NORWEGIAN UNIVERSITY OF LIFE SCIENCES
(NMBU)

REPORT: SIMULATION OF AN OIL SPILL

AUTHORS

MAKKA DULGAEVA
MILAD TOOTOONCHI
SANIA MINAIPOUR

NORWAY, JANUARY 2025

Contents

1 Introduction

1.1 Problem statement

In this report we will present the development of a Python-based simulation designed to model the movement of oil spill in a coastal region. Oil spills are a serious threat to marine ecosystems, coastal communities, and the environment. Developing a simulation enables a better understanding of how oil spreads and provide strategies to reduce its damage.

"This project aims to develop software that predicts how oil spreads over time and measures its movement per time step. By simulating an oil spill, the project aims to generate valuable insights that can help reduce environmental damage and support decision-making during cleanup operations

This report is organized into several sections. The "Introduction" outlines the background and motivation behind the project. The "Methods" section explains the development process, covering the algorithms and pseudo-code. The "Results and discussion" section presents the findings from the simulation, explains and analyzes the results, and discusses the strength and limitations of both the code and the model. The conclusion summarizes the key findings, while the "Appendix" includes formulas.

2

Methods and Implementation

2.1 Development Approach and Agile Methods

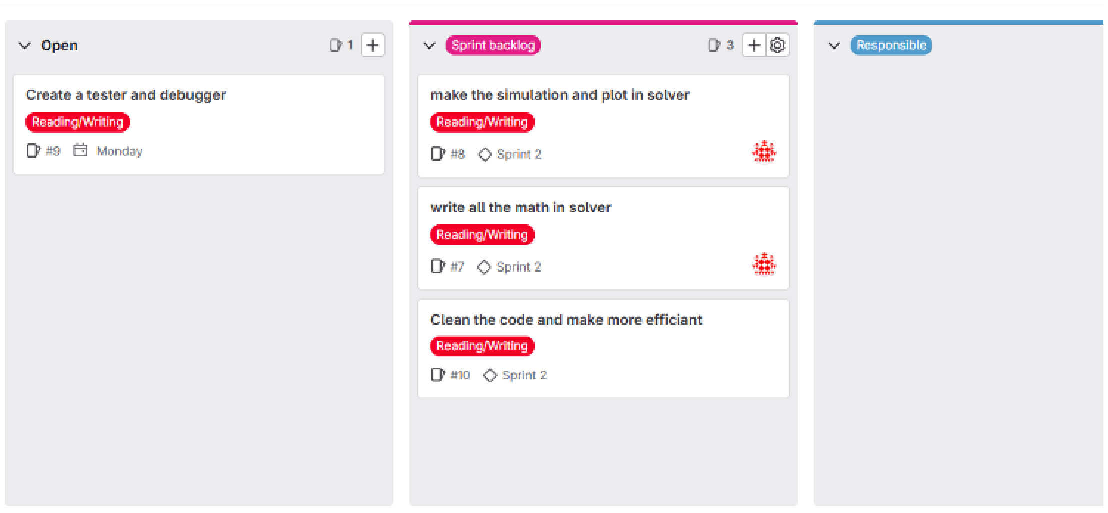
To develop the oil spill simulation, we followed an agile development method. This iterative approach allowed us to work step by step, make continuous improvements, and adjustments through the development. By dividing the project into smaller manageable tasks, we were able to efficiently address challenges and maintain consistent progress.

We organized the work process using Git board, which helped us track tasks and ensure that all project requirements were met. The git board was organized into key sections such as "Open," "Sprint Backlog," "Responsible," and "Closed," helping us prioritize tasks and distribute the tasks among the group members. This structured approach allowed us to maintain an organized the workflow, ensuring that we covered all the requirements of the project, making sure nothing was overlooked.

We also implemented a step-by-step plan to focus on specific components before moving forward. Initially, we focused on planning and understanding the project requirements, then developed the main algorithms, and later integrated features like parameter testing, function validation, and result visualization.

To organize the software development we paid close attention to the guidance provided during lectures. Our goal was to make the code as efficient as possible by avoiding unnecessary operations that could reduce the performance. At the same time we focused on including all the essential features to achieve the result we wanted.

Figure 2.1 and 2.2 shows our Git board, which shows how we planned and organized the project.


Open 1

Create a tester and debugger
Reading/Writing
#9 Monday

Sprint backlog 3

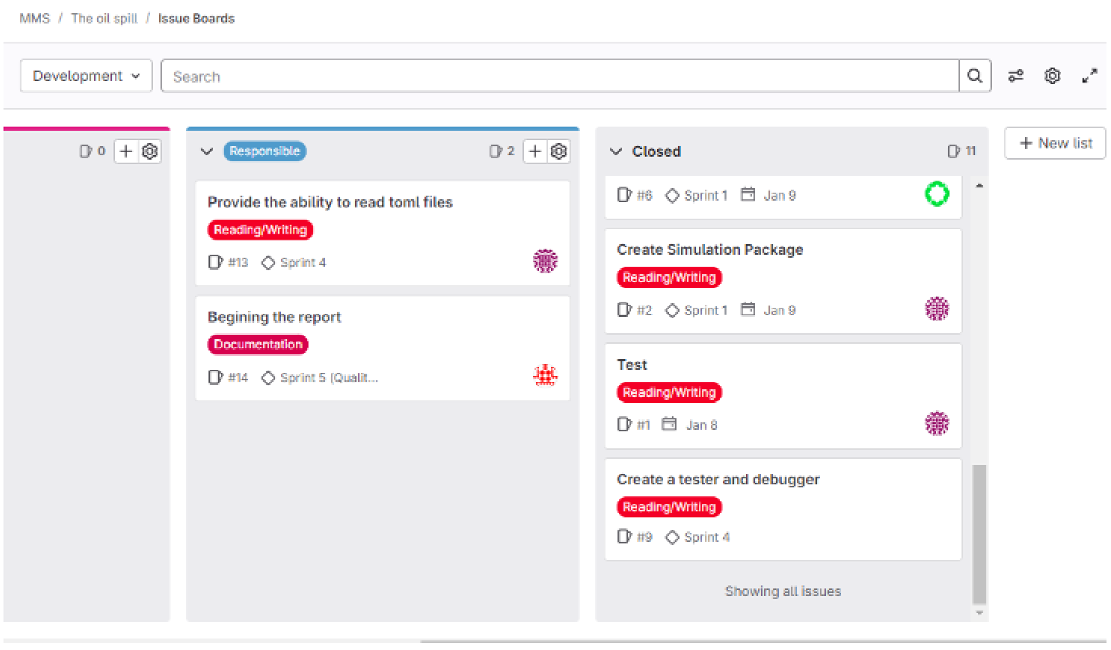
make the simulation and plot in solver
Reading/Writing
#8 Sprint 2

write all the math in solver
Reading/Writing
#7 Sprint 2

Clean the code and make more efficient
Reading/Writing
#10 Sprint 2

Responsible

Figure 2.1: *Git board start*


MMS / The oil spill / Issue Boards

Development Search

0

Responsible 2

Provide the ability to read toml files
Reading/Writing
#13 Sprint 4

Begining the report
Documentation
#14 Sprint 5 (Qualit...

Closed 11

#6 Sprint 1 Jan 9

Create Simulation Package
Reading/Writing
#2 Sprint 1 Jan 9

Test
Reading/Writing
#1 Jan 8

Create a tester and debugger
Reading/Writing
#9 Sprint 4

Showing all issues

+ New list

Figure 2.2: *Git board end*

2.2 User Guide

2.2.1 Running the Simulation

To run the simulation successfully, the following steps and materials must be followed:

1. **Required Files:** The simulation requires the placement of a TOML configuration file and a mesh file in the program's main directory. The TOML file must include the following three sections:

- **Settings:**
  - **nSteps:** Defines the number of simulation steps.
  - **tEnd:** Specifies the end time for the simulation.
- **Geometry:**
  - **meshName:** Indicates the mesh file used for simulation.
  - **borders:** Defines the boundary conditions of the simulation area.
- **IO (Input/Output):**
  - **logName:** (Optional) Specifies the name of the program's log file.
  - **writeFrequency:** (Optional) Defines how often output data is saved to create a simulation video.
  - If you have run the simulation before and obtained a TXT file containing solution values for the oil distribution at a specific time, you can set **tStart** to that time in the settings section and provide the TXT file's path in the IO section of the TOML file.
  - Note: If a restart file is provided, a start time must also be specified, and vice versa.

2.2.2 Execution Steps

Once the necessary files are in place, the following steps should be followed to execute the simulation:

1. Open the terminal and navigate to the main folder.
2. Type python main.py and include:
   - `-c` or `-config_file "name_of_toml_file"` to run the simulation for a specific TOML file.
   - `-find_all` to run the simulation for all TOML files in the directory.
   - include -f "folder_name" or `-folder "folder_name"` before the -c or –find_all if the file(s) are located in a subfolder.

2.2.3 Output Files

After successful execution, the program generates a folder named after the TOML file. This folder contains the following files:

- **TXT file:** Contains the final oil distribution solution at t_{End}.
- **Log file:** Documents the entire simulation process.
- **Image file:** Represents the oil distribution at t_{End}.
- **Optional video file:** Visualizes the entire simulation process per time.

2.3 Code structure

The structure of our program is organized to ensure readability and functionality. Within the "src" folder, we have a dedicated directory for the simulation package, which serves as the simulation of the program. Additionally, there is a separate folder for configurations that does not include the default configuration named "input.toml". The default configuration has a folder named after itself "input", which contains its corresponding solution. There exists more folder in the directory for Python tests and the images required for video generation. The main directory serves as the main folder, containing essential components such as the main Python script, the configuration python file, the Git log, the library requirements, the input configuration and the mesh file.

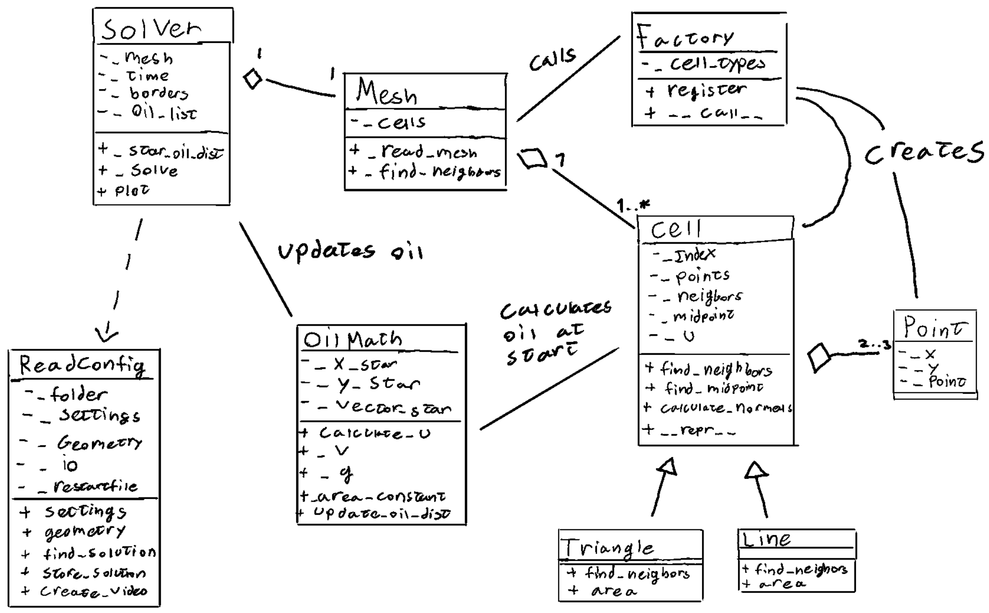


Figure 2.3: *Class map*

Figure 2.3 shows the class structure. The Simulation package includes the cell-, mesh-, solver- and the oilmath python file. The cell file contains the Cell class that serves as the parent class of the Triangle and Line classes which you also can find in the cells file. Another class you can find in the same file is the Point class that only describes the points in cells as objects with x and y properties. The CellFactory class is also in the cell python file. The Factory's task is to create new cells that contains points. Moving on to the mesh file. The python file contains a mesh class that has the main task of reading the mesh and calling on the CellFactory. It also makes all cells call on its find_neighbor method by storing all cells in a list.

1cm The oilmath file has a class with the same name that only consists of functions that evaluate start oil and updated oil using the formulas in chapter 6.

The Solver class lies in the solver file and is the solver for the simulation. It has properties like time which helps the simulation have a track of which method to call on from oilmath. The Solver is the glue between the mesh and the oil calculations. It also has a plot method that makes images of the plot. The whole Simulation package is dependent on the parameters given by the configuration files. The main python files make sure that everything between the Simulation and the configurations goes smoothly. The main python file keeps updating the log at the same time.

2.4 Pseudocode for the Oil Spill Simulation

2.4.1 Point and Cell Classes (cells.py)

Algorithm 1 Point Class

```
1: Attributes:
2:     $x, y$                                         ▷ Coordinates
3:     point ← $(x, y)$
4: Methods:
5:     Get x
6:     Get y
7:     Get point
```

Algorithm 2 Cell Class (Abstract)

```
Attributes:
    index                                          ▷ Cell index
    points                                         ▷ List of Point objects
    neighbors                                      ▷ List of neighboring cell indices
    midpoint ← find_midpoint()
    $u$ ← calculate_u(midpoint)                    ▷ Calls from OilMath class
Methods:
    Get index, points, neighbors, midpoint, $u$
    Set $u$
    Abstract: find_neighbors(cells)
    Function: find_midpoint()
        Calculate average of all point coordinates
        Return midpoint
    Function: calculate_normalscells
        for each neighbor_index in neighbors
            if $0 \leq$ neighbor_index < len(cells)
                Find shared points
                Calculate scaled normal vector
                Adjust normal direction if necessary
                Append scaled normal to list
            return list of normals
```

2.4.2 CellFactory Class

Algorithm 3 CellFactory Class

```
Attributes:
    cell_types ← { "triangle": Triangle, "line": Line }
Methods:
    register(key, name)                            ▷ Add new cell type
    Function: __call__(msh)
        Create list of Point objects from mesh cell in mesh
        Get corresponding cell type
        Create cell objects and add to list

        Return list of cells
```

Algorithm 4 Triangle and Line Classes

```
Triangle: Find neighbors by checking two shared points, calculate area.
Line: Find neighbors by checking one shared point.
```

2.4.3 Mesh Class (mesh.py)

Algorithm 5 Mesh Class (mesh.py)

```
Attributes:
    _cells                                          ▷ List of Cell objects
Initialization:
    Input: mesh file file
    Initialize _cells as an empty list
    Call _read_mesh(file)
    Call _find_neighbors()
Methods:
    Get: cells
        return list of cells (_cells)
    Function: _read_mesh(file)
        Try:
            Read mesh using meshio.read(file)
        Catch exception e
            Raise error "Failed to read mesh file file"
        Create cells using CellFactory
        Store cells in _cells
    Function: _find_neighbors() cell in cells
        Call cell.find_neighbors(cells)

```

2.4.4 Oilmath (Oilmath.py)

Algorithm 6 OilMath Class Initialization

```
Initialize:
Set x_star ← 0.35
Set y_star ← 0.45
Set vector_star ← $[0.35, 0.45]$
```

Algorithm 7 calculate_u Function

```
Input: $x, y$ (coordinates)
Create vector ← $[x, y]$
Calculate distance_from_star ← vector − vector_star
Compute oil concentration:
    oil_concentration ← $\exp(-\|$distance_from_star$\|^2/0.01)$
Return oil_concentration
```

Algorithm 8 _v Function (Calculate Oil Velocity)

```
1: Input: $x, y$ (coordinates)
2: Return $(y - 0.2 \cdot x, -x)$
```

Algorithm 9 _g Function (Calculate Oil Flux)

```
1: Input: $u_i, u_{ngh}, normal, v_i, v_{ngh}$
2: Compute average velocity:
3:     v_avg ← 0.5 · (v_i + v_ngh)
4: Calculate $v$_dot_normal ← dot product of $v_{\text{avg}}$ and normal
5: if $v$_dot_normal > 0 then
6:     Return $u_i \cdot v$_dot_normal
7: else
8:     Return $u_{ngh} \cdot v$_dot_normal
```

Algorithm 10 _area_constant Function

```
1: Input: $area, dt$
2: if $area \leq 0$ then
3:     Raise Error: "Area is negative or zero"
4: Return $dt/area$
```

Algorithm 11 update_oil_distribution Function

```
1: Input: $cell, all\_cells, dt$
2: Set $u_{new} \leftarrow cell.u$
3: Set $cell\_midpoint \leftarrow cell.midpoint$
4: Compute normals for cell ← cell.calculate_normals(all_cells)
       neighbor and normal in neighbors and normals
5: if $0 \leq neighbor <$ len(all_cells) then
6:     if neighbor is a triangle then
7:         $u_{ngh} \leftarrow neighbor.u$
8:     else
9:         $u_{ngh} \leftarrow 0$
10:    Compute velocity at cell midpoint
11:    Compute velocity at neighbor midpoint
12:    Calculate flux:
13:        $g \leftarrow \_g(cell.u, u_{ngh}, normal, velocity, velocity_{ngh})$
14:    Update oil concentration:
15:        $u_{new} \leftarrow u_{new} - \_area\_constant(cell.area(), dt) \cdot g$
16: Return $u_{new}$
```

2.4.5 Solver (Solver.py)

Algorithm 12 Solver Class Initialization

```
 1: Input: mesh file file, borders borders, oil list oil_list, time time, logger
 2: Initialize mesh using Mesh(file)
 3: Set time ← time
 4: Set borders ← borders
 5: if time = 0.0 then
 6:     Initialize oil distribution using _start_oil_distribution()
 7: else
 8:     Set oil list ← oil_list
```

Algorithm 13 Get Time and Oil List

```
 1: Function: get time()
 2: Return time
 3: Function: get oil list()
 4: Return oil list
```

Algorithm 14 Start Oil Distribution

```
 1: Function: _start_oil_distribution()
 2: Create an empty list oil_list cell in mesh.cells
 3: Add cell oil amount $u$ to oil_list
 4:
 5: Return oil_list
```

Algorithm 15 Solve Function

```
 1: Function: solve(dt)
 2: Update time: time += dt
 3: Initialize empty lists: u_new_list, u_in_fishground_list
        cell in mesh.cells
 4: Compute new oil amount using OilMath.update_oil_distribution
 5: Ensure oil concentration is non-negative
 6: Update cell oil value
 7: Add updated value to u_new_list
 8: if cell midpoint is inside fish ground borders then
 9:     Add oil amount to u_in_fishground_list
10:
11: Update oil list: oil_list ← u_new_list
12: Return sum of oil in fish ground
```

Algorithm 16 Plot Function

```
1: Function: plot(folder = "imgs")
2: Initialize colormap using oil list
3: Compute maximum and minimum oil values
       cell, oil amount in mesh.cells, oil_list
4: Extract cell coordinates
5: Normalize oil concentration
6: Add cell polygon to plot with color based on oil concentration
7:
8: Draw fishing grounds using rectangle patch
9: Label axes and adjust aspect ratio
10: Save plot as imgs/oil_dist_{time}.png
11: if folder is not "imgs" then
12:     Save plot in specified folder
13: Close the plot
```

2.4.6 Create Logger

Algorithm 17 CreateLogger(file_name)

```
1: Create a logger for file_name
2: Set up a log file with write mode
3: Format log messages with date, level, and text
4: Add formatter to logger
5: Set logging level to INFO
6: Return logger
```

2.4.7 Run Simulation

Algorithm 18 RunSimulation(conf_path, conf)

```
 1: Load configuration from conf_path
 2: Create logger
 3: Log simulation start message
 4: Get start time, solution from config
 5: Get end time and number of steps
 6: Compute time step (dt)
 7: Get mesh file and borders from config
 8: Create Solver with mesh, borders, and initial data
 9: for each step from 1 to nSteps do
10:     Generate plot
11:     Update oil distribution
12:     Log current time and oil amount
13: Save final plot and results
14: Create simulation video
15: Log simulation completion message
```

2.4.8 Main Program

Algorithm 19 Main Program

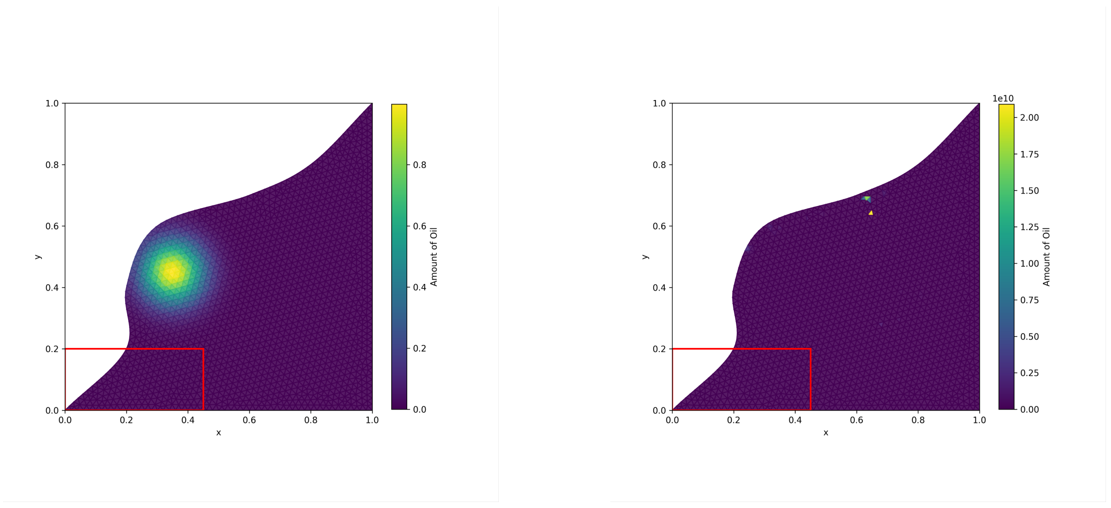
```
1: Read input arguments
2: if find all configs then
3:     Get all config files from folder
4:     for each config file do
5:         RunSimulation(config file)
6: if specific config file then
7:     RunSimulation(selected config file)
```

3

Results

By runnning the simulation of the oil spill with different configurations, we observed a significant impact on how the oil spread across the cells. We chose to run the simulation with different nSteps values. When nSteps was set to a high value, the simulation would show oil accumulating in a few cells instead of spreading more evenly across the grid.

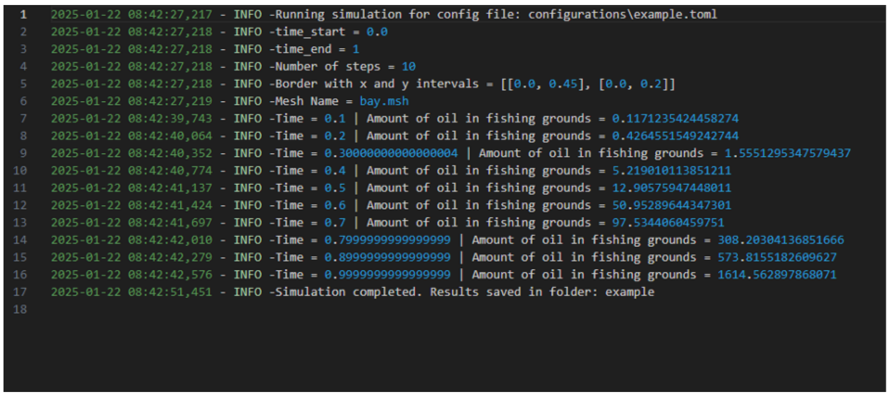
The figure 3.1 shows the results of the oil spill simulation with nSteps equal to 10, tStart will be automaticly be 0.0 since the parameter and a solution text was not given, and tEnd was set to 1.0. These figures illustrate how the oil accumulates in fewer cells compared to simulations with smaller time steps.



Simulation with low n_steps start *Simulation with low n_steps end*

Figure 3.1: *Comparison of simulation results*

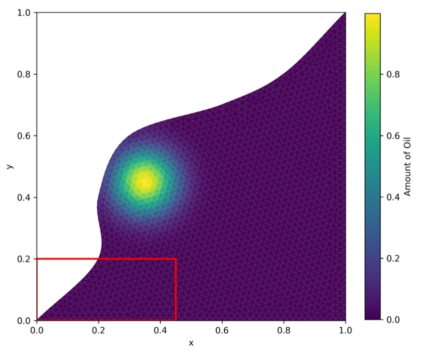
As shown in figure 3.2 , we observe that by using nSteps of 10, the amount of oil increases drastically in fishing ground.



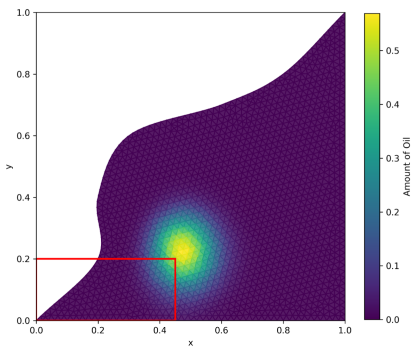
```
2025-01-22 08:42:27,217 - INFO -Running simulation for config file: configurations\example.toml
2025-01-22 08:42:27,218 - INFO -time_start = 0.0
2025-01-22 08:42:27,218 - INFO -time_end = 1
2025-01-22 08:42:27,218 - INFO -Number of steps = 10
2025-01-22 08:42:27,218 - INFO -Border with x and y intervals = [[0.0, 0.45], [0.0, 0.2]]
2025-01-22 08:42:27,219 - INFO -Mesh Name = bay.msh
2025-01-22 08:42:39,743 - INFO -Time = 0.1 | Amount of oil in fishing grounds = 0.1171235424458274
2025-01-22 08:42:40,064 - INFO -Time = 0.2 | Amount of oil in fishing grounds = 0.4264551549242744
2025-01-22 08:42:40,352 - INFO -Time = 0.30000000000000004 | Amount of oil in fishing grounds = 1.5551295347579437
2025-01-22 08:42:40,774 - INFO -Time = 0.4 | Amount of oil in fishing grounds = 5.219010113851211
2025-01-22 08:42:41,137 - INFO -Time = 0.5 | Amount of oil in fishing grounds = 12.90575947448011
2025-01-22 08:42:41,424 - INFO -Time = 0.6 | Amount of oil in fishing grounds = 50.95289644347301
2025-01-22 08:42:41,697 - INFO -Time = 0.7 | Amount of oil in fishing grounds = 97.5344060459751
2025-01-22 08:42:42,010 - INFO -Time = 0.7999999999999999 | Amount of oil in fishing grounds = 308.20304136851666
2025-01-22 08:42:42,279 - INFO -Time = 0.8999999999999999 | Amount of oil in fishing grounds = 573.8155182609627
2025-01-22 08:42:42,576 - INFO -Time = 0.9999999999999999 | Amount of oil in fishing grounds = 1614.562897868071
2025-01-22 08:42:51,451 - INFO -Simulation completed. Results saved in folder: example
```

Figure 3.2: *log with n=10*

N steps was also changed to have a higher value. The value was set to 2500, which made the simulation look more realistic. Figure 3.3 shows the simulation with nSteps 2500, tStart 0.0 and tEnd 1.0. The boarders had a x-interval of [0.0 , 0.45] , and a y-interval of [0.0 , 0.2]



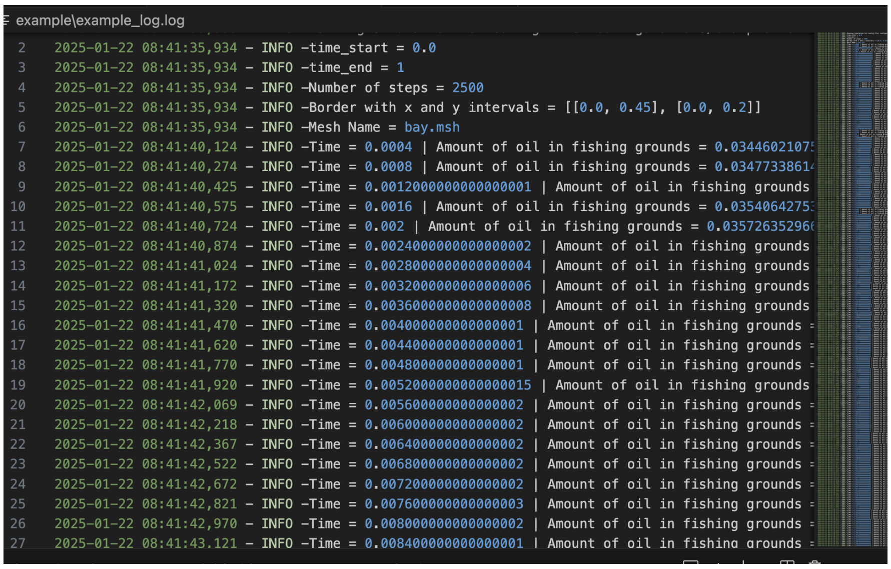
Simulation with low n_steps start



Simulation with low n_steps end

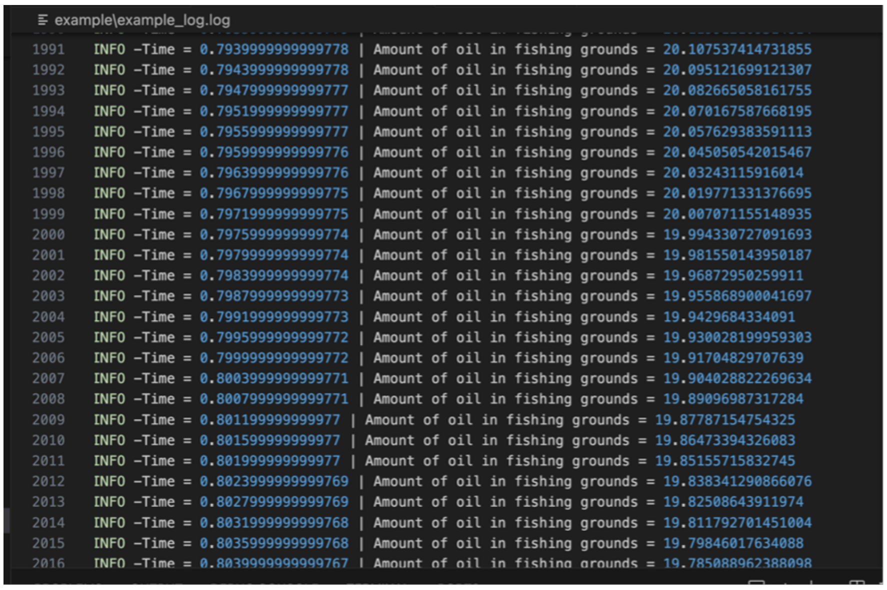
Figure 3.3: *Comparison of simulation results*

Figure 3.4 and 3.5 shows how the amount of oil changes with nSteps of 2500. The amount starts of by increasing, and then starts to decrease. The amount of oil is decreasing because the oil is moving away from the fishing ground.



```
example\example_log.log
2025-01-22 08:41:35,934 - INFO -time_start = 0.0
2025-01-22 08:41:35,934 - INFO -time_end = 1
2025-01-22 08:41:35,934 - INFO -Number of steps = 2500
2025-01-22 08:41:35,934 - INFO -Border with x and y intervals = [[0.0, 0.45], [0.0, 0.2]]
2025-01-22 08:41:35,934 - INFO -Mesh Name = bay.msh
2025-01-22 08:41:40,124 - INFO -Time = 0.0004 | Amount of oil in fishing grounds = 0.0344602107
2025-01-22 08:41:40,274 - INFO -Time = 0.0008 | Amount of oil in fishing grounds = 0.0347733861
2025-01-22 08:41:40,425 - INFO -Time = 0.0012000000000000001 | Amount of oil in fishing grounds
2025-01-22 08:41:40,575 - INFO -Time = 0.0016 | Amount of oil in fishing grounds = 0.0354064275
2025-01-22 08:41:40,724 - INFO -Time = 0.002 | Amount of oil in fishing grounds = 0.035726352966
2025-01-22 08:41:40,874 - INFO -Time = 0.0024000000000000002 | Amount of oil in fishing grounds
2025-01-22 08:41:41,024 - INFO -Time = 0.0028000000000000004 | Amount of oil in fishing grounds
2025-01-22 08:41:41,172 - INFO -Time = 0.0032000000000000006 | Amount of oil in fishing grounds
2025-01-22 08:41:41,320 - INFO -Time = 0.0036000000000000008 | Amount of oil in fishing grounds
2025-01-22 08:41:41,470 - INFO -Time = 0.004000000000000001 | Amount of oil in fishing grounds =
2025-01-22 08:41:41,620 - INFO -Time = 0.004400000000000001 | Amount of oil in fishing grounds =
2025-01-22 08:41:41,770 - INFO -Time = 0.004800000000000001 | Amount of oil in fishing grounds =
2025-01-22 08:41:41,920 - INFO -Time = 0.0052000000000000015 | Amount of oil in fishing grounds
2025-01-22 08:41:42,069 - INFO -Time = 0.005600000000000002 | Amount of oil in fishing grounds =
2025-01-22 08:41:42,218 - INFO -Time = 0.006000000000000002 | Amount of oil in fishing grounds =
2025-01-22 08:41:42,367 - INFO -Time = 0.006400000000000002 | Amount of oil in fishing grounds =
2025-01-22 08:41:42,522 - INFO -Time = 0.006800000000000002 | Amount of oil in fishing grounds =
2025-01-22 08:41:42,672 - INFO -Time = 0.007200000000000002 | Amount of oil in fishing grounds =
2025-01-22 08:41:42,821 - INFO -Time = 0.007600000000000003 | Amount of oil in fishing grounds =
2025-01-22 08:41:42,970 - INFO -Time = 0.008000000000000002 | Amount of oil in fishing grounds =
2025-01-22 08:41:43,121 - INFO -Time = 0.008400000000000001 | Amount of oil in fishing grounds =
```

Figure 3.4: *Log with n=2500*



```
example\example_log.log
INFO -Time = 0.7939999999999778 | Amount of oil in fishing grounds = 20.107537414731855
INFO -Time = 0.7943999999999778 | Amount of oil in fishing grounds = 20.095121699121307
INFO -Time = 0.7947999999999777 | Amount of oil in fishing grounds = 20.082665058161755
INFO -Time = 0.7951999999999777 | Amount of oil in fishing grounds = 20.070167587668195
INFO -Time = 0.7955999999999777 | Amount of oil in fishing grounds = 20.057629383591113
INFO -Time = 0.7959999999999776 | Amount of oil in fishing grounds = 20.045050542015467
INFO -Time = 0.7963999999999776 | Amount of oil in fishing grounds = 20.03243115916014
INFO -Time = 0.7967999999999775 | Amount of oil in fishing grounds = 20.019771331376695
INFO -Time = 0.7971999999999775 | Amount of oil in fishing grounds = 20.007071155148935
INFO -Time = 0.7975999999999774 | Amount of oil in fishing grounds = 19.994330727091693
INFO -Time = 0.7979999999999774 | Amount of oil in fishing grounds = 19.981550143950187
INFO -Time = 0.7983999999999774 | Amount of oil in fishing grounds = 19.96872950259911
INFO -Time = 0.7987999999999773 | Amount of oil in fishing grounds = 19.955868900041697
INFO -Time = 0.7991999999999773 | Amount of oil in fishing grounds = 19.9429684334091
INFO -Time = 0.7995999999999772 | Amount of oil in fishing grounds = 19.930028199959303
INFO -Time = 0.7999999999999772 | Amount of oil in fishing grounds = 19.91704829707639
INFO -Time = 0.8003999999999771 | Amount of oil in fishing grounds = 19.904028822269634
INFO -Time = 0.8007999999999771 | Amount of oil in fishing grounds = 19.89096987317284
INFO -Time = 0.801199999999977 | Amount of oil in fishing grounds = 19.87787154754325
INFO -Time = 0.801599999999977 | Amount of oil in fishing grounds = 19.86473394326083
INFO -Time = 0.801999999999977 | Amount of oil in fishing grounds = 19.85155715832745
INFO -Time = 0.8023999999999769 | Amount of oil in fishing grounds = 19.838341290866076
INFO -Time = 0.8027999999999769 | Amount of oil in fishing grounds = 19.82508643911974
INFO -Time = 0.8031999999999768 | Amount of oil in fishing grounds = 19.811792701451004
INFO -Time = 0.8035999999999768 | Amount of oil in fishing grounds = 19.79846017634088
INFO -Time = 0.8039999999999767 | Amount of oil in fishing grounds = 19.785088962388098
```

Figure 3.5: *Log with n=2500*

By adjusting the value of nSteps, affects the time step size (delta t), as the two are connected. When nSteps increases, "delta t" decreasee, allowing a finer resolution in the simulation progression. This issue emphasizes the importance of selecting appropriately high values for nSteps to acghive smaller time steps. By doing the accuracey and realsim if the simualtion enhances, ensuring that time-dependent processes are represented more accurately.

4 Discussion

Our simulation successfully showed how an oil spill moves in a coastal area, helping us understand how it spreads over time under different conditions. One of the biggest challenges we faced was the long time it took for the code to run, which made it harder to test the simulation multiple times. This means we need to improve the code to make it faster and more practical for real-time use.

4.1 Implications

The results of our simulation can be very useful for managing environmental issues and planning responses to oil spills. By providing a better understanding of how oil spreads, our model can help decision-makers predict the possible effects of spills and create better plans to reduce damage. This project also showed how important it is to write efficient code for environmental simulations, making sure it's both accurate and fast.

4.2 Limitations

Even though our simulation is helpful, it has some limitations. The biggest issue is the long time it takes to run, which could have been reduced if we had focused more on making the code faster from the start. Also, the model uses some simplified assumptions, like steady water currents and ideal conditions, which might not fully match real-life situations. Another challenge was that we found and fixed problems late in the development process, making it harder to improve the simulation early on. Additionally, we realized the importance of having a clear and structured plan. If we had spent more time understanding the problem at the beginning, we could have avoided some of the confusion and challenges we faced with the code. This would have helped us identify issues earlier and prevented mistakes from being carried into later stages of development. Testing and debugging throughout the process would have also made it easier to catch problems, as we found it difficult to locate the source of errors when we chose to test and debug later in the project.

4.3 Future research

Future improvements should prioritize optimizing the code for faster execution and enabling it to handle larger simulations efficiently. One key focus should be reducing the number of unnecessary conditional (if) statements, as these can slow down the execution when used excessively. Additionally, implementing more memory-efficient methods could further enhance the performance. Regular testing and debugging throughout development should also be emphasized to identify and resolve potential issues earlier in the process.

5 Conclusion

This project successfully developed a Python-based simulation to study the behavior of oil spills in coastal regions. The simulation provided meaningful insights into how oil spreads over time under different conditions, showing how important accurate modeling is for reducing environmental damage. By exploring factors like the number og simulation steps(nSteps), we demonstrated how the simulation affects the accuracy and realism of results.

Despite the project achievements, it also revealed several challenges. One major issue was the long runtime of the simulation, which made it harder to test, debugg and refine the code quickly. The late-stage discovery of errors underscored the need for a more structured development approach, including early testing and debugging, to identify and resolve issues promptly.

Future projects should focus on optimizing the code to improve efficiency and scalability, so the code can run faster and handle larger datas more smoothly. This could involve reducing the use of complex conditions, using memory more efficiently, and improving the algorithms to boost performance. These improvements would not only enhance the simulation's practicality but also ensure more reliable and precise results.

6 Appendix

6.1 Formulas

Formula 1:

$$u(t=0,\vec{x}) = \exp\left(-\frac{\|\vec{x}-\vec{x}_*\|^2}{0.01}\right),$$

Formula 2:

$$\vec{v}(\vec{x}) = \begin{pmatrix} y-0.2x \\ -x \end{pmatrix}.$$

Formula 3:

$$F_i^{(\text{ngh},n)} = -\frac{\Delta t}{A_i} g\left(u_i^n, u_{\text{ngh}}^n, \vec{v}_i, \vec{v}_{\text{ngh}}, \frac{1}{2}(\vec{v}_i + \vec{v}_{\text{ngh}})\right),$$

Formula 4:

$$g(a,b,\vec{v},\vec{v}_{\text{ngh}}) = \begin{cases} a\cdot(\vec{v},\vec{v}) & \text{if } (\vec{v},\vec{v}) > 0, \\ b\cdot(\vec{v},\vec{v}_{\text{ngh}}) & \text{else.} \end{cases}$$

Formula 5:

$$u_i^{n+1} = u_i^n + F_i^{(\text{ngh}_1,n)} + F_i^{(\text{ngh}_2,n)} + F_i^{(\text{ngh}_3,n)}.$$