line is considered to be a straight line between $f(x)$ and $f(x + \Delta x)$ (fig.1.1). Thus a trapezoid is formed, and its area is used for substitution of the area under the graph.

Table 1.1: Errors of linear approaches

	error$_{midpoint}$	error$_{trapezoidal}$
$\int_0^{\pi/2} x\,dx$	0	0
$\int_0^{\pi/2} cos(x)\,dx$	0.1107	0.5708

For a linear function approximation, both approaches fit perfectly (tab.1.1) since the ideas behind both of them are linear (fig.1.1A). The trapezoidal approach creates the same triangle as the original line, in the midpoint overshoot is exactly the same as undershoot for any straight line.

Whereas, the cosine function is already curved. As it is depicted (fig.1.1B), the midpoint describes it better (tab.1.1) via its overshoot/undershoot balancing philosophy. It could fail if in between it will be a pivot point ($f'(x - \Delta x) * f'(x + \Delta x) < 0$). Nevertheless, it is easily cured by increasing the number of integration points. Whilst a trapezoidal line is always over- or undershoot a curve if there is no change in slope direction, which is the case for a sufficient amount of integration points.

However, it is beyond my understanding why there is some error for a linear function via the trapezoidal rule (fig.1.2A). Since increasing international points still does not change the general shape of a straight line which should match with a trapeze.

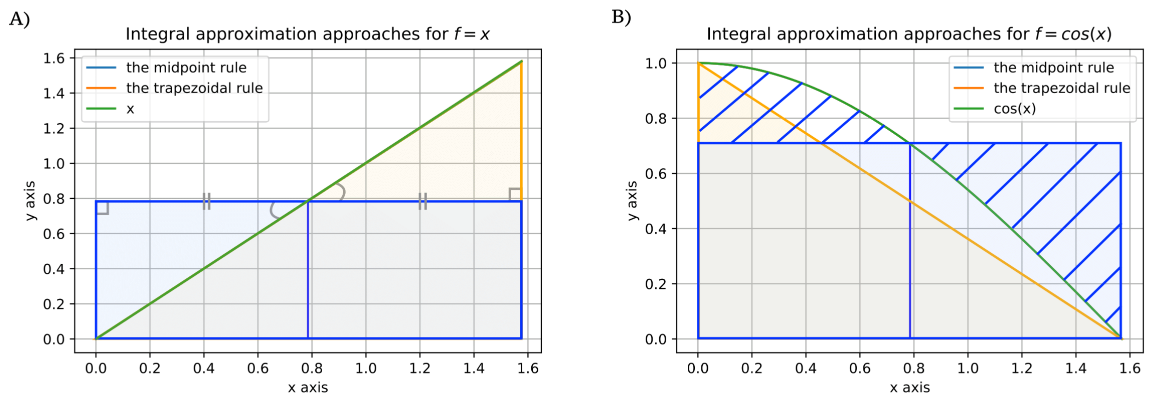


Figure 1.1: Absolute errors in calculating integrals by the midpoint and the trapezoidal rules for linear and cosine functions.

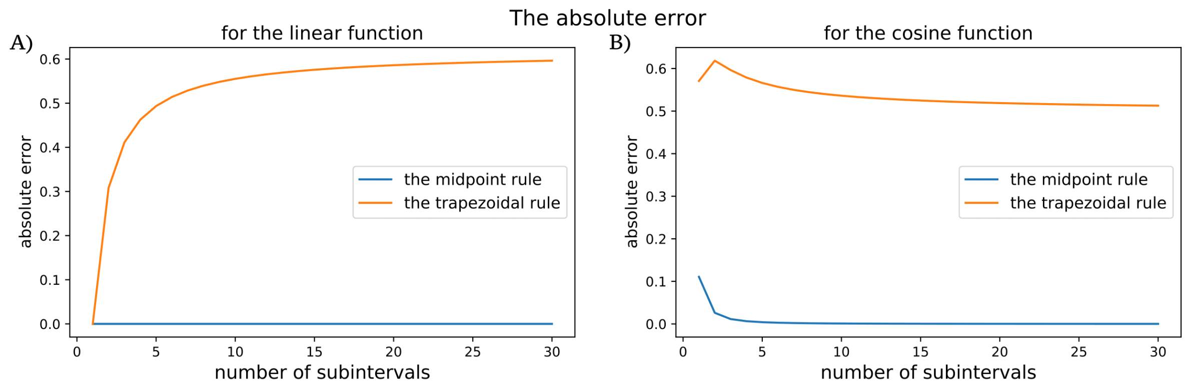


Figure 1.2: Absolute errors in calculating integrals by the midpoint and the trapezoidal rules.

bonus: 2D approach

As for volume approximation in 2D via double integrals, the general rule is that more integration points lead to better results for any rule (fig.1.3, fig.1.4). Nonetheless, the trend for the midpoint rule to be the fittest is remaining. For this case of calculating $\int_1^3 \int_2^4 (x^2y^3 + xy + 1)\,dx\,dy$, the trapezoidal approach is twice as inaccurate in average (fig.1.3).

Also, now two variables and both of them have different impacts. Since y variable has the higher order of power (y^3 vs x^2), it influences total errors for both rules more (fig.1.3, fig.1.4).

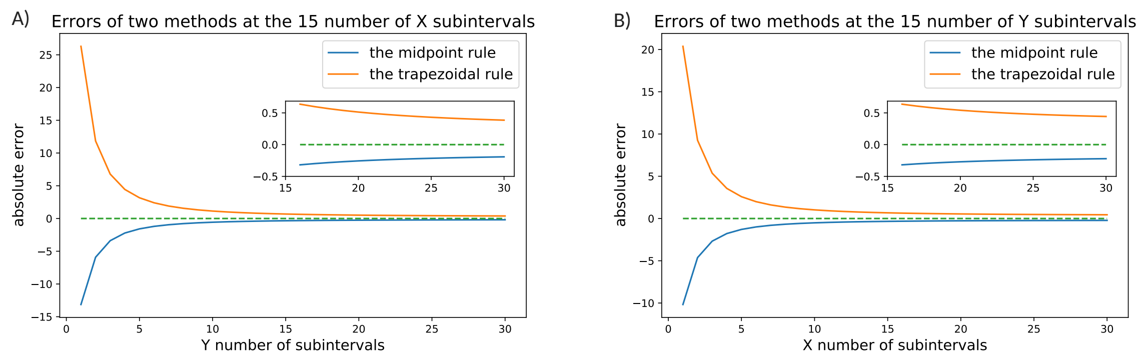


Figure 1.3: Absolute errors in calculating double integral $\int_1^3 \int_2^4 (x^2y^3 + xy + 1)\,dx\,dy$ by the midpoint and the trapezoidal rules.

Summing all mentioned above, the midpoint rule is more accurate since it is more fitted not only for straight lines but also for curves. It is most likely overshooting, where the trapezoidal rule is almost always undershooting. If there are several variables the highest contribution is impacted by those in the highest power.

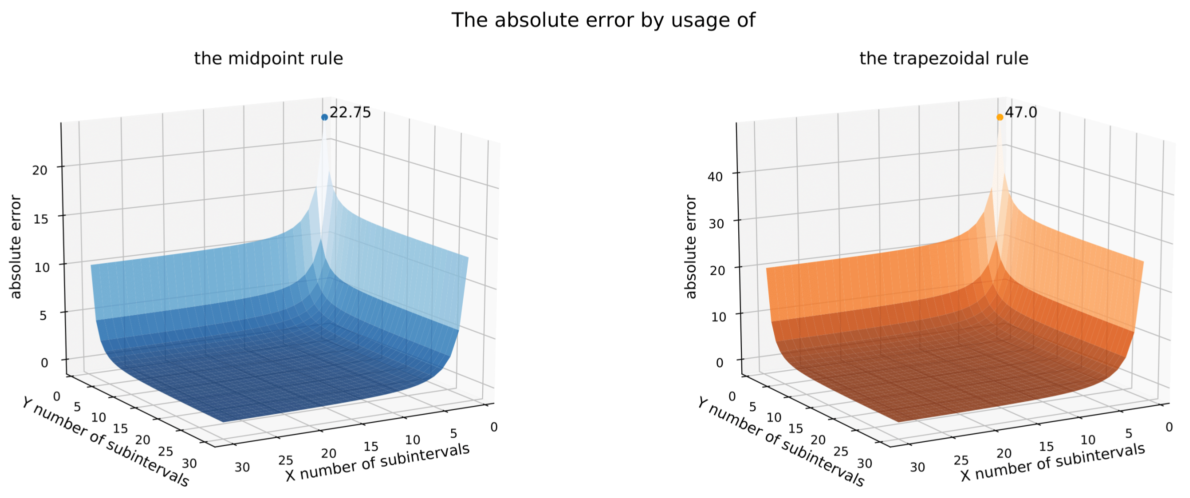


Figure 1.4: Absolute errors in calculating double integral $\int_1^3\int_2^4 (x^2y^3 + xy + 1)\,dx\,dy$ by the midpoint and the trapezoidal rules.

1.2.2 Gaussian-Legendre quadrature algorithm

This approach is based on fitting the curve by a polynomial approximation. Despite the general rule about a better fit of higher-order polynomials still applying, there is not always a need to take a high degree. Yet the accuracy will rise with higher polynomial order, it will also take some computational time and the error of lower degree could be already small enough. For simple functions, a bigger amount of quadrature points can lead to way more accurate results compare to the difference in time (tab.1.2).

Table 1.2: Gaussian quadrature approximation with different amount of points

function	2 quadrature points, $\Delta(\mathrm{I}_{GQ2} - \mathrm{I}_{analytical})$	5 quadrature points, $\Delta(\mathrm{I}_{GQ5} - \mathrm{I}_{analytical})$	Δtime, $\times 10^{-5}$ s
$sin(\pi x + \pi/2)$	4.8124 $\times 10^{-1}$	6.2 $\times 10^{-5}$	-1.60
$\sqrt{x+1}$	2.042 $\times 10^{-2}$	1.78 $\times 10^{-3}$	1.72
$x^4 - x^3$	1.7778 $\times 10^{-1}$	5.6 $\times 10^{-17}$	1.36

As for volume approximation in 2D via double integrals, even the smallest amount of points (2) is enough to reach the really close to an analytical solution result (tab.1.3). The time required for computations is so small that it varies for different code runs and sometimes appears to be higher for a smaller amount of quadrature points.

Table 1.3: Gaussian quadrature for $\int_7^{10}\int_{11}^{14} (x^2 + 4y)\,dx\,dy$

quadrature points	Δresult, $\times 10^{-13}$	required time, $\times 10^{-4}$ s
2x2	-2.2737	5.01
3x3	-2.2737	5.51
4x4	2.2737	7.18
5x5	2.2737	7.09

Overall, the Gaussian quadrature works way more accurate than the midpoint or the trapezoidal rules. As for computational time, even for a high amount of quadrature points for simple functions, it runs fast enough to implement it for processing massive data calculations.

1.3 Iteration techniques

To compute the next generation of something based on the previous iteration methods are used. Here their application to temperature diffusion will be revised as well as efficiency.

1.3.1 Theory: solving linear systems

System of linear equations could be represented as a matrix multiplication: **Ax**=**b**.

$$\begin{cases} a_0^0 x_0 &+& a_1^0 x_1 &+& \cdots &+& a_n^0 x_n &=& b^0 \\ a_0^0 x_0 &+& a_1^0 x_1 &+& \cdots &+& a_n^0 x_n &=& b^0 \\ \vdots &+& \vdots &+& \ddots &+& \vdots &=& \vdots \\ a_0^0 x_0 &+& a_1^0 x_1 &+& \cdots &+& a_n^0 x_n &=& b^0 \end{cases} \longrightarrow \underset{\mathbf{A}}{\begin{bmatrix} a_0^0 & a_1^0 & \cdots & a_n^0 \\ a_0^1 & a_1^1 & \cdots & a_n^1 \\ \vdots & \vdots & \ddots & \vdots \\ a_0^n & a_1^n & \cdots & a_n^n \end{bmatrix}} \times \underset{\mathbf{x}}{\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}} = \underset{\mathbf{b}}{\begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix}}$$

Then for finding a vector of **x** the simple mathematical operation should be performed: **x**=**A**$^{-1}$ **b**. However, in terms of computational efficiency, it is not convenient. Therefore, there are many ways to optimise the process:

$$\mathbf{x}^{(k+1)} = D^{-1}(\mathbf{b} - (L+U)\mathbf{x}^{(k)}) \qquad \text{the Jacobi method}$$

$$\mathbf{x}^{(k+1)} = (\mathbf{b} - U\mathbf{x}^{(k)})(\mathrm{L} + \mathrm{D})^{-1} \qquad \text{the Gauss-Seidel method}$$

$$\begin{aligned} &\mathbf{x}^{(k+1)} = B_1 B_2 \mathbf{x}^{(k)} + \omega(2-\omega)(D+\omega U)^{-1} D (D+\omega L)^{-1}\, \mathbf{b}, \qquad \text{the SSOR method} \\ &\quad B_1 = (D+\omega U)^{-1}(-\omega L + (1-\omega)D) : \text{Backward SOR sweep} \\ &\quad B_2 = (D+\omega L)^{-1}(-\omega U + (1-\omega)D) : \text{Forward SOR sweep} \end{aligned}$$

The Jacobi and Gauss-Seidel methods are called **conditionally stable** since for them to work certain conditions almost always must be satisfied. Thus the matrix **A** have to be strictly diagonally dominant and diagonally nonzero. If these conditions are not met, a system potentially can explode.

The SSOR (Symmetric successive over-relaxation) method has no strictly defined rules for convergence. However, it is proved to work if 1) the relaxation parameter $0 < \omega < 2$ and 2) the matrix **A** is symmetric and positive-definite.

The mentioned above method could be applied to calculate temperature evolution during given conditions (expressed via coefficients into matrices).

1.3.2 Other methods to calculate temperature evolution

As it was used for a derivative approximation, the temperature can 1) be calculated based on previous conditions (forward or explicit), 2) itself define several future points (backward or implicit), 3) be a combination of both (central or Crack-Nicolson). Therefore, there are two symmetric matrices used for this purpose. The Crack-Nicolson method uses respectfully both of them.

1.3.3 Comparison of all approaches

To identify some characteristics (computational time, correlation) of discussed above methods used for temperature evolution calculations, a series of completely random conditions is generated such as 1) matrix size; 2) temperature field and boundaries; 3) dimensionless parameter s, which represents correlate with diffusivity, time step and space step; 4) the relaxation parameter ω; 5) convergence tolerance. To meet convergence conditions $s < 0.25$ and $0 < \omega < 2$. Matrix size is kept between [3,7] for time-saving purposes. Errors are chosen thus to be small enough to converge and big enough to not spend much time. Based on these parameters matrices implicit and explicit methods are created.

Since the Jacobi solver can be really slow a block for 0.5 seconds is implemented. Yet, it does not influence results just cuts fliers.

Also, for comparison, a solver from the scipy package is used.

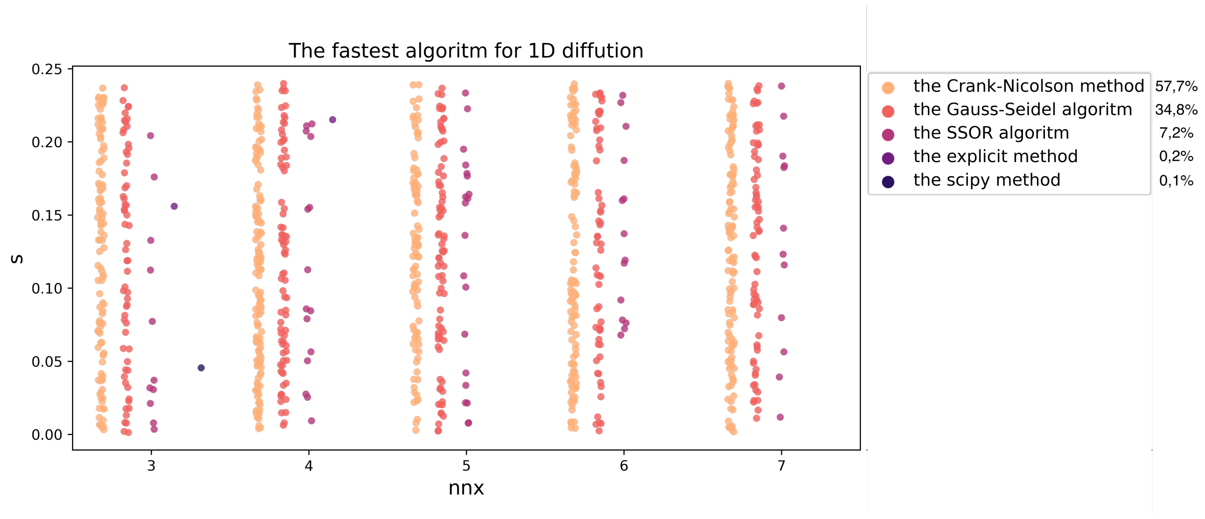


Figure 1.5: Distribution of the fastest algorithms with respect to s and matrix size based on 1000 randomly generated initial conditions. Percentages reflect the possibility for an algorithm to win.

For complete randomness, 1000 sets of parameters are generated which allows researching the impact of each of them separately. All other things being equal, minimal time to compute a result belongs to the Crack-Nicolson (in 57,7% of all generations) and Gauss-Seidel (in 34,8%) methods (fig.1.5).

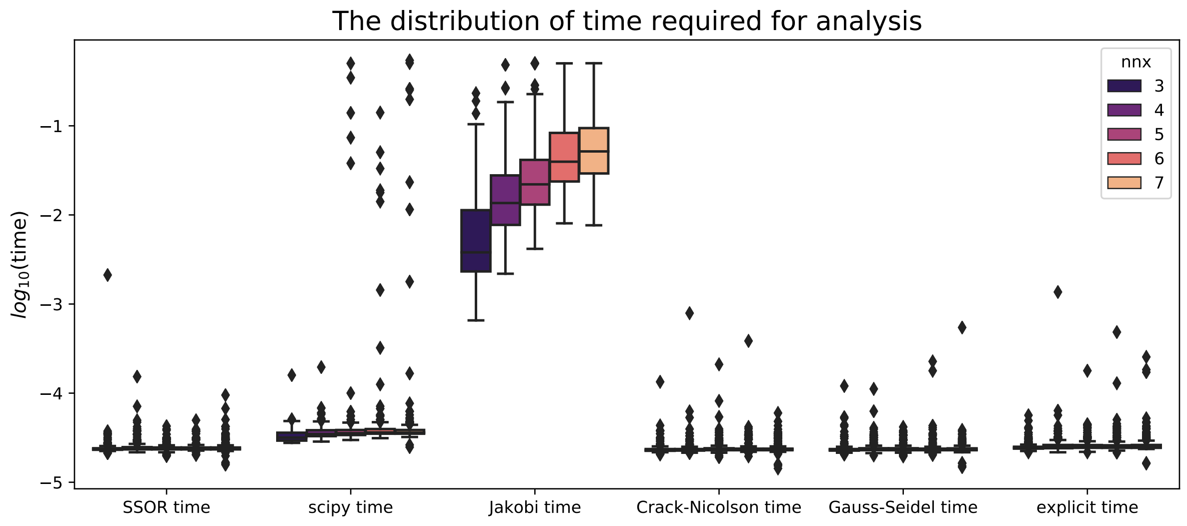


Figure 1.6: Computational time for different methods in terms of matrix size. A logarithmic scale for time is implemented to enhance the impact on time.

There is no obvious correlation between the s parameter and the fastest algorithm (fig.1.5). Meanwhile, for all methods matrix size can positively correlate with the time needed, but that is not the case for everyone (fig.1.6). Also, there is no clear connection between tolerance levels (fig.1.7) and logarithmated time seen with the naked eye either.

For proving all correlations by statistics the Spearman rank-order correlation coefficient is calculated (fig.1.8). This method is used since it reflects the direction and strength between two variables rather than linear correlation (the Pearson correlation coefficient) or possible dependency (the Kendall's τ correlation coefficient).

So, based on obtained results the Jacobi method strongly depends on a matrix size and scipy relies on it weakly too. It could be explained by the poorly performed LU-decomposition of these algorithms, which could lead to a slower convergence.

Also, the Jacobi method has a moderate negative correlation with s. Since the Jacobi algorithm is conditionally stable for better performance it requires a diagonally-dominant matrix. The matrix for this method is implicit and contains $1+2s$ on its main diagonal, where other values sum up to $-2s$. Therefore, decreasing the parameter s diagonally-dominance is lowering too, which leads to more computational time. For the Gauss-Seidel method it is a condition too, but sufficient not a necessary one if a matrix is symmetric and positive-definite. Therefore, in this case, there is no correlation between this algorithm.

Other parameters have no or very weak correlation with examined methods. Therefore, it could be concluded that for this set of boundaries for each of the inputs none of them has a significant influence on *every method*.

To conclude, all discussed above methods could be used for a quick convergence except the Jacobi algorithm. The solver by scipy could potentially slow down the process in case of big matrices, nevertheless, it is a professionally made product and has definitely performed workarounds for unforeseen here conditions.

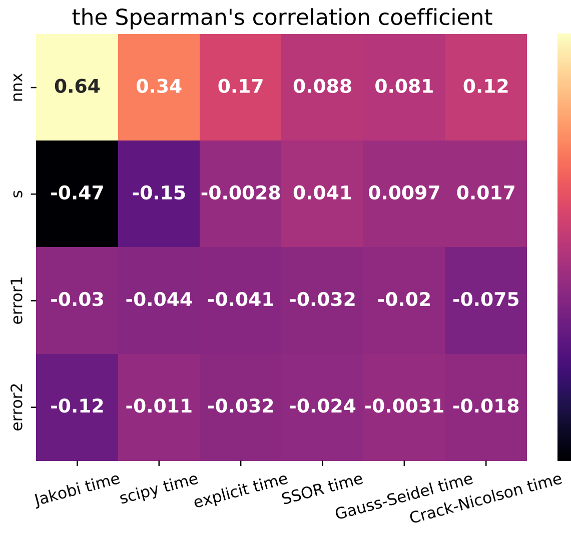


Figure 1.8: Correlation between randomly generated parameters and time needed (log_{10} scale)

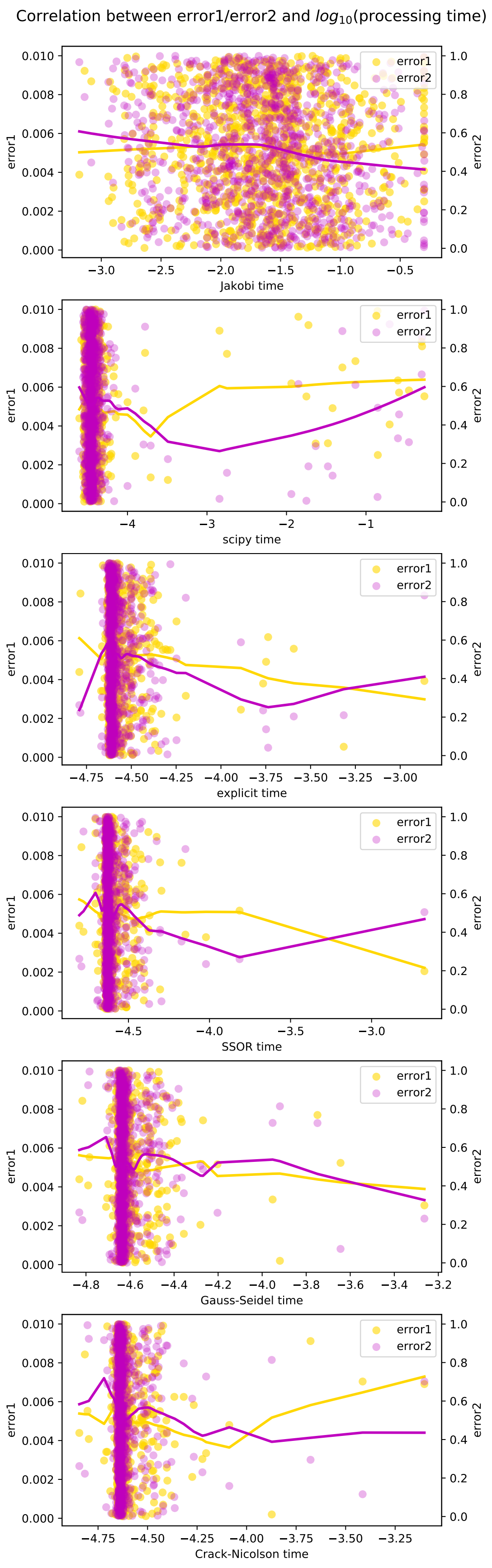


Figure 1.7: Computational time (log_{10} scale) for different methods on basis of two tolerances.

Chapter 2

Philosophy behind FDM and FEM

Both of these methods use the discretisation of a domain into units/points, where to each of them some properties are prescribed. Must-known should be just boundary conditions where their amount depends on dimension. By knowing boundary conditions, necessary conditions and a formula describing a pattern it is possible to recreate the whole system.

2.1 FDM

The Finite Difference method is based on the idea, that the whole property is divided into equally spaced sections and a characteristic is linked to each node enclosing them (fig.2.1). Since the equity between units for 2D and higher dimensions can be provided only by Cartesian coordinates, it imposes some restrictions on the usage of FDM as a method.

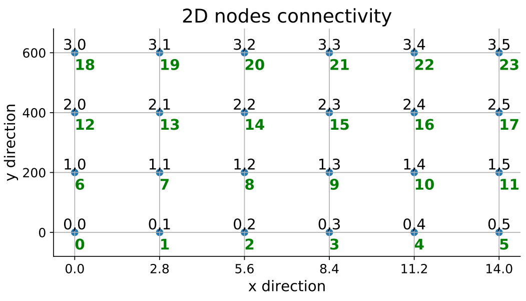


Figure 2.1: Nodes in 2D with coordinates.

2.1.1 Exercise 11.1

By transforming (eq.11.8 [1]) and taking the next further neighbour point ($x_{i+1} = x_i + \Delta x$), i.e. forward differences, the pressure in the crust could be calculated (fig.2.2). Since infinite or close to that amount of points cannot be used, the total amount of nodes is 10. Then, the difference between them is h=3333.3 m. It is way far from the desirable $\Delta x \to 0$ but is enough especially including into account that the initial formula for pressure itself is linear. As it is represented (fig.2.2), pressure is linearly increasing with depth. It happens because there is more mass above it with depth and centrifugal force pushes it to the centre of the Earth. Also, the given density is the same for the whole analyzed unit. Without that the impact on each next step would be different $\to$ non-linear pressure profile. The expected pressure at the bottom is 883 MPa, whereas at the surface p=0 Pa.

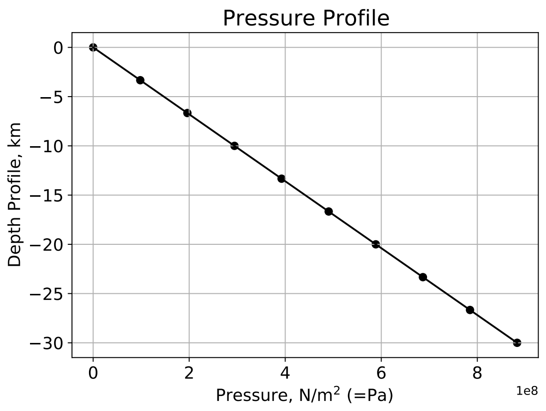


Figure 2.2: The 1D pressure field inside the crust by Finite Difference. Nodes marked by points.

For FDM to work boundary conditions, necessary physical quantities and a formula describing a behavioural pattern are enough to recreate the whole system.

2.2 FEM

The Finite Element approach is used to review the characteristics of an element as a combination of properties by surrounding it nodes (fig.2.3). Since then nodes could be placed without strict order, which allows applying FEM more generally including irregularly shaped elements. As no order is implemented additional matrices with all characteristics are needed, such as icon (as known as a connectivity array), mass matrix (to define the weight of an element based on its nodes), and others if needed (density, velocity, resilience, etc). All of these additionally increase computational time, yet it lets to model the behaviour of many parameters in different shapes flexibly.

2.2.1 Connectivity array

In the case of 1D, it is nothing special since an element only has one node on the left and one on the right, where their numbers are n and n+1 respectively. System in 2D (fig.2.3) or in higher dimensions is more complicated as the number of nodes per element is $2^{\text{dimension}}$. That is where a connectivity array is needed. It contains an array with all nodes for each element. This is also helpful not only in cases of random element shapes but if the numbering of nodes has no order.

In case there are irregularly shaped mesh, an array with node coordinates is also needed to form a mass matrix.

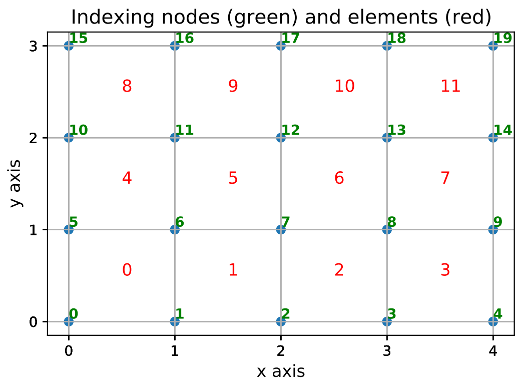


Figure 2.3: Nodes and elements in 2D.

2.2.2 Matrix creation

Once again for the 1D approach, it is all clear with properties of nodes based on elements since there is only one characteristic that could be changed — a length of a section (h_x).

However, in 2D since each element can potentially be different in 4 directions (fig.2.3) this information should be transferred to the characteristics of its nodes. By integrating an original equation, specific parts for specific characteristics could be obtained. For each property of an element, there is a 4x4 matrix resulting from integration over the area. After that based on a connectivity array, each value from this local matrix is prescribed to a specific node in a global matrix (fig.2.4).

$$\begin{bmatrix} L_{0,0} & L_{0,1} & L_{0,2} & L_{0,3} \\ L_{1,0} & L_{1,1} & L_{1,2} & L_{1,3} \\ L_{2,0} & L_{2,1} & L_{2,2} & L_{2,3} \\ L_{3,0} & \boxed{L_{3,1}} & L_{3,2} & L_{3,3} \end{bmatrix} \qquad \begin{array}{c} \text{row coordinate} \\ \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 1 & 2 & 7 & 6 \end{bmatrix} \\ \text{column coordinate} \end{array} \qquad \begin{bmatrix} L_{0,0} & L_{0,1} & L_{0,2} & \cdots & L_{0,n} \\ L_{1,0} & L_{1,1} & L_{1,2} & \cdots & L_{1,n} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ \vdots & \vdots & \boxed{L_{6,2}} & \cdots & \vdots \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ L_{n,0} & L_{n,1} & L_{n,2} & \cdots & L_{n,n} \end{bmatrix}$$

local matrix connectivity array global matrix

Figure 2.4: Scheme how to transfer a value from a local matrix to a global one via connectivity array, where n is a total amount of nodes [2]. The example is given for the 1st element from fig.2.3.

After matrix creation, at the places with unchangeable node values a row with zeros and 1 at the number of a node should be implemented.

Chapter 3

Heat Distribution in 1D

Here, 1D diffusion and advection will be discussed for Finite Difference (FDM) and Finite Element (FEM) methods. Since the task and start conditions are similar for both methods, the results will be compared.

3.1 Diffusion

As it was mentioned before, the difference between FDM and FEM is in looping over nodes or over elements between nodes. If all things are equal (for this conditions from 11.2 are used (fig.3.1)), FDM is an order faster than FEM (tab.3.1) since at each element reference to several nodes is happening. When h$_x$ is not the constant, the calculations take just slightly more (tab.3.1) but not critically.

For a 1 km distance by FDM it takes 16 822 years to completely diffuse a 100°C difference (fig.3.1) to a steady state. While for a 100 km distance by FEM (fig.3.2), it takes 46 945 315 years for the same Δ100°C. The third-order difference in time during the second-order difference in space appears due to integration of the Fick's second law. If $T = T(x)$ and $T = T(t)$:

$$\frac{\partial T(t)}{\partial t} = D\,\frac{\partial^2 T(x)}{\partial x^2} \quad\longrightarrow\quad \int \frac{\partial T(t)}{\partial t} = \int D\,\frac{\partial^2 T(x)}{\partial x^2} \quad\longrightarrow$$
$$\longrightarrow\quad C_0^0\,T(t^2 + ...) + C_1^0 = D\,(C_0^1\,T(x^3 + ...) + C_1^1)$$

That means that diffusion works well for short-distance heat transport but otherwise it takes ages.

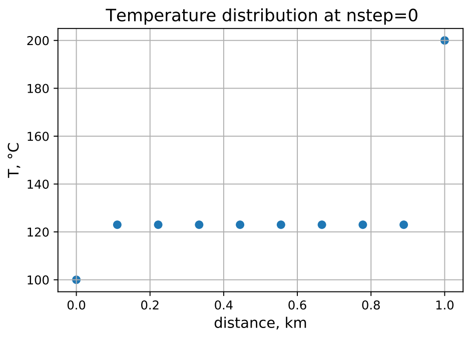


Figure 3.1: Initial conditions FDM. A steady state is a line from $f(0)$ to $f(\text{end})$.

Table 3.1: Computational time for 1D diffusion. All things being equal. FEM* — FEM with ±20% perturbations to h$_x$

time$^{\text{FDM}}$, s	time$^{\text{FEM}}$, s	time$^{\text{FEM}^*}$, s
0.00023	0.05586	0.06072

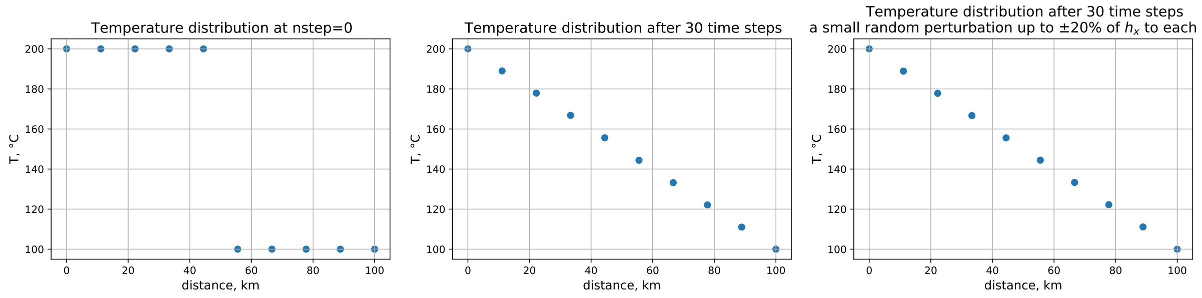


Figure 3.2: 1D diffusion by FEM. A small random perturbation up to ±20% of h$_x$ to each node position inside the domain does not change the recovered steady state.

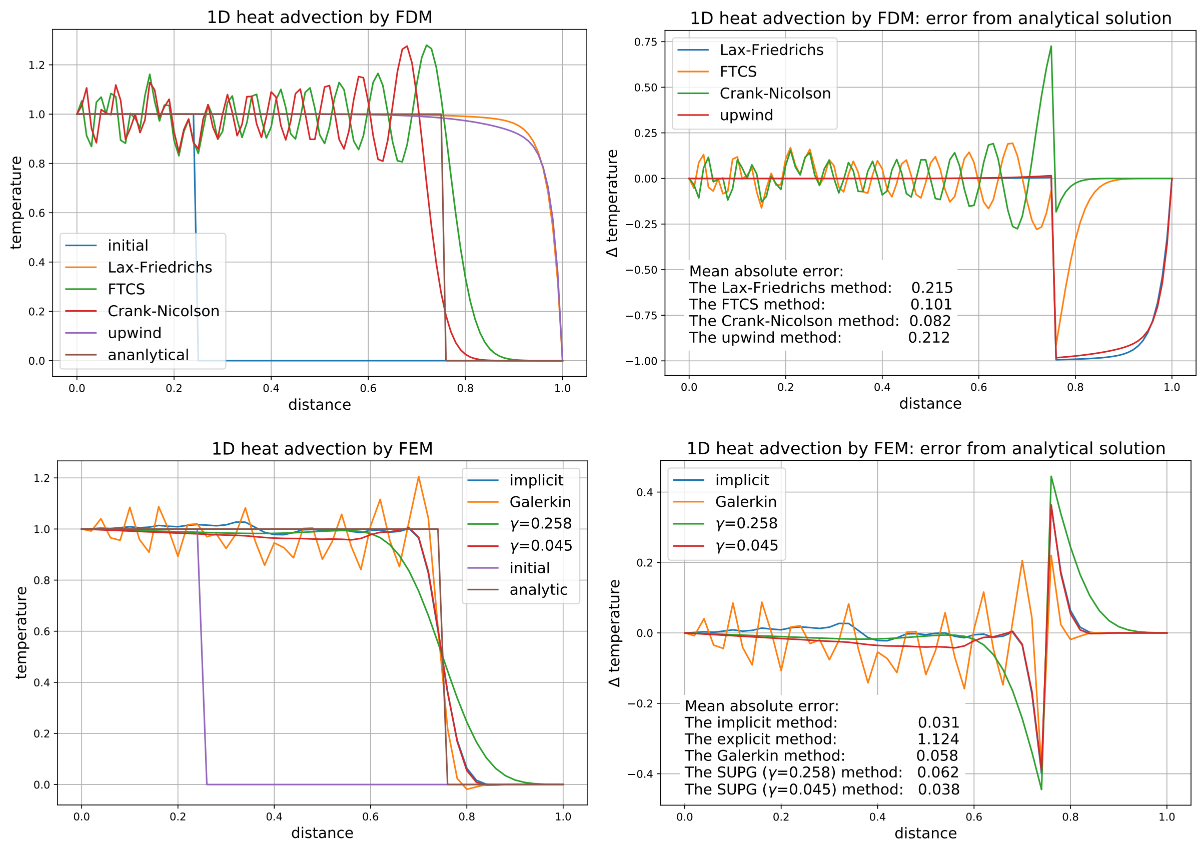
3.2 Advection

Under the same initial conditions FDM methods are generally faster than FEM methods (tab.3.2). Where for FDM forward in time methods (the Lax-Friedrichs method, the upwind method, FTCS (Forward in Time, Central in Space)) are more rapid than mid-point algorithms (the Crank-Nicolson method). All FEM methods run in time almost (during different running sessions the numbers can vary yet the trend is the same) equally including those with the central approach (the Galerkin method, the SUPG (Streamline Upwind Petrov–Galerkin) method).

Table 3.2: Computational time for 1D advection. All initial conditions being equal.

FDM	FTCS		Crank-Nicolson	Lax-Friedrichs	upwind
time, s	0.0264		0.3758	0.0388	0.0288
FEM	explicit	implicit	Galerkin	SUPG (γ=0.258)	SUPG (γ=0.045)
time, s	0.6192	0.7531	0.5666	0.5573	0.5455

Error-wise, FEM methods with an order of magnitude greater are more accurate than FDM ones (fig3.3). The exception is the Crack-Nicolson method, yet it takes 10x more time. Upwind and Lax-Friedrichs perfectly fit at the beginning of energy transfer, but enormously overshoot at the end. Thus amount of extra heat is 20%. On the other hand, the FTCS and Crank-Niciolson methods are varying around the truth but constantly over- or undershoot up 13%. Among FDM algorithms all options are not the best choice to choose. Oppositely in FEM only the Galerkin method (which is the same approach as for the Crack-Nicolson) has over/undershooting problems. Other methods lay out close to reality just slightly diverge at the border. Overall, the best in terms of heat transfer is SURG or implicit by FEM.



Mean absolute error:
The Lax-Friedrichs method: 0.215
The FTCS method: 0.101
The Crank-Nicolson method: 0.082
The upwind method: 0.212

Mean absolute error:
The implicit method: 0.031
The explicit method: 1.124
The Galerkin method: 0.058
The SUPG (γ=0.258) method: 0.062
The SUPG (γ=0.045) method: 0.038

Figure 3.3: 1D advection by FDM and FEM. All initial parameters are the same.

Chapter 4

Heat distribution in 2D

In 2D or higher dimensions, the difficulty in calculations is to take into account all neighbouring nodes and refresh the system simultaneously. Revised below examples by FDM and FEM have analytical solutions and due to the same initial conditions could be compared between one another. For all methods and approaches .gif files over time are created (Appendix A).

4.1 Diffusion

For the discussed below example the given function (eq.(11.80) from Fieldstone) with time will behave as a hyperbole (fig.4.1):

$$\lim_{t\to\infty}\left(T_0 + \frac{T_{max}}{1+4t\kappa/\sigma^2}\ exp\left[-\frac{x^2+y^2}{\sigma^2+4t\kappa}\right]\right) = \frac{1}{t}$$

The FDM explicit approach is conditionally stable and explodes if a time step is too big (fig.4.1A). This happens once again due to numerical approximations which require $\Delta step \to 0$. There is no difference in the end result if different stable time steps are chosen (fig.4.1A). Yet with a smaller size of time steps, a bigger amount of time steps is needed to cover the same time frame. Also, around a border of instability, there are perturbations in the temperature profile (fig.4.1B). On the other hand, the implicit FDM method is unconditionally stable (fig.4.1C), but the computational time required is almost the same for small mesh sizes. Yet for a large number of nodes, the time for implicit is way much more due to the process of matrix creation being ∼nnp^2.

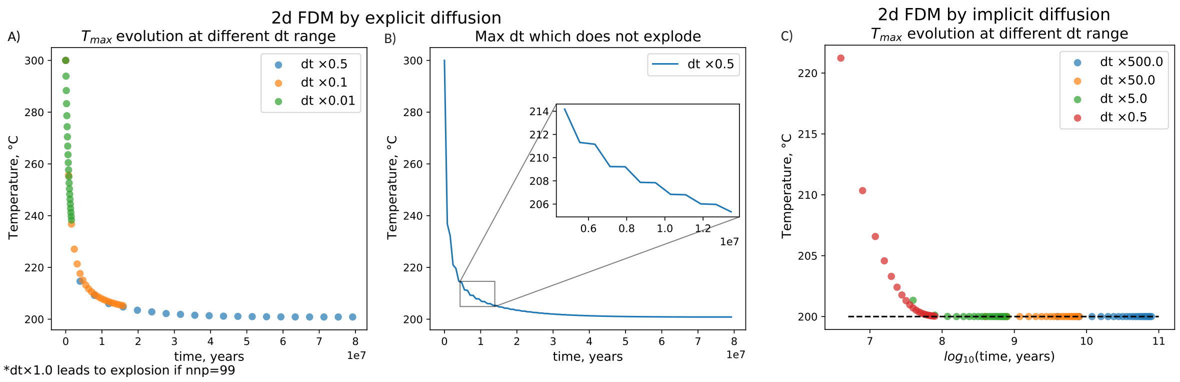


Figure 4.1: 2D FDM diffusion by the explicit (A-B) and implicit (C) method. T_{max} evolution over time in various sizes of time steps. By dt it is meant $min(h_x^2, h_y^2)/2\kappa$.

Geophysics-wise, it is shown again that heat diffuses fast where there is a huge temperature gradient (fig.4.1-4.2). But to reach a steady state with all heat equally distributed could take more than ten million years for an area of 100km x 80km (fig.4.1C). That means that for a larger distance ΔT in order of a hundred °C will diffuse for ages. Also, to recreate such processes by FDM without any pretensions to preciseness a small mesh can work well enough (fig.**??**).

For reasons unknown to me, FDM works ∼27.5x times faster than FEM all else being equal.

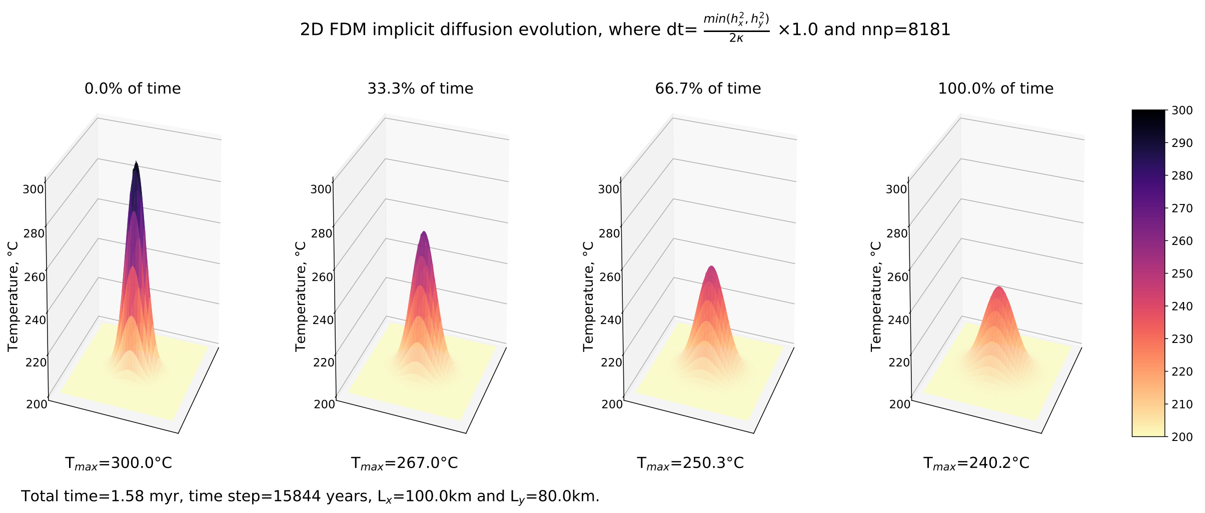


Figure 4.2: 2D FDM diffusion by the implicit approach in the first short time frame.

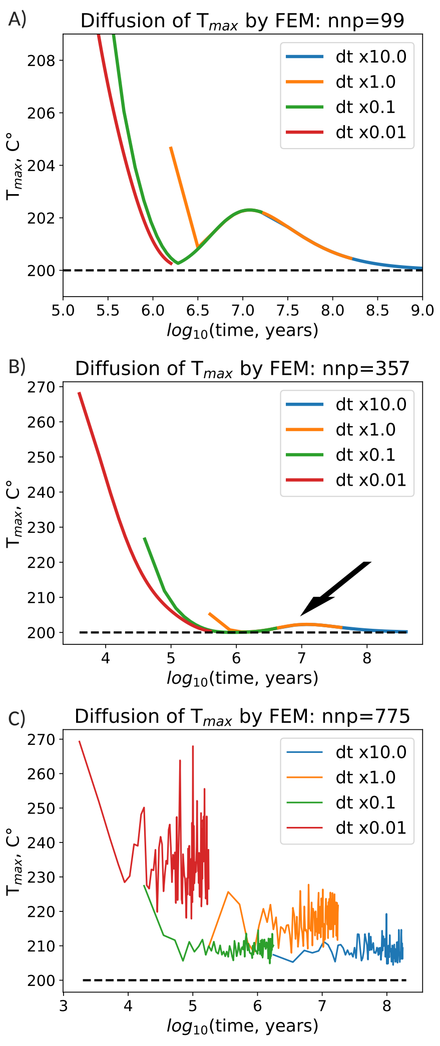


Figure 4.3: 2D diffusion by FEM GQ: errors(A,B) and instability via T_{max}.

In FEM there are at least two mathematical approaches how to calculating elemental matrices: 1) via symbolic integration of weak forms of an initial equation, 2) via integral approximation by Gaussian quadrature (GQ).

For the first option, the sympy package in Python could be used. The disadvantage of this method is a really low speed for the creation of an elemental matrix (tab.4.1). However, this step could be performed just once and does not depend on mesh size. For the second option, 2 quadrature points are used since as proved before, for calculations of double integrals it is already good enough (tab.1.3).

Both of them are conditionally stable too (fig.4.3, 4.4). Moreover, both of them have bouncing back up temperature after almost reaching ΔT=0 (fig.4.3(A, B arrow), the same is with sympy). Probably it is an inaccuracy caused by returning back or reflecting from the vicinity since there are too many surrounding nodes involved. Maybe that is the reason why the temperature in the systems with a large number of nodes bounce around some point too (fig.4.3, 4.4).

Table 4.1: Computational time required for 2D diffusion for a domain 100km x 80km, where nnp=99 and dt=$\frac{min(h_x^2,h_y^2)}{2\kappa}$ ×0.1.

time, s	matrix creation	loop over time	**total**
FDM explicit	-	0.027	**0.027**
FDM implicit	0.001	0.01	**0.011**
FEM sympy	4.667	0.324	**4.991**
FEM GQ	0.083	0.286	**0.369**

Error-wise, all methods differ from one another in order of <2% (fig.4.5). The most significant mismatch is between all applied math models and the analytical solution. Whereas, the values of both FDM methods are almost identical. Also, symbolic integration (sympy) matches well with both of them too. Consequently, the Gaussian quadrature stands out of the three of them by almost the same value.

To conclude, all revised methods give nearly the same result. Even though it varies from analytical solution, the difference is less than critical. In terms of computational time (*for a small mesh), explosibility, and error preciseness (even in comparison with analytical), the implicit method by FDM shows the best result.

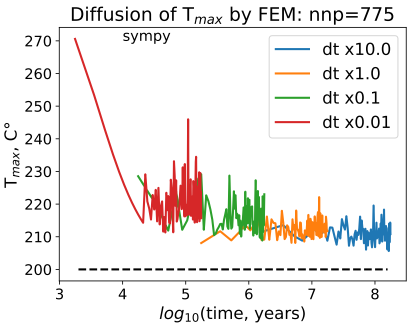


Figure 4.4: 2D diffusion by FEM symbolic integration: instability.

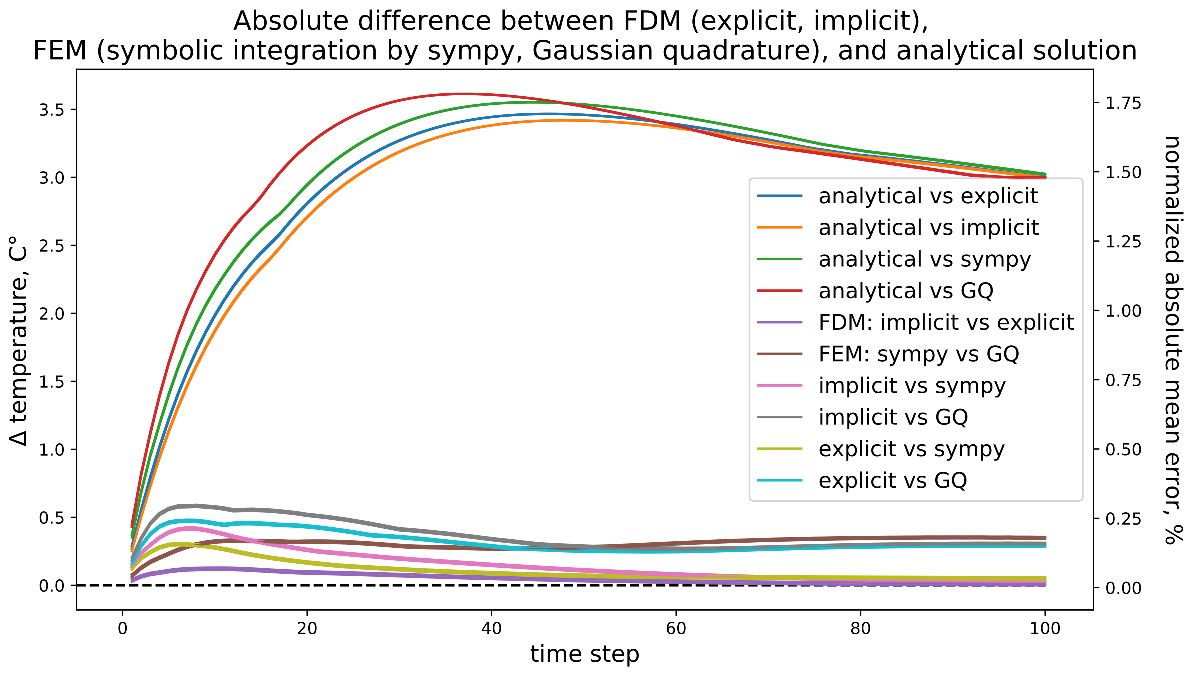


Figure 4.5: Mean absolute error for 2D diffusion between FDM, FEM, and analytical solution. Conditions: nnp=99, for the FDM methods dt=$\frac{min(h_x^2,h_y^2)}{2\kappa}$ × 27.5/300, for the FEM methods dt=$\frac{min(h_x^2,h_y^2)}{2\kappa}$ × 1/300. This step was performed due to ambiguous to me difference in diffusion speeds. This value was found empirically as true scientists usually do. Analytical solution was calculated with dt for FEM. Such a low value was chosen to prevent explosions.

4.2 Advection

The same patterns in speed, convergence, and stability as highlighted in previous sections are applied for 2D advection too. Time-wise, overall FDM methods are way faster than central by FEM (tab.4.2).

Once again the FDM explicit method is conditionally stable for 2D advection too (fig.4.6). Yet due to derivative approximations only with enough small time steps the system is not exploding. The process of the explosion is triggered by accumulating curly heat waves behind a moving object. There down holes cause more temperature fall and up picks — more temperature ups. In the end, the domain consists only of opposite extremes. The implicit approach by FDM (or the Lax-Friedrichs method) not only moves prescribed heat but also diffuses it (fig.4.7). Instead of the initially shaped cone (math yellow line), the temperature is still distributed as a cosine (nice green dotted line). Whereas, the central Crank-Nicolson method (fig.4.8) is relatively stable (or at least for the revised period of time). It also saves the given amount of heat yet the waves of the future potential explosion are there (fig.4.11C).

Table 4.2: Computational time required for 2D advection, where nnp=961. All FEMs are central.

time, s	**FDM** explicit	**FDM** implicit	**FDM** central	**FEM** sympy	**FEM** GQ
matrix creation	-	-	0.1664	36.7550	0.4106
loop over time	0.4366	0.4385	0.1526	19.4420	21.8273
total	**0.4366**	**0.4385**	**0.3190**	**56.1970**	**22.2379**

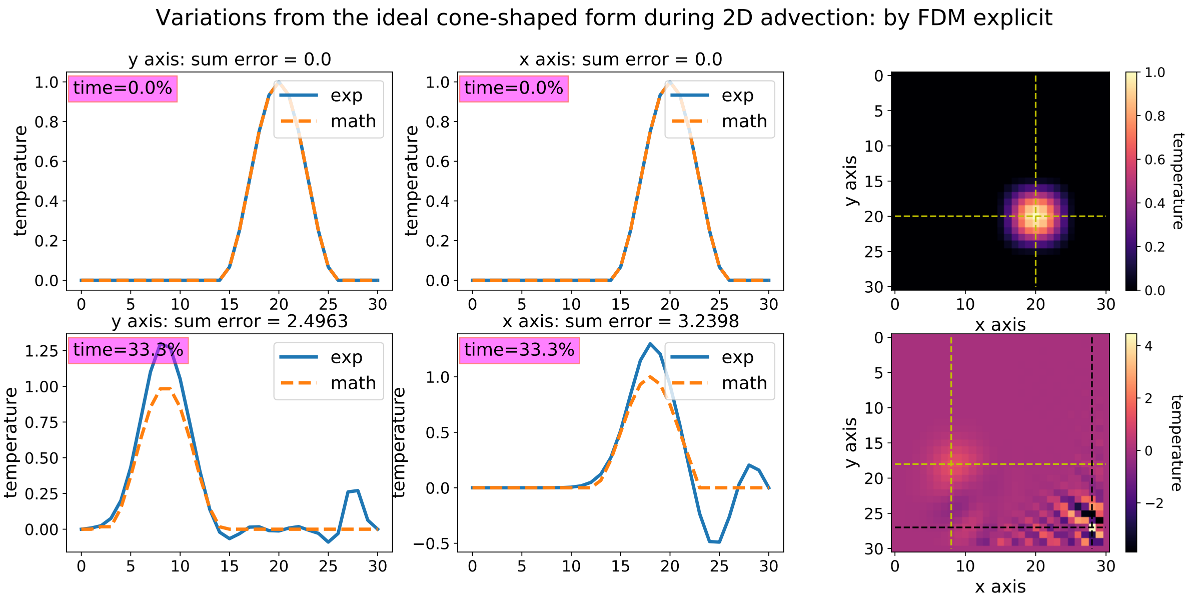


Figure 4.6: 2D explicit FDM advection. Yellow lines represents cross-sections at the left. Orange lines stand for a prescribed initially cone. Black dotted lines show coming explosion by heat ups and downs.

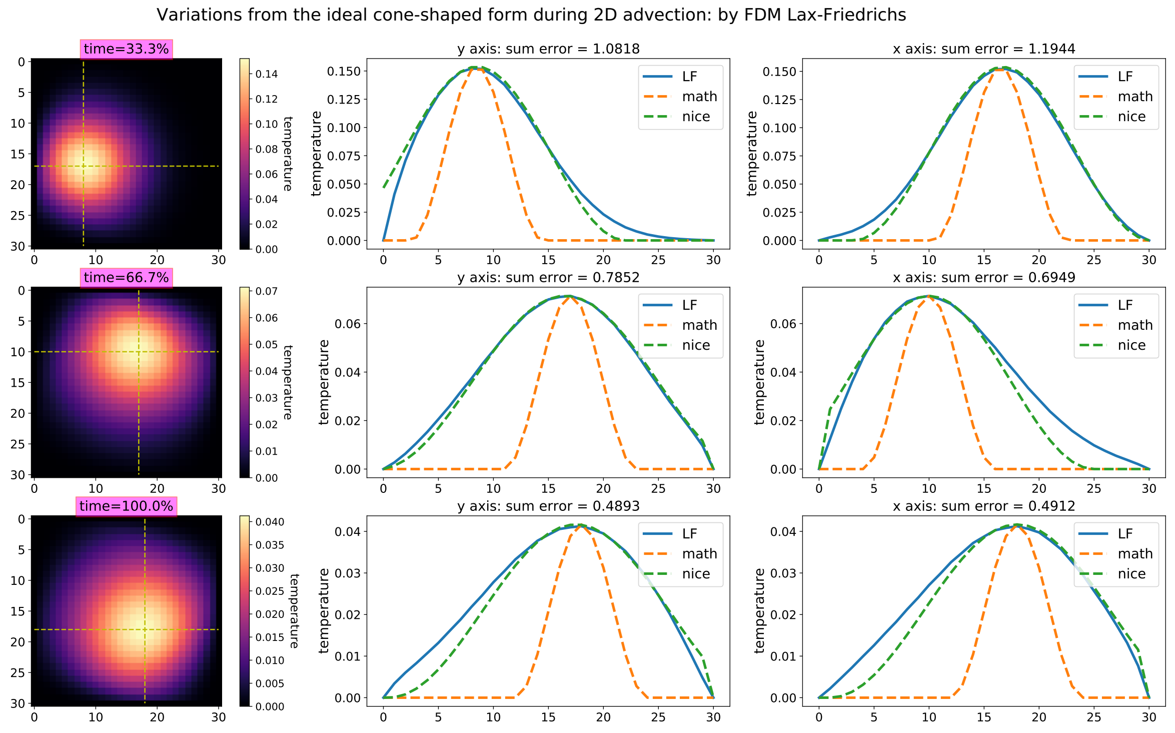


Figure 4.7: 2D implicit FDM advection. Nice green dotted lines represent the most fittable cosine. 'LF' is for Lax-Friedrichs.

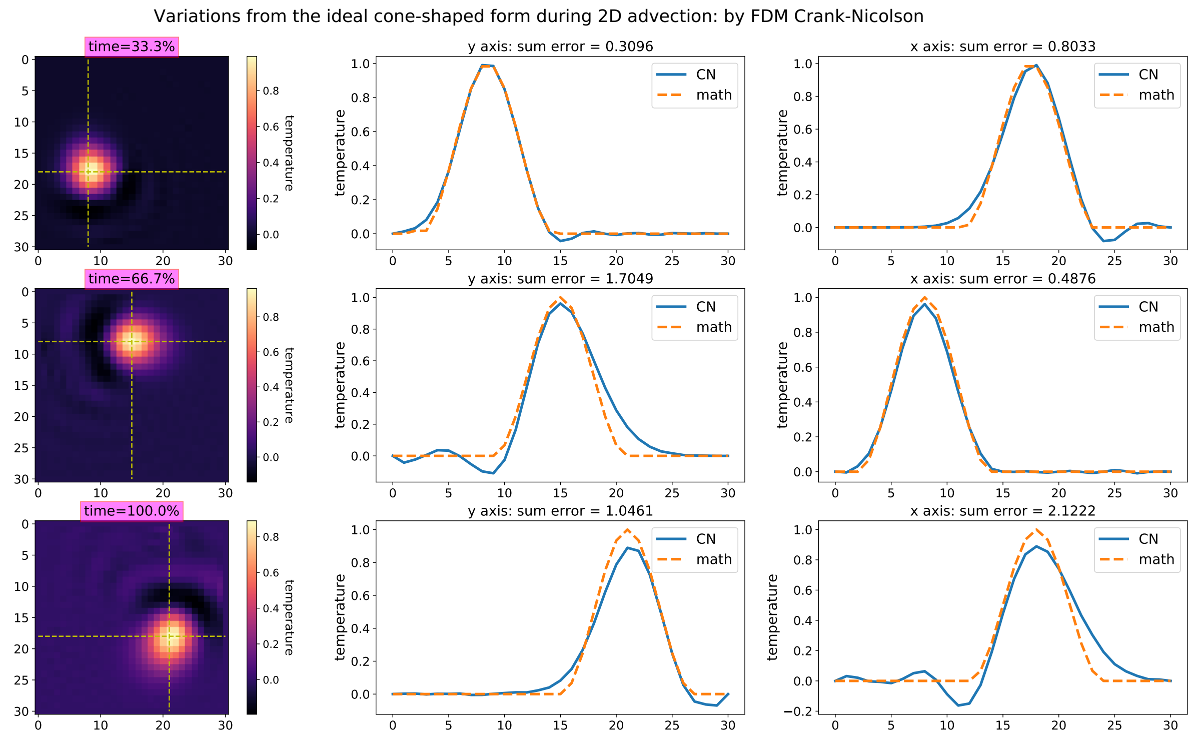


Figure 4.8: 2D central FDM advection by the Crank-Nicolson method. 'CN' is for Crank-Nicolson.

In contrast, the FEM advection both by symbolic integration and by Gaussian quadrature (for all α=1/2) produces excellent results (fig.4.9). It does not show any big signs of a potential explosion and converges after the whole round indeed close to the initial conditions. Nonetheless, there are still some tiny waves and some elements have negative temperatures (fig.4.10, 4.11). Although these results cost time (tab.4.2) since the matrix that goes to the solver is not extremely diagonally dominant. On the other hand, for the FDM methods, there is only the main, one before, and one after diagonals are filled. Probably that difference is one of the reasons that ensure the stability of the FEM approaches.

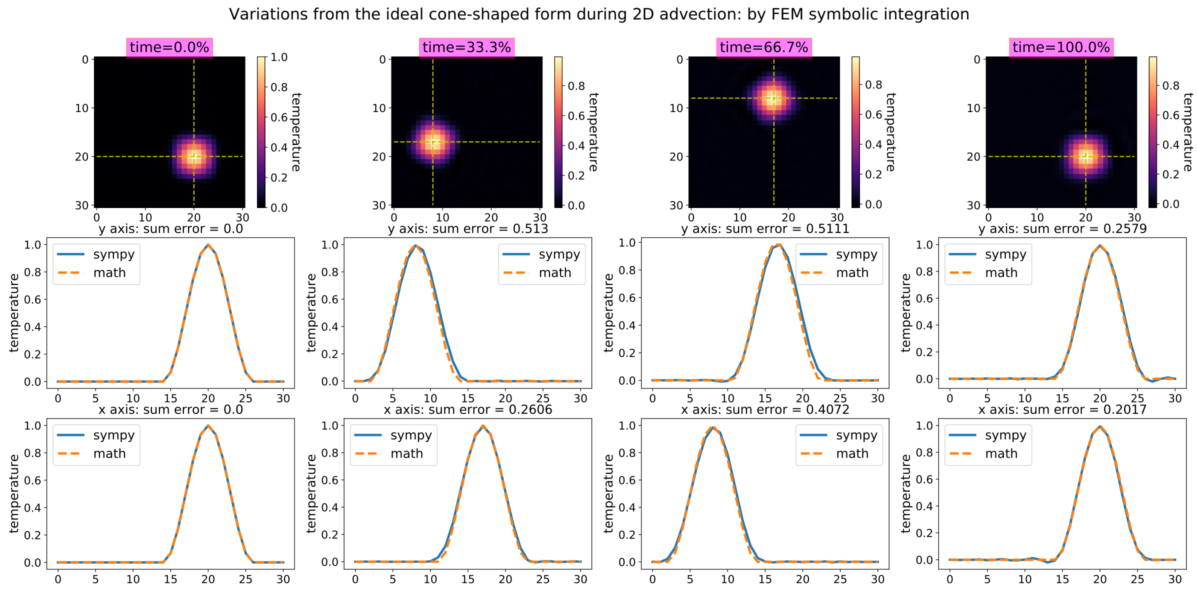


Figure 4.9: 2D FEM advection by symbolic integration. No SUPG is applied. The GQ plot is the same.

As for the implicit and explicit approaches for FEM, even with SUPG correction there are both showing worse results than the central one (fig.4.10). The correctly chosen gamma coefficient will improve convergence and/or postpone the explosion (fig.4.10). As in 1D (fig.3.3) γ=0.045 is more precise.

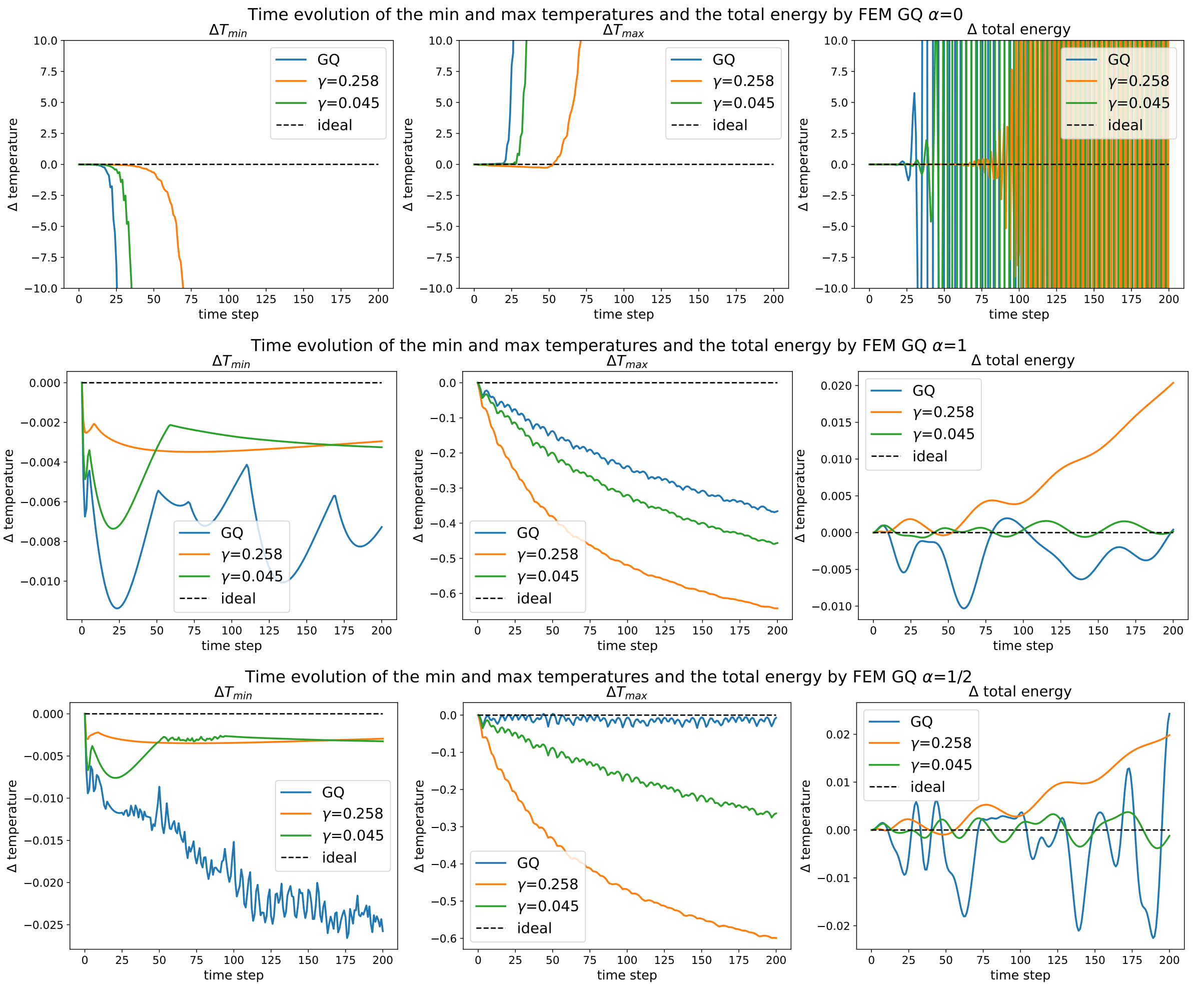


Figure 4.10: 2D FEM advection by GQ with applying SUPG and with the implicit (α=1), explicit (α=0), and Crank-Nicolson (α=1/2) approaches.

Overall, for 2D advection, FEM is more accurate than FDM (fig.4.11), especially with SUPG tuning. The Crank-Nicolson approach prevents and/or delays instability and has better convergence than completely explicit or implicit. If possible explicit should be avoided (even more unstable for FEM (fig.4.11)). Implicit in FDM could cause the effect of diffusion (fig.4.7). Yet in terms of time (tab.4.2) FEM is two orders of magnitude slower than FDM. This could be crucial for big mesh sizes i.e. for detailed resolution since computational time exponentially depends on the number of nodes/elements.

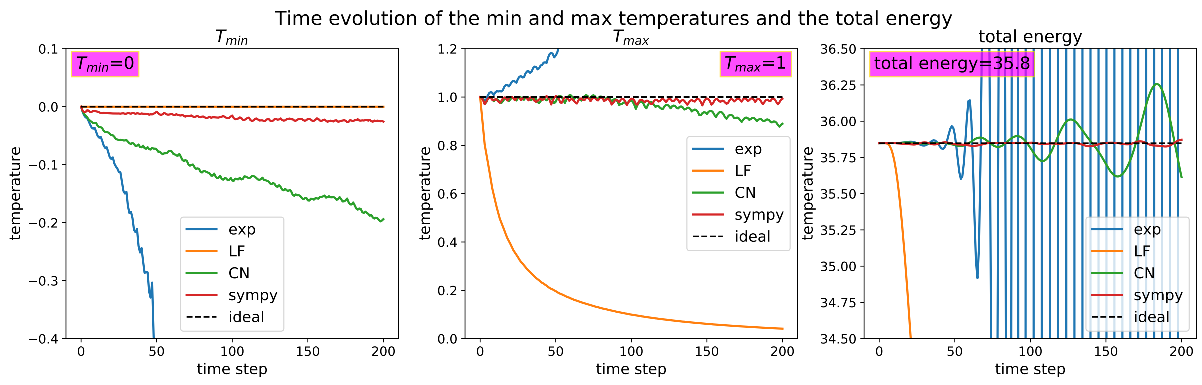


Figure 4.11: 2D advection: deviations by FDM and FEM methods and approximation approaches.

Chapter 5

Project: 2D diffusion by FEM in annulus

The current project is to recreate a diffusion in a 2D doughnut. For that, the connectivity array is reshaped to represent not ideal circles but close to them polygons. Also, since element size varies between elements all matrices are mapped for a general quadrilateral form via the Jacobian matrix. Finally, the assembled system is solved over time and the result is compared to the analytical solution.

5.1 Connectivity and coordinate arrays

Since an element size is not equal there should be not only information about linking elements to nodes but also about the coordinates of all nodes.

As for the icon, the only difference between the Cartesian version is that nodes at the left and right sides of a rectangular now merge into one closing the circle (fig.5.1). Therefore, the number of nodes and elements is the same in the x-direction ($nnx = nelx$). Thus, the usual counter-clockwise numbering is restored and all shared nodes are considered (fig.5.2).

The coordinates of the nodes are calculated based on an angle and a radius from the centre of the coordinates $(0, 0)$. The rotation angle (α) is 360°/number of elements in the x-direction. Since h_y for all elements/nodes is the same, the fixed step (r) is based on the total length in the y-direction. So, for a node with the index n:

$$\alpha_n = -\alpha\pi/180^\circ\ n \qquad r_n = r\ (n//nnx + 1),$$

where $n//nnx$ is a nodal index (fig.2.1) in the y-direction; and:

$$x_n = cos(\alpha_n)\ r_n \qquad y_n = sin(\alpha_n)\ r_n,$$

thus, there is an additional array with all nodes coordinated stored as (x_k, y_k). It provides coordinates for the assessment of the weighted mass of an element.

Hereby, there are two supporting arrays:

- between elements and their nodes;
- between nodes and their coordinates.

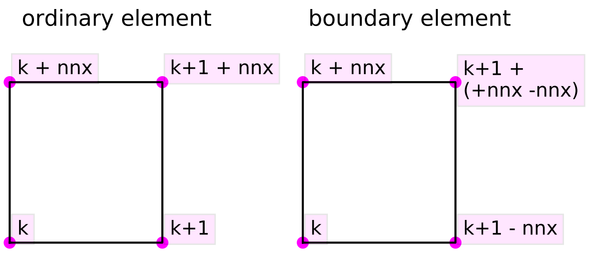


Figure 5.1: Numbering nodes around ordinary and boundary elements, where k is an index of element.

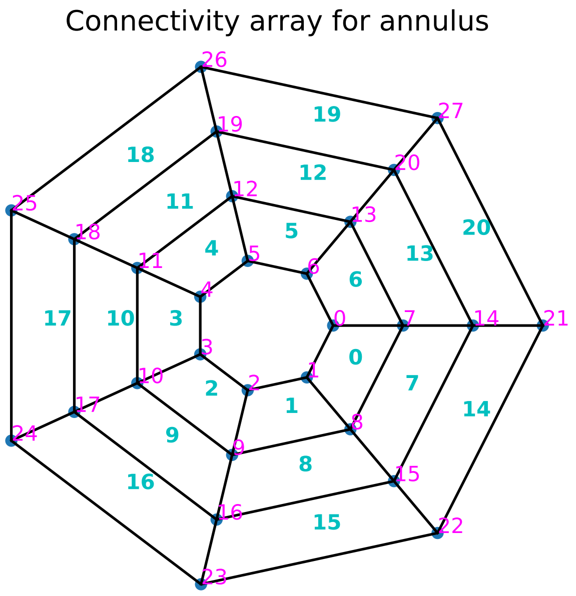


Figure 5.2: Connection (aka icon) between elements and nodes in 2D annulus, where $nelx$=7 and $nely$=3.

5.2 Elemental matrices for a generally shaped quadrilateral

For the purpose of time- and energy-saving, one element of annulus is considered not as a truncated circular sector, but as a trapeze. Therefore, all elemental matrices should be rewritten for mapping a general quadrilateral figure by bilinear basic functions. For that, the Jacobian matrix and determinant are used.

First, all x- and y-coordinates of 4 nodes are transformed into $x(r,s)$ (eq.(8.131) from Fieldstone [1]) and $y(r,s)$ (eq.(8.132) [1]). The Jacobian matrix is a derivative by r,s of these new coordinates (eq.(8.125-8.128) [1]) and by that transfers real coordinates into a [-1,1] × [-1, 1] box. The Laplace operator for K_d^e is displaced by J^-. Therefore, generalized elemental matrices are:

$$M^e = \rho C_p \int_{-1}^{1}\int_{-1}^{1} NN^T \ |J_{det}| \ drds \qquad\qquad K_d^e = k\int_{-1}^{1}\int_{-1}^{1} B_* B_*^T \ |J_{det}| \ drds$$

where $B_* = BJ^-$.

5.3 Assembly, solving, comparison

Overall, these steps are similar to already performed exercises.

The assembly of a global matrix is the same as for 2D diffusion with rectangular, with one exception of generating M^e and K_d^e not just ones but for each element separately. These matrices are constructed outside of the time loop since the properties of elements (such as mass and diffusivity) do not change over time.

The first experiment is a diffusion inside an annulus with boundaries of 0 inside and 1 outside, the initial condition is the domain halved between these two values (fig.A.4) and dt= half-length of a domain/amount of time steps.

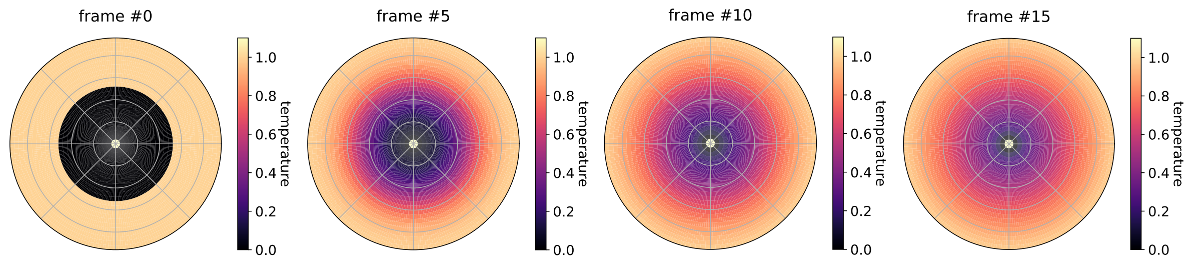


Figure 5.3: Project: experiment №1. The first 15 time frames, where nnt=20. Ripple effect due to high resolution of plot grid (nnp=648).

For the current exercise, there is an analytical solution: a steady state for the diffusion in polar coordinates:

$$\frac{\partial T}{\partial t} = \Delta T = 0 \qquad \longrightarrow \qquad \Delta T = \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial T}{\partial r}\right) + \frac{1}{r^2}\frac{\partial^2 T}{\partial \theta^2} = 0$$

where Δ is the Laplace operator, r is a radial distance, and θ is an angle.

Since in this exercise, the system is symmetrical (sum of all absolute errors within all layers for $nnp = 648$ after $dt \times nnt = 3(r + L_y) = 2.9 \times 10^{-13}$), therefore:

$$\frac{\partial^2 T}{\partial \theta^2} = 0 \qquad \longrightarrow \qquad r\frac{\partial T}{\partial r} = \text{constant} \qquad \longrightarrow \qquad T(r) = \int \frac{\text{constant}}{r} dr = C_0 ln(r) + C_1$$

where the constants C_0 and C_1 could be found due to known temperature at boundaries.

$$\begin{cases} T(r) = T_{inside} \\ T(r+L_y) = T_{outside} \end{cases} \qquad \begin{cases} C_0 ln(r) + C_1 = T_{inside} \\ C_0 ln(r+L_y) + C_1 = T_{outside} \end{cases} \qquad \begin{cases} C_0 = (T_{inside} - T_{outside})/ln\frac{r}{r+L_y} \\ C_1 = T_{inside} - C_0 ln(r) \end{cases}$$

With more time running the difference between the FEM solution and the analytically obtained one is becoming lower (fig.5.4). Already at $nnt \times dt = 5(r + L_y)$ the pick of error is less than 1%. Time-wise, Computational time required for the mentioned above running time and mesh size is 3.3 seconds. As for boundary conditions, they are always correct (fig,5.4).

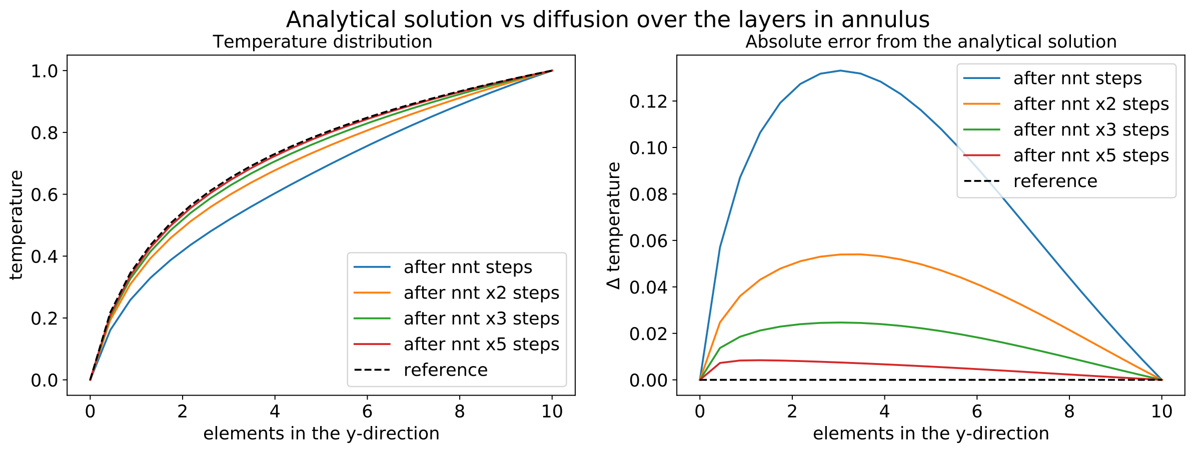


Figure 5.4: Project: experiment №1. Comparison to the analytical solution, where $dt = (L_y + r)/nnt$.

Figure 5.5: Animated diffusion in annulus.

Thus, the current model by FEM, Gaussian quadrature with 2 quadrature points, and bilinear basic functions works relatively precisely, converges and does not explode (fig.5.5). In terms of time, it is fast for at least small matrices.

Bibliography

[1] C. C. Thieulot, “Fieldstone: The finite element method in computational geodynamics,” 8 2019.

[2] D. May, M. Frehner, T. Philippe, and A. Bauville, “Introduction to finite element modelling in geosciences,” 2016.

Appendix A

2D heat transfer animation

Figure A.1: 2D diffusion by FDM: explicit, explicit with no limits, implicit, implicit with no limits.

Figure A.2: 2D advection by FDM: explicit (FTCS), implicit (Lax-Friedrichs), Crank-Nicolson.

Figure A.3: 2D diffusion by FEM Crank-Nicolson: symbolic integration, symbolic integration no limits, GQ, GQ no limits.

Figure A.4: 2D advection by FEM Crank-Nicolson: symbolic integration and Gaussian quadrature.