Norwegian University of Life Sciences (NMBU)

Oil Spill Simulation: Project Report

Group 2: **Roland Patrick Fiksdal, Isma Sohail, Srivatsav Saravanan**

January 22, 2025

Contents

1 Overall Problem

The fictional fishing town of **Bay City** has experienced an oil spill, threatening local fishing grounds. The objective of this project is to simulate the spread of oil over time using a numerical simulation over multiple time steps. The initial oil spill is given by a Gaussian function, and the simulation models the oil movement through triangular cells driven by a predefined velocity field, ensuring accurate predictions to assist in environmental decision-making.

2 User Guide

2.1 Executing the Simulation

To execute the simulation, navigate to the project directory named `OilSimulateProject`. Run the program with the default parameters by entering the following command in your terminal:

```
python main.py
```

The user can specify a particular configuration file for the simulation by utilizing one of the following options:

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

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

Additionally, to designate a specific directory for the configuration file, include a folder argument as shown below:

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

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

To simulate all valid configuration files within a specified folder, use the following command:

```
python main.py --find all -f folder_name
```

If certain arguments are omitted, the program will default to predefined values for the folder name and configuration file, based on the context:

- If only a folder argument is provided, the program will search for a configuration file named `input.toml` within the specified folder:

  ```
  python main.py -f folder_name
  ```

- If only a configuration file argument is supplied, the program will search for the specified file within the default folder, `config_files`:

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

2.2 Configuring a .toml file

A look inside the default configuration file `input.toml` provided with the project:

```
# Default configuration file
[settings]
nSteps = 600 # number of time steps
tStart = 0 # start time (optional)
tEnd = 0.6 # end time

[geometry]
meshName = "bay.msh"
oilSpillCenter = [0.35, 0.45]
borders = [ [0.0, 0.45], [0.0, 0.2] ] # define where fish are located

[IO]
logName = "log" # name of the log file created
writeFrequency = 15 # Frequency of output video. If not provided, no video is
# restartFile = "solutions/solution.txt"
# Restart file must be provided if start time is provided.
```

Both `[settings]` and `[geometry]` are mandatory top-level sections that must be included in the configuration file for the program to process it successfully. The `[settings]` section is required to define the keys `nSteps` and `tEnd`, while the `[geometry]` section must specify the keys `meshName`, `oilSpillCenter`, and `borders`.

```
# Example
[settings]
nSteps = 500
tEnd = 0.5

[geometry]
meshName = "bay.msh"
oilSpillCenter = [0.35, 0.45]
borders = [ [0.0, 0.45], [0.0, 0.2] ]
```

The top-level section `[IO]` is optional, as all keys within this section are non-mandatory. To exclude an optional key, either remove its line entirely or comment it out using `#` for convenient toggling between enabling and disabling. Every time the configuration file is ran, an solution file will be generated inside the `solutions` folder. A solution file can be used a restart file, if you want to run a continue a simulation from the end of another simulation – just make sure to match tStart with the time provided at the top of the solution file.

2.3 Key Features

- Simulate oil spread using a velocity field and computational mesh.

- Restart simulations from saved states using restart files.
- Generate visual outputs such as videos and oil distribution plots.
- Log key simulation parameters and results for analysis.

3 Code Structure

The program follows an object-oriented design to ensure extensibility and maintainability. Below are the core components:

OilSimulationProject Folder Structure

- **config_files**
  - `input.toml` - default toml file
- **data/mesh**
  - `bay.msh`
- **results** - folder to store the results of `animation.gif` and the result image
- **solutions**
- **src**
  - **cell**
    - `__init__.py`
    - `base_cell.py`
    - `line_cell.py`
    - `triangle_cell.py`
  - **io**
    - `__init__.py`
    - `config_reader.py`
    - `mesh_reader.py`
    - `solution_reader.py`
    - `solution_writer.py`
  - **simulation**
    - `__init__.py`
    - `simulator.py`
  - **visualization**

    - `__init__.py`
    - `plotter.py`

- **tests**
- `conftest.py`
- `main.py`
- `requirements.txt`

The chosen code structure effectively balances organizational clarity with the need to manage a reasonable number of files. This design ensures an intuitive workflow, allowing us or future users to quickly locate and modify specific parts of the code, thereby streamlining the development process and enhancing overall efficiency. Additionally, the structure facilitates straightforward integration of new features, such as computational meshes with new cell types (e.g., quadrangles), by providing a logical framework for their implementation.

4 Mathematical Model

4.1 Initial Oil Distribution

The initial oil concentration $u(t=0,\vec{x})$ is modeled using a Gaussian function centered at $(x^\star, y^\star) = (0.35, 0.45)$:

$$u(t=0,\vec{x}) = \exp\left(-\frac{\|\vec{x}-\vec{x}^\star\|^2}{0.01}\right). \tag{4.1}$$

4.2 Velocity Field

Oil movement is governed by the velocity field $\vec{v}(\vec{x})$:

$$\vec{v}(\vec{x}) = \begin{pmatrix} y - 0.2x \\ -x \end{pmatrix}. \tag{4.2}$$

4.3 Flux Computation

Flux across a triangular cell edge is calculated as:

$$F_i^{(n)} = -\frac{\Delta t}{A_i} g(u_i^n, u_{\text{ngh}}^n, \vec{\nu}_{i,\ell}, \vec{v}_i), \tag{4.3}$$

where the flux function g is defined as:

$$g(a,b,\vec{\nu},\vec{v}) = \begin{cases} a\langle \vec{v}, \vec{\nu}\rangle, & \text{if } \langle \vec{v}, \vec{\nu}\rangle > 0, \\ b\langle \vec{v}, \vec{\nu}\rangle, & \text{otherwise.} \end{cases} \tag{4.4}$$

5 Results and Analysis

5.1 Oil Distribution

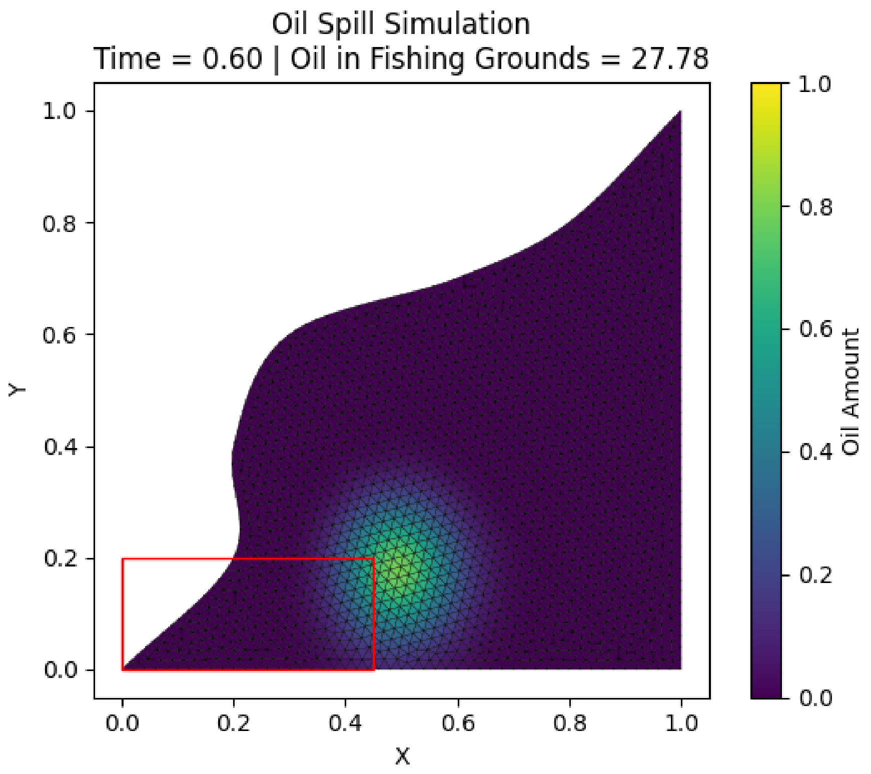

Figure 5.1: Result of the oil distribution at t = 0.60 (nSteps = 600)

Figure 5.1 shows that oil amount inside the fishing ground at t = 0.60 is 27.78 when using 600 simulation steps up until t = 0.60. Figure 5.2 shows that using a small number of simulation steps will result with the flux not being calculated correctly.

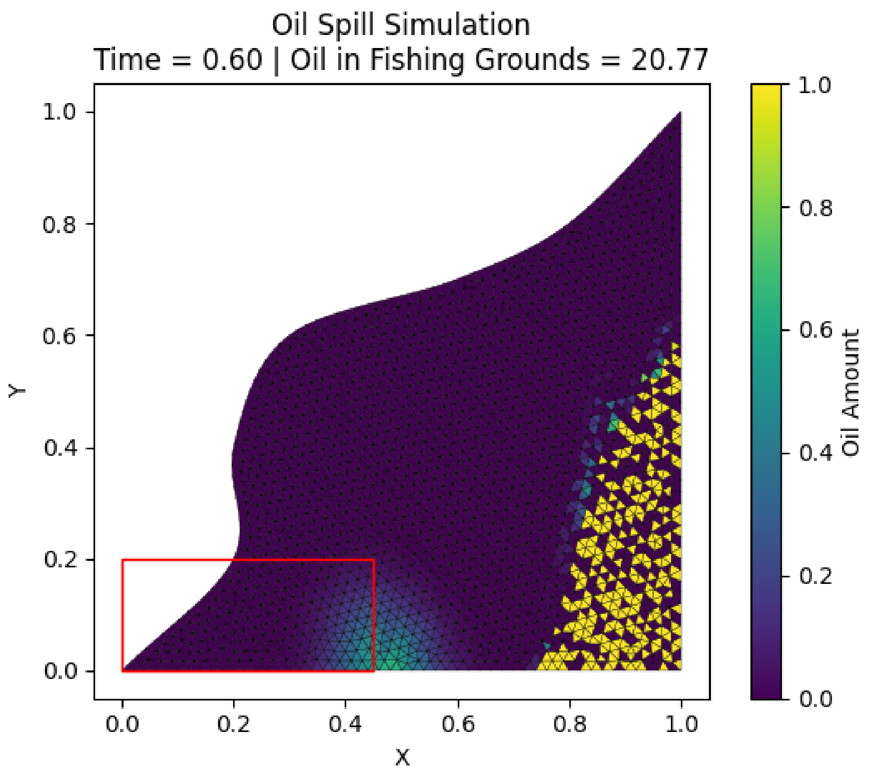

Figure 5.2: Result of the oil distribution at t = 0.60 (nSteps = 50)

5.2 Impact on Fishing Grounds

Given the fishing ground borders `[ [0.0, 0.45], [0.0, 0.2]]`, the simulation indicates that a substantial portion of the spilled oil will flow into the fishing grounds of Bay City. Table 5.1 presents the oil amount within these boundaries over time. Notably, the results suggest that the fishing grounds experience the highest oil amount at approximately t = 0.7

Time (t)	**Oil amount**
0.0	0.03
0.1	0.28
0.2	1.48
0.3	5.04
0.4	11.86
0.5	20.57
0.6	27.78
0.7	30.60
0.8	28.33
0.9	22.34
1.0	14.98

Table 5.1: nSteps = 1000 | tStart = 0 | tEnd = 1.0

6 Conclusion

This project demonstrates a robust and flexible approach to modeling oil spill behavior in Bay City. By combining a carefully structured codebase with a well-defined mathematical framework, the simulation provides meaningful insights into how oil moves through the local fishing grounds under a predefined velocity field. The use of an object-oriented design not only facilitates clear organization and future feature additions—such as new cell types or alternative meshes—but also ensures that the solution is maintainable over time.

From the results, it is evident that an adequate number of simulation steps is crucial for accurately capturing oil dispersion; reducing the time steps can lead to significant inaccuracies in flux calculations. The simulation also highlights key moments when the oil load in the fishing grounds peaks, offering a clear basis for planning mitigation efforts. Overall, the software serves as a practical decision-support tool for environmental management, helping stakeholders to predict spill trajectories and take timely action to safeguard marine ecosystems and local fisheries.