Homework 03
Implementation and Analysis of Time Integration Methods for Parabolic PDEs in FVM

Batuhan Çoruhlu 2397735 - Hasan Kaan Özen 2378586

December 16, 2023

Abstract

This report focuses on the explicit and implicit time integration methods for a parabolic unsteady diffusion equation. The implementation details of the second-order Adams Bashfort Method and the second-order Backward Differencing method are presented. These methods along with other time integration methods are tested throughout various mesh types and resolutions and the infinity norm of error and L2 norm of error results are presented in tables. The results show that implicit methods provide more stable and fast results whereas the explicit methods implemented have stability problems.

Contents

1 Introduction **3**

2 Temporal Discretization Methods **3**
2.1 CFL Condition 4
2.2 Explicit Methods 4
2.2.1 Forward Euler 4
2.2.2 Runge Kutta 5
2.2.3 Adams Bashfort 6
2.3 Implicit Methods 6
2.3.1 Backwards Euler 6
2.3.2 Backwards Differencing 7
2.4 Comparison of Methods 7

3 Implementation Details **8**
3.1 Implementation of Adams-Bashforth Method 8
3.1.1 setAdamsBashforth 8
3.1.2 runAdamsBashforth 9
3.2 Implementation of BDF Method 11
3.2.1 setBDF 11
3.2.2 runBDF 11

4 Discussion and Results **13**
4.1 Details and Results of the Cases 13
4.1.1 Case 1 13
4.1.2 Case 2 16
4.1.3 Case 3 16

5 Conclusion **18**

6 References **19**

1 Introduction

The fourth dimension, time is an important system parameter that affects the solution entirely. There are a variety of methods and discretizations that tackle steady-state problems. However, most problem definitions contain the time parameter and the previously discussed solution methodologies converge to inaccurate results. In order to achieve accurate results, several different time integration methods are developed and evolved to fit into the Finite Volume Methods. However, these time integration methods come in various forms for different orders of solutions. Depending on the case, one must choose a method that will provide stable and fast results.

In this report, the second-order forms of the time integration methods, Adams Bashford and Backward Differencing are implemented. Moreover, these methods along with Forward Euler, Backward Euler, and Runge Kutta methods are tested under the parabolic unsteady diffusion discretization problem of the form presented in Equation 1. The results are presented in tables where the norms of errors are compared. After the comparison, the report progresses with new iterations based on the previously presented result and focuses on building discussions for the case and the integration methods.

$$\frac{\partial q}{\partial t} + \nabla \cdot (k \nabla q) + Q_s = 0, \tag{1}$$

2 Temporal Discretization Methods

Transient phenomena introduce an additional spatial dimension due to time variations. *Temporal Discretization* focuses on handling these transient terms. The equations for transient systems undergo both spatial and temporal discretization. Spatial discretization mirrors steady-state cases, while temporal discretization introduces time as a coordinate.

The transient behavior of a variable, ϕ, is represented by:

$$\frac{\partial \phi}{\partial t} + L(\phi) = 0 \quad (1)$$

$L(\phi)$ includes non-transient terms while $\frac{\partial \phi}{\partial t}$ is the transient operator. Integrating over an element results:

$$\int_{\Omega_e} \frac{\partial(\phi)}{\partial t} dV + \int_{\Omega_e} L(\phi) dV = 0 \quad (2)$$

Applying spatial discretization about the volume centroid leads to:

$$L(\phi_e^t) = a_e \phi_e^t + \sum_{f=1}^{N_f} a_{ef} \phi_{ef}^t + b \quad (3)$$

Overall, equation 3 denotes the discretized form of the transient behavior in the spatial domain. Explicit methods directly use this equation to predict the future state, while implicit methods solve equations involving future states.

2.1 CFL Condition

Before proceeding with the methods, the most important phenomena of time integration methods or transient solutions in general, the CFL condition, named after Richard Courant, Kurt Friedrichs, and Hans Lewy must be understood. The finite volume method discretizes partial differential equations to be solved on a boundary to observe the field variable. Since the observation is rather a piece of information that is being transferred throughout the domain, the system parameter time introduces a simple equation from physics, which is presented in Equation 2, that creates the phenomena, CFL condition.

$$time = \frac{Distance}{Velocity} \tag{2}$$

This is a simple equation that results in the required time to cover a distance for a given velocity. The interpretation of this equation in terms of finite volume method and time integration methods is that the time step of the transient solution cannot be higher than the required time for information to create a solution. In other words, selecting high-time step sizes may create problems where the solution for an element can be skipped, affecting the results vastly. The CFL condition is presented in Equation 3 where c is the maximum wave speed in the system and C is a stability constant.

$$\text{CFL} = \frac{c\Delta t}{\Delta x} \leq C \tag{3}$$

2.2 Explicit Methods

Explicit methods are a branch of time integration methods where the approach is the predict a solution at a time using history points depending on the order of the method used. The accuracy of these methods depends immensely on the time step size and might be computationally expensive when the time step sizes are small compared to Implicit Methods.

2.2.1 Forward Euler

The Forward Euler method approximates $\phi(t + \Delta t)$ with Taylor expansion around time t as:

$$\phi(t + \Delta t) = \phi(t) + \frac{\partial \phi(t)}{\partial t}\Delta t + O(\Delta t^2)$$

The first derivative approximation:

$$\frac{\partial \phi(t)}{\partial t} \approx \frac{\phi(t+\Delta t) - \phi(t)}{\Delta t} + O(\Delta t)$$

Using the relation in Eq. 2 and by defining the *rhsQ* function, which evaluates $\langle -\frac{1}{V_e} L(\phi_e^t) \rangle$, leads to:

$$\phi_e^{t+\Delta t} = \phi_e^t + (\Delta t) \cdot \text{rhsQ}(\phi_e^t)$$

In order to find the next time-level solution while considering a parabolic equation, $\frac{\partial \phi}{\partial t} + \nabla \cdot (k \nabla \phi) = 0$ The linear operator $L(\phi)$ for the element:

$$L(\phi) = \int_{\Omega_e} \nabla \cdot (k \nabla \phi)\, dV$$

Then using the divergence theorem, central fluxes, and considering Dirichlet and Neumann boundary conditions with FVM discretization $L(\phi_e)$ is approximated as:

$$L(\phi_e^t) \approx a_e \phi_e^t + \sum_{f=1}^{N_f} a_{ef} \phi_{ef}^t + b_e$$

2.2.2 Runge Kutta

The Runge-Kutta method efficiently solves initial value problems without requiring high-order derivatives.
The first-order Runge-Kutta method (explicit Euler) is achieved by:

$$\phi_e^{t+\Delta t} = \phi_e^t + (\Delta t) \cdot \text{rhsQ}(\phi_e^t)$$

For second-order Runge-Kutta methods, which involves evaluating the function at intermediate locations:

1. $k_1 = \text{rhsQ}(\phi_e^t)$ (Evaluate function at t)
2. $\phi_e^{t+\alpha\Delta t} = \phi_e^t + \alpha \Delta t \cdot k_1$ (Find field variable at some intermediate location)
3. $k_2 = \text{rhsQ}(\phi_e^{t+\alpha\Delta t})$ (Evaluate function at $t + \alpha\Delta t$)
4. $\phi_e^{t+\Delta t} = \phi_e^t + (\Delta t) \cdot (b_1 \cdot k_1 + b_2 \cdot k_2)$ (Use weighted average of intermediate slopes)

As one can observe for the different values of α (step size ratio), we get different versions of the second-order Runge-Kutta method. Which shows they are not unique.

2.2.3 Adams Bashfort

Adams Bashfort is a multi-step technique and it is designed to use and preserve knowledge from previous stages rather than discarding it. As a result, these techniques consider many preceding points and their derivative values.

$$\begin{aligned}
\text{First order:} \quad & \phi_e^{t+\Delta t} = \phi_e^t + (\Delta t) \cdot \text{rhsQ}(\phi_e^t) \\
\text{Second order:} \quad & \phi_e^{t+\Delta t} = \phi_e^t + \Delta t \left(\frac{3}{2} \cdot \text{rhsQ}(\phi_e^t) - \frac{1}{2} \cdot \text{rhsQ}(\phi_e^{t-\Delta t}) \right) \\
\text{Third order:} \quad & \phi_e^{t+\Delta t} = \phi_e^t + \Delta t \left(\frac{23}{12} \cdot \text{rhsQ}(\phi_e^t) - \frac{16}{12} \cdot \text{rhsQ}(\phi_e^{t-\Delta t}) + \frac{5}{12} \cdot \text{rhsQ}(\phi_e^{t-2\Delta t}) \right)
\end{aligned}$$

2.3 Implicit Methods

Implicit methods use sparse matrixes to solve the system of linear equations at every time step to create a solution. Implicit methods are generally more stable however the computational cost depends on the case. The main difference of explicit and implicit methods will be discussed using the results of the tests.

2.3.1 Backwards Euler

The Backwards Euler method approximates $\phi(t)$ with Taylor expansion at time t as:

$$\phi(t) = \phi(t + \Delta t) - \frac{\partial \phi(t + \Delta t)}{\partial t} \Delta t$$

The first derivative approximation:

$$\frac{\partial \phi(t + \Delta t)}{\partial t} \approx \frac{\phi(t + \Delta t) - \phi(t)}{\Delta t}$$

Using the relation in Eq. 2 leads to:

$$V_e \phi_e^{t+\Delta t} + \Delta t \cdot L(\phi_e^{t+\Delta t}) = V_e \phi_e^t$$

Equivalently:

$$A' \phi_e^{t+\Delta t} = b',$$

Where:

$$A' = \Delta t A + V \qquad \text{and} \qquad b' = -(\Delta t) b + V$$

2.3.2 Backwards Differencing

Similar to Adams Bashfort, Backwards Differencing Formula (BDF) is a multi-step technique and it is designed to use and preserve knowledge from previous stages rather than discarding it. As a result, these techniques consider many preceding points and their derivative values. The first-order BDF is written as:

$$V_e \cdot \left(\phi_e^{t+\Delta t} - \phi_e^t\right) = -\Delta t \cdot L\phi_e^{t+\Delta t}$$

This method is equivalent to the implicit Euler method. The second-order method can be derived as:

$$V_e \cdot \left(\phi_e^{t+\Delta t} - \frac{4}{3}\phi_e^t + \frac{1}{3}\phi_e^{t-\Delta t}\right) = -\Delta t \cdot L\phi_e^{t+\Delta t}$$

And similarly, the third-order method can be expressed as:

$$V_e \cdot \left(\phi_e^{t+\Delta t} - \frac{18}{11}\phi_e^t + \frac{9}{11}\phi_e^{t-\Delta t} - \frac{2}{11}\phi_e^{t-2\Delta t}\right) = -\Delta t \cdot L\phi_e^{t+\Delta t}$$

2.4 Comparison of Methods

Numerical methods for simulating transient behavior in the spatial domain can be broadly categorized as explicit and implicit schemes.

Explicit Methods

- **Forward Euler:**
  - **Strengths:** Simplicity in implementation, computationally less expensive.
  - **Limitations:** Prone to stability issues, requires very small time steps due to the CFL condition.
- **Runge-Kutta:**
  - **Strengths:** Higher accuracy compared to Forward Euler, suitable for various types of differential equations.
  - **Limitations:** Still subject to stability constraints, increased computational cost compared to Forward Euler.
- **Adams-Bashforth:**
  - **Strengths:** May provide more accurate results than Runge Kutta and Forward Euler methods since it uses multiple history points at the order increases.

  - **Limitations:** Stability is again a major problem especially as the order increases.

Implicit Methods

- **Backwards Euler:**
  - **Strengths:** Offers stability advantages, allowing larger time steps compared to explicit methods.
  - **Limitations:** Requires solving systems of equations, high computational complexity.
- **Backward Differentiation Formulas (BDF):**
  - **Strengths:** Improved accuracy because of solutions of higher order.
  - **Limitations:** Has a narrower stability region than Backwards Euler.

Overall, both the strengths and limitations of the time integration methods are similar under their branch of approach (Explicit/Implicit). Explicit methods suffer from stability while being simple whereas Implicit methods are complex and are more stable at large time steps. The preference depends immensely on the case.

3 Implementation Details

In this section, we discuss the particular implementation details of the transient integration methods used to solve parabolic partial differential equations. The Adams-Bashforth and BDF methods implementation are examined.

3.1 Implementation of Adams-Bashforth Method

3.1.1 setAdamsBashforth

```
1    def setAdamsBashforth(self,args):
2        self.coeff = {1: [1.0], 2:[1.5 , -0.5], 3:[23.0/12.0, -16.0/12.0,
         ↪  5.0/12.0]}
3        self.Nsteps = int(np.ceil((args.tend - args.tstart)/args.dt));
4        self.dt = (args.tend - args.tstart)/self.Nsteps
5        self.time = args.tstart
6        self.step = 0
7        self.cf1=self.coeff[1]
8        self.cf2=self.coeff[2]
9        self.cf3=self.coeff[3]
```

Purpose: The `setAdamsBashforth` function initializes parameters required for the Adams-Bashforth method used in solving the diffusion equation.
Algorithm
Determining the Time Steps:

1. Calculate the total number of time steps (`Nsteps`) required for the simulation based on the provided start and end times and the time step size (`dt`).
2. Set the simulation start time (`time`) to the provided start time (`args.tstart`).
3. Initialize the simulation step count (`step`) to zero.

Coefficients for Adams-Bashforth:

- Define coefficients for different orders of the Adams-Bashforth method.
- For instance, for the first order, use a coefficient array `cf1` with a single element.

It mainly sets up variables and coefficients for the Adams-Bashforth time integration method.

3.1.2 runAdamsBashforth

```
    def runAdamsBashforth(self, Qe, Qb):
        self.report(Qe)
        #-----------------------------------------------------#
        #Initialize time-step loop
        for step in range(self.Nsteps):
            if step==0:
                #Assign Q at t=0
                prevQe = Qe-0
                #Find Q according to First Order Adams Bashfort @time =
                ↪  t+dt where t=0
                Qe += self.dt*self.solver.rhsQ(Qe, Qb)
            else:
                #Apply Second Order Adams-Bashfort without updating Qe
                postQe = ((self.dt)*(self.cf2[0]*self.solver.rhsQ(Qe, Qb)
                ↪  +   self.cf2[1]*self.solver.rhsQ(prevQe, Qb)))
                #Assign Q @time = t to Q @time = t-dt
                prevQe = Qe-0
                #Update Q for time=t+dt
                Qe += postQe
            #Update global step
            self.step = step
```

```
20              #produce an output file for postprocessing
21              if step % self.Noutput==0 and step != 0:
22                  self.report(Qe)
23              #Update time
24              self.time += self.dt
```

Purpose: The `runAdamsBashforth` function executes the Adams-Bashforth time integration method to solve the diffusion equation.

Algorithm:

1. Initialize the simulation and set the number of time steps, time, and step count based on the input parameters.
2. For each time step:
   (a) For the first time step:
   - Evaluate the function using the provided initial conditions to compute Q_e at time $t + \Delta t$.
   - Since the second-order Adams-Bashfort method is implemented, the method itself is not self-starting and requires the first-order Adams-Bashford method to initialize the solution. The first-order implementation is presented under the if statement on 4.

   (b) For subsequent time steps:
   - Utilize the second-order Adams-Bashforth method to predict Q_e at time $t+\Delta t$ based on the previous steps' information.
   - Update the current time step and compute the new Q_e at $t + \Delta t$ using the predicted values.
   - The second-order implementation is presented under the else statement on line 7.

   (c) Store the results, update the time, and continue to the next time step.
3. Generate output files at specified intervals for post-processing.

Overall, it mainly updates the field variables (`Qe`) using the Adams-Bashforth time integration method.

3.2 Implementation of BDF Method

3.2.1 setBDF

```
    def setBDF(self,args):
        self.coeff = {1: [1.0], 2:[4.0/3.0 , -1.0/3.0], 3:[18/11.0,
        ↪  -9.0/11.0, 2.0/11.0]}N
        self.Nsteps = int(np.ceil((args.tend - args.tstart)/args.dt));
        self.dt = (args.tend - args.tstart)/self.Nsteps
        self.time = args.tstart
        self.step = 0
        self.cf1=self.coeff[1]
        self.cf2=self.coeff[2]
        self.cf3=self.coeff[3]
```

Purpose: The `setBDF` function initializes parameters required for the Backward Differentiation Formula (BDF) time integration method used in solving the diffusion equation.
Algorithm
Determining Time Steps:

1. Calculate the total number of time steps (`Nsteps`) required for the simulation based on the provided start and end times and the time step size (`dt`).
2. Set the simulation start time (`time`) to the provided start time (`args.tstart`).
3. Initialize the simulation step count (`step`) to zero.

Coefficients for BDF:

- Define coefficients for different orders of the Backward Differentiation Formula method
- For instance, for the first-order implementation use `cf1`.

It primarily sets up variables and coefficients for the BDF time integration method.

3.2.2 runBDF

```
    def runBDF(self, Qe, Qb):
        #-------------------------------------------------------#
        self.report(Qe)
        #-------------------------------------------------------#
        # COMPLETE THIS FUNCTION
        # First form the sytem matrices for integration
```

```
        A, b  = self.solver.assemble(Qe, Qb)

        #-------------------------------------------------------#
        vals = np.zeros(self.mesh.Nelements, float);
        vol = vals.reshape((self.mesh.Nelements,1))
        for elm, info in self.mesh.Element.items():
            vol[elm] = info['volume']

        Ap = sp.sparse.spdiags(np.transpose(vol),0, m=self.mesh.Nelements,
        ↪  n=self.mesh.Nelements,
         format=A.getformat())
        #-------------------------------------------------------#
        # (qn1 - qn0)*V/ dt = L(q) = [A(qn1) + b]
        # [qn1*V - dt*A(qn1)] = qn0*V + dt*b
        # Ap (qn1) = rhs
        Ap = Ap - self.dt*A
        rhs = self.dt*b  + np.multiply(Qe, vol)
        #-------------------------------------------------------#
        for step in range(self.Nsteps):
            #Apply first order BDF at step=0
            if (step==0):
                #Store initial Qe for the first step
                prevQe=Qe-0
                Qe[:] = self.solver.solve(self.solver.args, Ap, rhs);
            #Apply Second Order BDF at step>0
            else:
                #Compute b' using BDF coefficients for second order.
                rhs = self.dt*b  + self.cf2[0]*np.multiply(Qe, vol) +
                 ↪  self.cf2[1]*np.multiply(prevQe,  vol)
                #Store Qe value at t=t-dt
                prevQe=Qe-0
                #Solve Qe for t=t+dt
                Qe[:] = self.solver.solve(self.solver.args, Ap, rhs);
                #Update global step
            self.step = step
            if step % self.Noutput == 0:
                self.report(Qe)
                #Update time
            self.time += self.dt
```

Purpose: The `runBDF` function executes the Backward Differentiation Formula (BDF) time integration method for solving the diffusion equation.
Algorithm/Implementation:

1. Initialize the simulation and set the number of time steps, time, and step count based on the input parameters.
2. For each time step:
   (a) For the first time step:
      - Solve the Backward Differentiation Formula (BDF) to compute Q_e at time $t + \Delta t$.
      - Just like the Adams Bashforth method, this function also requires two history points as it is second order. Consequently, the implementation on the first time step is a Backward Euler method application. The details are presented under the if statement on line 13.

   (b) For subsequent time steps:
      - Utilize the BDF method to iteratively update Q_e based on the previous steps' information.
      - Compute the new Q_e at $t + \Delta t$ using the BDF formula.
      - The second-order implementation details are presented under the else statement on line 16.

   (c) Store the results, update the time, and proceed to the next time step.
3. Generate output files at specified intervals for post-processing.

4 Discussion and Results

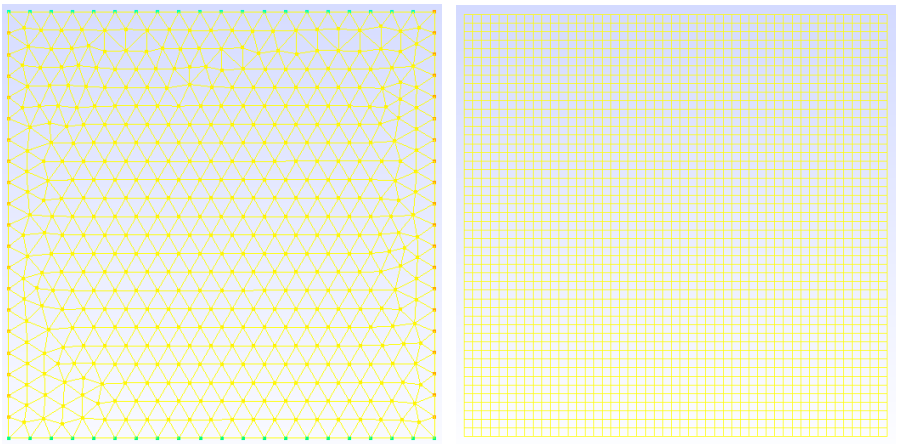
4.1 Details and Results of the Cases

4.1.1 Case 1

In this case, the results of two structured meshes are compared. The details of the meshes are presented in Figure **??**. The types of meshes are triangular and quadratic where the number of elements in the mesh are 946 and 2401 respectively. The settings for the initial condition, boundary condition, diffusion coefficient, diffusion source, start time and end time are presented in Table 1. The results of the first case for different time step sizes and the aforementioned meshes are presented on Tables 2, 3, 4, 5, 6, 7.

Table 1: Default Settings

Initial Condition	0
Boundary Condition	$\sin(\pi x)\sin(\pi y)$
Diffusion Coefficient	1
Diffusion Source	$2\pi^2 \sin(\pi x)\sin(\pi y)$
Start Time	0
End Time	1



((a)) Cavity Triangular Mesh ((b)) Cavity Quadratic Mesh

Figure 1: Pictures of the meshes used for case 1.

Table 2: Results of Cavity Triangular Mesh for dt = 0.01 .

Parameters	Forward Euler	Adams Bashfort	LSRK4	Backward Euler	BDF
Infinity Norm of Error	3.5131×10^{141}	6.41593×10^{159}	0.0000×10^{0}	3.33637×10^{-2}	3.33677×10^{-2}
L2 Norm of Error	1.2661×10^{281}	∞	0.0000×10^{0}	1.43492×10^{-4}	1.43615×10^{-4}

Table 3: Results of Cavity Quadratic Mesh for dt = 0.01 .

Parameters	Forward Euler	Adams Bashfort	LSRK4	Backward Euler	BDF
Infinity Norm of Error	$6.64392 \times 10^{+157}$	$5.95285 \times 10^{+175}$	0.0000×10^{0}	2.62341×10^{0}	2.87621×10^{0}
L2 Norm of Error	∞	∞	∞	2.21535×10^{0}	2.87396×10^{0}

Table 4: Results of Cavity Triangular Mesh for dt = 0.001 .

Parameters	Forward Euler	Adams Bashfort	LSRK4	Backward Euler	BDF
Infinity Norm of Error	9.78382×10^{291}	0.00×10^{0}	3.33642×10^{-2}	3.33642×10^{-2}	3.33677×10^{-2}
L2 Norm of Error	∞	∞	1.43507×10^{-4}	1.43505×10^{-4}	1.43615×10^{-4}

Table 5: Results of Cavity Quadratic Mesh for dt = 0.001 .

Parameters	Forward Euler	Adams Bashfort	LSRK4	Backward Euler	BDF
Infinity Norm of Error	0.0000×10^{0}	0.0000×10^{0}	$9.7458 \times 10^{+113}$	2.62769×10^{0}	2.87865×10^{0}
L2 Norm of Error	∞	∞	$9.5117 \times 10^{+227}$	2.22474×10^{0}	2.88097×10^{0}

Table 6: Results of Cavity Triangular Mesh for dt = 0.0001 .

Parameters	Forward Euler	Adams Bashfort	LSRK4	Backward Euler	BDF
Infinity Norm of Error	3.33642×10^{-2}	3.33642×10^{-2}	3.33642×10^{-2}	3.33642×10^{-2}	3.33677×10^{-2}
L2 Norm of Error	1.43507×10^{-4}	1.43507×10^{-4}	1.43507×10^{-4}	1.43507×10^{-4}	1.43615×10^{-4}

Table 7: Results of Cavity Quadratic Mesh for dt = 0.0001 .

Parameters	Forward Euler	Adams Bashfort	LSRK4	Backward Euler	BDF
Infinity Norm of Error	2.62821×10^{0}	2.62816×10^{0}	2.62816×10^{0}	2.62812×10^{0}	2.87889×10^{0}
L2 Norm of Error	2.22590×10^{0}	2.22579×10^{0}	2.22579×10^{0}	2.22569×10^{0}	2.88167×10^{0}

Observing the results, the main takeaways are as follows:

- The Tables 2 and 3 show us that explicit methods have divergent results. As the time step size decreases, the Runge-Kutta method is the first method to converge to a solution and the rest converges at the smallest time step size, dt=0.0001 as presented in Tables 6 and 7.
- The results of implicit methods are not affected by the time step size for this particular case. The results do not show any fluctuations for the Cavity Triangular Mesh and fluctuations of 0.0083% for the BDF method and 0.016% for the Backward Euler method which are negligible.
- Although the runtimes are not presented, the implicit methods were much faster to compute than the explicit methods where the Runge-Kutta method was the slowest among the explicit methods because it had five stages.

- Before testing, it was expected that the results of the Quadratic Mesh would have lower errors than the Triangular Mesh because of the number of elements and the Triangular mesh is not perfectly orthogonal at the boundaries as seen in Figure 1. However, the results show that the errors of triangular mesh are far smaller than the quadratic mesh.

The testing continues with case 2.

4.1.2 Case 2

From the results of case 1, a new question arose. Since the geometries of the meshes were the same but the resolution of the quadratic mesh was two times higher than the resolution of the triangular mesh, another mesh if formed that was equal to the resolution of the quadratic mesh, but with triangular elements, resulting in a total of 3712 elements. Keeping the other setting the same, the testing continues for this new finer triangular mesh. The results are presented in Table 8.

Table 8: Results of The New Mesh for dt = 0.01 .

Parameters	Forward Euler	Adams Bashfort	LSRK4	Backward Euler	BDF
Infinity Norm of Error	3.48996×10^{209}	1.38945×10^{227}	0.0000×10^{0}	2.62832×10^{0}	2.88127×10^{0}
L2 Norm of Error	∞	∞	∞	2.22244×10^{0}	2.88122×10^{0}

Comparing the results of Table 3 and 8, it is evident that the difference in the results of the triangular and quadratic meshes was a problem of resolution. The new testing shows that the results of the implicit methods are very similar. Thus, no further testing on smaller time step sizes was done. Usually, increase in the resolution provides better results, however in this case it did not. At really high resolutions, the solvers might not converge thus leading to inaccurate results. This might have been the case here. The testing continues with case 3.

4.1.3 Case 3

In this case, an unstructured mesh with a mix of triangular and linear elements is tested. This mesh has a total of 200 elements.The picture of the mesh is presented in Figure 2. From the previous cases we have seen that explicit solvers converge for dt=0.01 .Thus, solutions will be provided only for this time step size. The results of the testing is presented on Table 9.

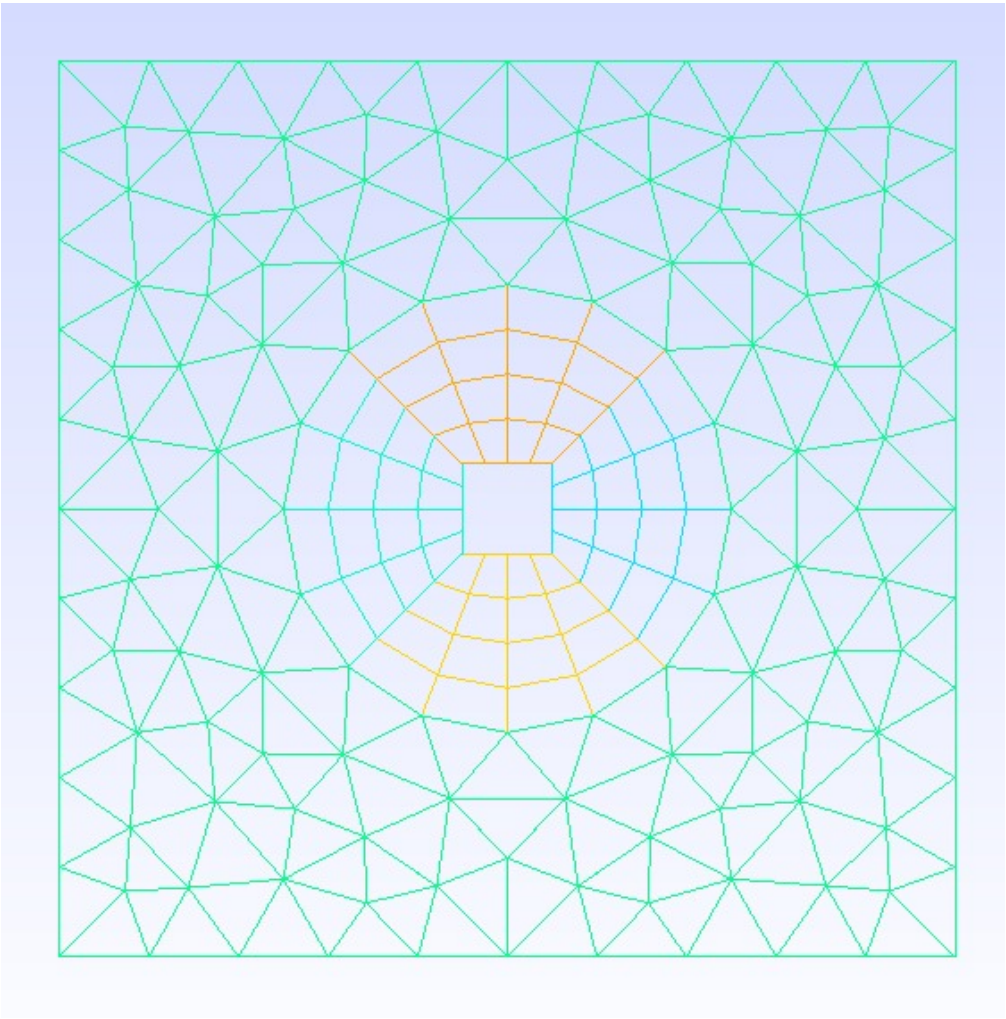


Figure 2: Mixed Mesh.

Table 9: Results of The New Mesh for dt = 0.01 .

Parameters	Forward Euler	Adams Bashfort	LSRK4	Backward Euler	BDF
Infinity Norm of Error	6.74515×10^{0}	9.896470×10^{-1}	6.74500×10^{0}	6.74961×10^{0}	8.5736×10^{0}
L2 Norm of Error	2.79976×10^{3}	9.24155×10^{1}	2.79963×10^{3}	2.84065×10^{3}	4.3708×10^{3}

The results show that all integration methods except Adams-Bashfort converged to similar results while the Adams-Bashfort method has significantly smaller norms of errors. However, the L2 norm of error results are very large for all methods. This problem might arise due to a mistake in the orthogonality correction implementation for the assembly of the linear system of diffusion equation since the mesh is unstructured.

5 Conclusion

In this report, implementation details of two time integration methods, which are the explicit method Adams-Bashford and the implicit method Backward-Differencing, and results of different cases that include testing of time integration methods were presented. These case groups included meshes of the same geometry, different resolutions, and different element types, meshes of the same geometry and resolution but different element types, and a different geometry with mixed unstructured element types. From the results of these case groups it has been observed that for the case of the parabolic diffusion problem, explicit methods suffered from large time step sizes, and the results varied between different resolutions for the same geometry. Moreover, the last case showed that for an unstructured mixed mesh, the difference in results between methods for a feasible time step size increased, leading the second-order Adams-Bashfort method the best option for the unstructured case. This leads to a conclusion point where for simpler meshes, first order methods yield faster and accurate results where for complex meshes, second order methods yield more accurate results. In addition to that, for a simple mesh, Forward Euler and Backward Euler methods can be used whereas for a complex mesh, the Adams-Bashfort method proved to be the best. The last remark would be that although the Runge-Kutta method is significantly slower compared to the other explicit methods, it converges to an accurate solution at higher time step sizes, leading to a conclusion that for a large, simple system where solution times of implicit methods would be large and second order methods would not be necessary, the Runge-Kutta method can be a viable option of time integration.

6 References

[1] Lecture notes of Ali Karakuş for ME485.