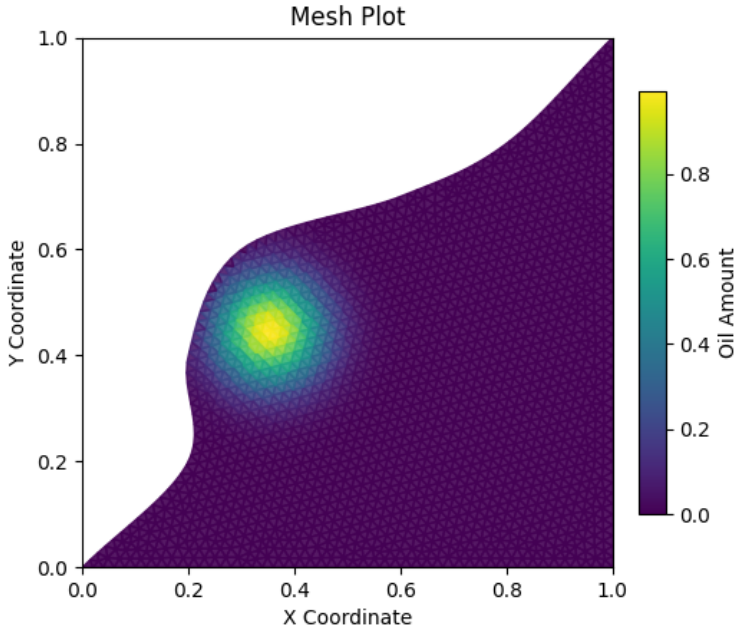
The Art of the Spill

Modeling the Movement of Oil Spills Outside Bay City



Ludvik Høibjerg Aslaksen
Christopher Ljosland Strand
Frederic Ljosland Strand

Project Task in INF202 at the University of Life Sciences

January 2025

1 Introduction

Simulations have become an essential tool for solving complex problems in physics, engineering, and chemistry, where traditional experiments are either too expensive or insufficient. For example, simulating airflow for different airplane designs is often more practical than building multiple prototypes. Computer simulations address these limitations by creating a virtual representation of the physical system, such as the laws of aerodynamics. This allows researchers to accurately predict real-world outcomes.

This project focuses on a fictional scenario involving an oil spill in "Bay City," a coastal town concerned about the potential impact of the spill on its fishing grounds. The simulation models the spread and movement of the oil over time, considering ocean currents. By using a computational mesh, the project predicts the oil spill's movement and its environmental impact on the fishing grounds. These insights are crucial in determining whether drastic countermeasures are needed to protect the marine ecosystem.

2 Simulation Process

The simulation process for modeling the oil spill in Bay City consists of the following steps:

1. **Creating a Computational Mesh:** The domain is divided into a computational mesh consisting of small triangles and lines at the edges. This mesh is read from the file `bay.msh`. Each triangle consist of three Point objects and has an index.

2. **Modeling Initial Distribution:** The oil concentration at time $t = 0$ is modeled as a Gaussian function, centered around the point $\mathbf{x}^* = (x^*, y^*) = (0.35, 0.45)$:
$$u(t = 0, \mathbf{x}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}^*\|^2}{0.01}\right). \tag{1}$$
The oil's movement is governed by the velocity field:
$$\mathbf{v}(\mathbf{x}) = \begin{pmatrix} y - 0.2x \\ -x \end{pmatrix}. \tag{2}$$

3. **Simulating Oil Movement:** Over time, the oil flows according to the velocity field. The simulation computes the flux of oil across each cell edge e_ℓ using the formula:
$$F_i^{(n)} = -\frac{\Delta t}{A_i} g(u_i^n, u_{\text{ngh}}^n, \nu_{i,\ell}, \mathbf{v}_{edge}), \tag{3}$$
where the flux function g is defined as:
$$g(a, b, \nu, \mathbf{v}) = \begin{cases} a \cdot \langle \mathbf{v}, \nu \rangle & \text{if } \langle \mathbf{v}, \nu \rangle > 0, \\ b \cdot \langle \mathbf{v}, \nu \rangle & \text{otherwise.} \end{cases} \tag{4}$$

4. **Direction of flux:** To compute the flow direction across each triangular cell, we use the velocity field at the midpoint of each cell. However, to get the best representation of velocity vectors for the edge cases we take the mean velocity for cells i and ngh:

$$\mathbf{v}_{\text{edge}} = \frac{1}{2}\big(\mathbf{v}_i + \mathbf{v}_{\text{ngh}}\big) \tag{5}$$

To determine the direction of the flux for cell i, we multiply the length of v_{edge} with the scaled outward normal.

$$\nu_\ell = \mathbf{n}_\ell \cdot \|e_\ell\|, \tag{6}$$

The sign of the dot product $\langle \mathbf{v}, \nu_\ell \rangle$ determines the flow direction:

- If $\langle \mathbf{v}, \nu_\ell \rangle > 0$, the flow is outward from the current cell, and the flux is computed based on the oil concentration within the cell.
- If $\langle \mathbf{v}, \nu_\ell \rangle \leq 0$, the flow is inward to the current cell, and the flux depends on the oil concentration in the neighboring cell.

5. **Updating Oil Distribution:** The total oil concentration in a triangular cell i at the next time step t_{n+1} is computed as:

$$u_i^{n+1} = u_i^n + \sum_{\ell=1}^{3} F_{\text{ngh},i,\ell}^{(n)}. \tag{7}$$

6. **Visualizing Results:** The simulation generates plots at regular intervals (specified in the configuration file) to show the oil spill's movement over time.

3 User Guide

This package offers the user options for customization to your need. Here is a guide to how to use our package.

3.1 Configuration file

The program uses a configuration file to customize its behavior. Below is an example configuration file:

```
[settings]
nSteps = 100
t_start = 0.1
t_end = 1.0

[geometry]
```

```
    filepath = "meshes/bay.msh"
    fish_area = [[0.0, 0.45], [0.0, 0.2]]
    initial_oil_area = [0.35, 0.45]

    [IO]
    logName = "logfile"
    writeFrequency = 5
```

Replace `example.toml` with the path to your custom configuration file.

3.2 Restarting a Simulation

We have also added the option to restart the simulation to extend the timespan without calculating the change from the beginning. Then you have to write this below [IO]:

```
    restartFile = "results/default_experiment_results/input/
       restartFile.txt"
```

3.3 How to run the program

To run the program, use the following command in the terminal:

```
    python main.py -c example.toml
```

You can also run through all your configuration files by typing this command in your terminal:

```
    python main.py --find-all -f examples/
```

4 Project Structure

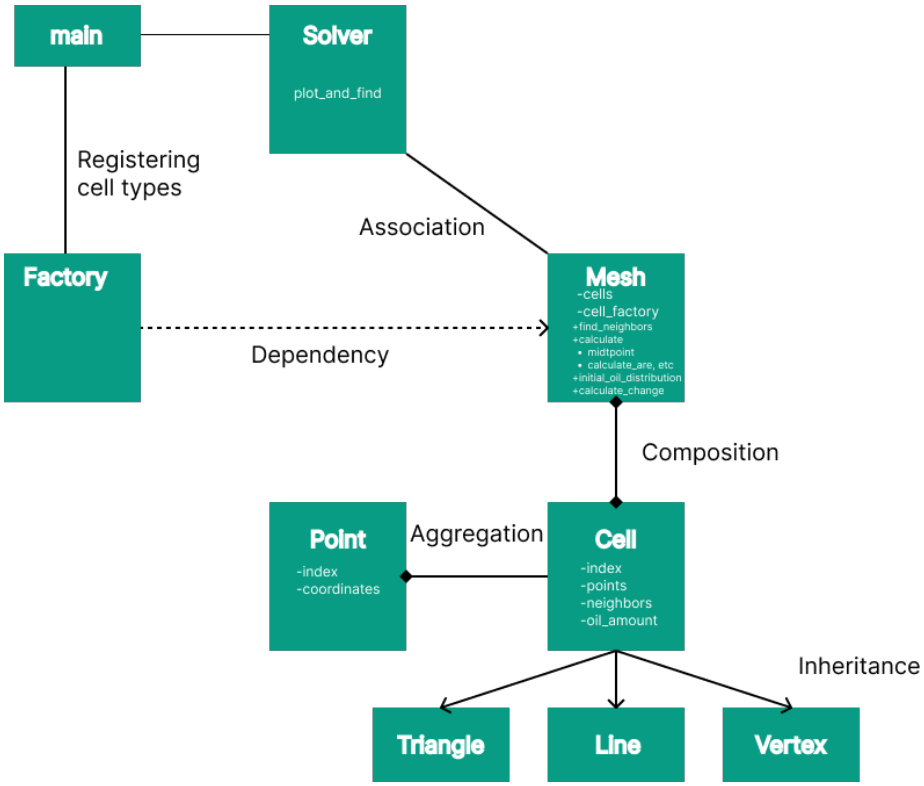
The overall file and folder structure is made for the possibility to publish it as a Python package. This can be done by including setup files and publish the module to PyPI. By using configuration files and a clear structure, it would be easy for a external person to use the simulation code. The code is also structured to make room for scalability and modularity. By organizing the functionality into distinct modules `mesh.py`, `solver.py`, and `cells.py` each with options for easy extensions for different shapes and problems, people can alter the code without affecting the entire system.

4.1 Folder structure

- **meshes/**: Contains mesh files for the computational map.
- **src/**: Contains the source code with subdirectories for the code
  - **Simulation/**: Main directory containing the simulation package/scripts
    - `cells.py`: Contains the Point and Cell class, storing the coordinates and the oil amount in the cell
    - `create_video.py`: Contains functions that creates videos of the simulation results using opencv.
    - `mesh.py`: Holds the mesh class containing the cells and using a the cell factory to read the mesh `.msh` files. This class also contains a lot of the math functions used in the simulation. The idea behind this is to calculate a lot of the cell's properties in the beginning to save time and make sure the calculations are not called multiple times.
    - `plotting.py`: Plots the result/simulation at each $writeFrequency$ timestep.
    - `solver.py`: Performs the actual simulation. The function "find and plot" creates the folder structure for each experiment, checks for restartFile and calulates the flux for each timestep.
  - **tests/**: Contains unit tests or integration tests that verify the correctness of various modules.
  - **config.py**: A Python script for parsing the configuration files so they can be used in the main.py
  - **input.toml**: Default configuration file
  - **logger.py**: Provides logging functionality to capture simulation progress, errors, or other diagnostic messages.
  - **main.py**: The central script to run the simulation. It parses command-line arguments or reads from `input.toml`, initializes the mesh and solver, and organizes the entire process.

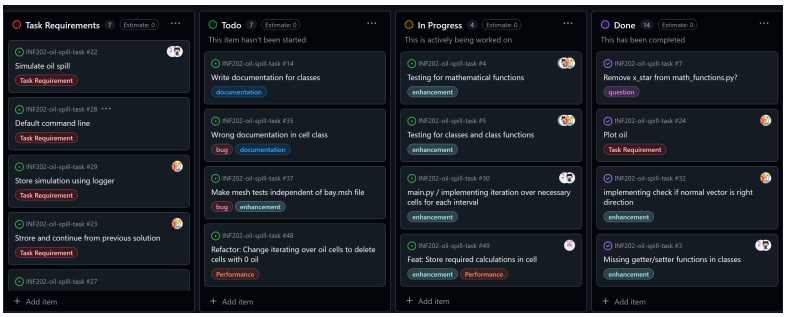
4.2 UML Diagram

We have also made an UML diagram to visualize how the classes interact with each other:



5 Agile Development

- `Our Approach`: Firstly we agreed on structure for varible names and such in the *structure.txt* file and then we drew a outline for the software on a whiteboard with inspiration from the mesh task in INF201. Then we diveded tasks trying to implement things before tying them together afterwards. This was the generel approach throughout the task as well.
- `Git board`: After implementing the fundamental applications we started using the git board more for issues needing to solved and such.



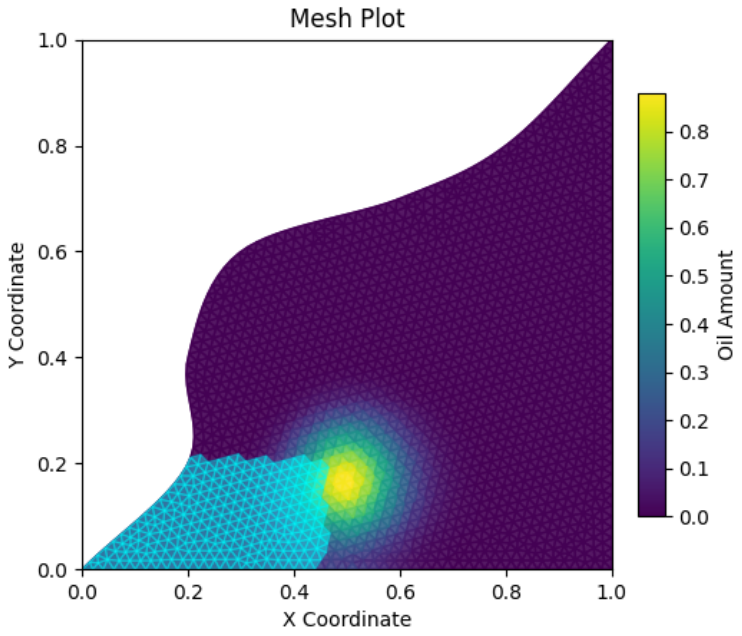
– `Story mapping`: This is an Agile planning technique that visualizes the user's journey and breaks it into smaller tasks or user stories. This can give the team an overview of the product or feature. The tasks are organized horizontally, while the priority is organized vertically. In our story mapping we have our main tasks as headers, while subtasks of these are organized below in their respected priority. In our case we have classified Minimal Viable Product (MVP) as just b eing able to plot the initial spill. In a bigger application it would make more sense to plan the project this way.

PRIORITY	Read mesh	Oil distribution	Plotting	Config and logging
MVP	Reading mesh in mesh class and creating cells with cell.type	Finding initial distribution with given formula	Plotting initial distribution	
HIGH		Working flux for each timestep	Plotting for each timestep	argparse for reading config file
MEDIUM	Cell Factory		Making a video from the plots	option to run --find-all for config files
LOW				

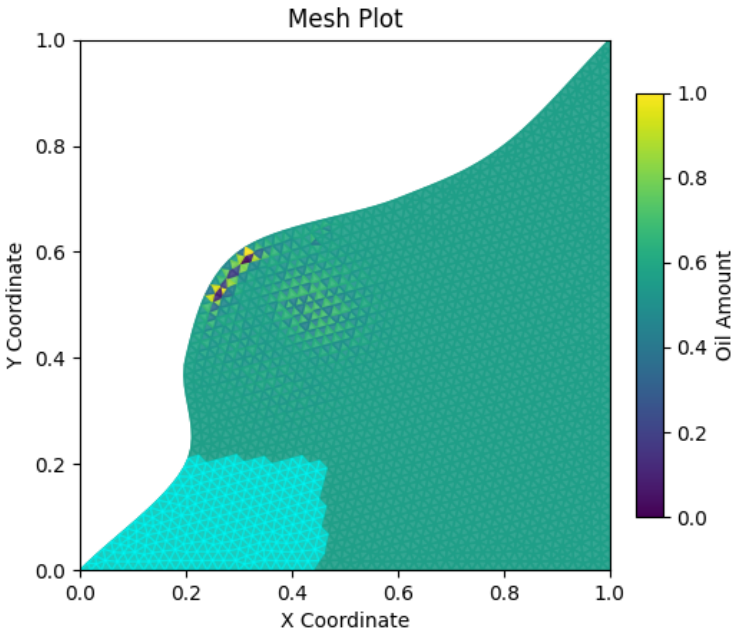
6 Results

The task was to investigate if the fish area will be affected by the oil spill. Therefore, we judge the result by what extent the oil reaches the fish area.

Given the initial conditions given in the configuration file from earlier, there will be some oil in the fish area, but the worst is avoided. By adding up the oil amount in the cells within the fish area, we get 39.66 oil in given unit in the fish area. To determine if the oil amount is above a dangerous threshold, we would check with a field expert. Below is a visualization of the oil passing the area.



We've also experimented with different values to see how they effect the result. By doing this, we've discovered that by using too big timesteps the results wont be realistic. This is because it will not match the magnitude of the velocity and therefore the numbers will create some strange results. For example, if the theoretical distance covered given timestep and velocity is 2 cells, but it can only flow to neighboring cells it won't work. We've also found a timestep around 0.01 as the most sensible, since it balances the movement per timestep the best. See image below:


Mesh Plot
Y Coordinate
X Coordinate
Oil Amount
0.0
0.2
0.4
0.6
0.8
1.0