Don't Cry Over Spilled Oil: A Numerical Simulation

Håkon Strømfjord Kjellsen
Sverre Stikbakke
Heidi Tønnesson

Norwegian University of Life Sciences (NMBU)

January 21, 2026

Contents

Chapter 1

Introduction and problem statement

Oil spill in coastal areas can have large severe consequences for both environment and local fisheres. To assess risk and implement appropriate measures quickly, it is essential to understand how oil is transported by ocean currents over time. In this project, we study a simplified version of this exact problem. How a velocity field transports oil across a triangular mesh, and how the oil is accumulated in specific areas, such as fishing grounds.

1.1 Problem statement

The small Fishing town of Bay City has experienced oil spill and requires immediate assessment of how the oil is conveyed by the ocean currents and whether or not it will affect the neighbouring Fishing grounds.

This project's objective is to develop a numerical simulation that models the developments in the oil distribution over time on the ocean surface. The oil transport is controlled by a predefined two-dimensional velocity field and is simulated on a mesh grid representing the coastal area.

The simulation must precisely model the flux of oil between neighbouring mesh cells over a given time, and impart quantative results describing the oil concentration in the area of interest. In addition to creating visualizations and output data, the program must be designed to be intuitive, structured, and easily extendable while following modern software development practices.

Chapter 2

User guide

The aim of this chapter is to describe how a user can run the simulation, what configuration options exists and how a user can generate plots and videos. This guide is directed towards an end user that does not need to understand the code, but only use the program as a tool.

2.1 Run the program

The program is executed by sending `python main.py` in the terminal. This command will launch the standard setup using `input.toml`.

2.2 Configuration files

The simulation is controlled by a set of input parameters contained in TOML-format. A typical TOML-file has three sections as described in the table below.

Section	**Parameter**	**Description**
[settings]	`nSteps`	Number of time steps in the simulation
[settings]	`tEnd`	Total simulation time
[geometry]	`meshName`	Name of mesh file (must exist in main directory)
[geometry]	`borders`	Fishing grounds borders
[IO]	`logName`	Name of the log file
[IO]	`writeFrequency`	How often images is saved to video

`logName` and `writeFrequency` are optional fields. If the user wishes to omit `logName`, it must be removed entirely from the `[IO]` section (or comment it out using `#`). It must *not* be left blank, as empty values are invalid TOML syntax. When `logName` is omitted, the program automatically uses the default name `"logfile"`.

The same logic applies to `writeFrequency`. If the field is omitted or commented out, no video will be produced. A value must not be left empty.

2.3 Run specific configuration files

Using `case1.toml` as an example TOML-file and `configs` as the example folder with TOML files in the main project folder, a specific configuration file can be run using

```
python main.py -c case1.toml
```

or equivalently

```
python main.py --config_file case1.toml
```

If the file is located in a different folder, the relative path must be given, for instance

```
python main.py -c configs/case1.toml
```

To run all `.toml` files in a given folder, the user can use the following command:

```
python main.py --find_all --folder configs
```

or with the short flag:

```
python main.py --find_all -f configs
```

This will detect all configuration files in the folder `configs` and run each simulation sequentially, storing the results in separate folders.

2.4 Output and results

When a simulation is ran (e.g., `"config_name"`), a new folder is created under `results/"config_name"`, which will then contain the following:

File	**Description**
`[simulation_plot.png]`	Plot at final time of simulation
`["logname".log]`	Log file with all input parameters and oil at each time step
`[simulation.mp4]`	Video of the oil development

2.5 Error handling

Config errors are handled with `config.py` and provides error messages if the configuration file fails. This can happen if incompatible values are input wrongly, not in line with the input parameter table in Section 2.2 above.

Chapter 3

Numerical Problem and Code Implementation

This chapter entails how the oil transport is defined as a numerical problem and how this logic is implemented in the programming architecture.

3.1 Oil flow

Ultimately, the goal is to calculate the amount of oil in the fishing grounds over time, which can be calculated by the sum of products between the oil concentration for a given triangle and a given time, as well as its area that has its center point within the borders of the fishing grounds. This can be seen in Equation 3.1. In addition, a tabulated nomenclature explaining the symbols is provided at the end of this section.

$$Oil_{fg}^{t+1} = \begin{cases} \sum u_i^{t+1} A_i, & \text{if } x_{fg_{min}} \leq x_i \leq x_{fg_{max}} \text{ and } y_{fg_{min}} \leq y_i \leq y_{fg_{max}} \\ 0, & else \end{cases} \tag{3.1}$$

Furthermore, in order to calculate the oil concentration for triangle i at time $t+1$, the current oil concentration and the total flux needs to be taken into account, as can be seen in Equation 3.2

$$u_i^{t+1} = u_i^t + F_i \tag{3.2}$$

In order to calculate the total flux, the edge fluxes between triangle i and each neighbour ngh_n needs to be summed up, befre multiplying with the time step and the respective area for triangle i. See Equation 3.3

$$F_i = -\frac{\Delta t}{A_i}(g_{ngh_1} + g_{ngh_2} + g_{ngh_3}) \tag{3.3}$$

The edge flux is determined by the dot product between the scaled normal of the sharing edge between neighbour ngh_n and triangle i ,and the mean flow between the two midpoints of triangle i and ngh_n. If the dot product is larger than zero, the oil will flow over to the neighbour, but if it is smaller or equal to zero, the oil will flow from the neighbour ngh_n over to the triangle i. This can be seen in Equation 3.4.

$$g_{ngh_n} = \begin{cases} u_i^t \, \eta_{i,ngh_n} \cdot \dfrac{v_i + v_{\text{ngh}}}{2}, & \text{if } \eta_{i,ngh_n} \cdot \dfrac{v_i + v_{\text{ngh}}}{2} > 0 \\ u_{ngh_n}^t \, \eta_{i,ngh_n} \cdot \dfrac{v_i + v_{\text{ngh}}}{2}, & \text{if } \eta_{i,ngh_n} \cdot \dfrac{v_i + v_{\text{ngh}}}{2} \leq 0 \end{cases} \tag{3.4}$$

The initial oil distribution is centred around the source point, seen in Equation 3.5

$$\vec{x}_* = (x_*, y_*)^\top = (0.35,\ 0.45)^\top, \tag{3.5}$$

and is given by the Gaussian profile

$$u(t=0, \vec{x}) = \exp\left(-\frac{\|\vec{x} - \vec{x}_*\|^2}{0.01}\right), \tag{3.6}$$

where $u(t, \vec{x})$ denotes the oil concentration at position $\vec{x}$ and time t.

The fishing grounds are defined as the rectangular region

$$[0.0,\ 0.45] \times [0.0,\ 0.2], \tag{3.7}$$

that is, all mesh cells whose centre points satisfy $0.0 \leq x \leq 0.45$ and $0.0 \leq y \leq 0.2$. The movement of oil is dictated by the underlying flow field

$$v(\vec{x}) = \begin{pmatrix} y - 0.2x \\ -x \end{pmatrix}. \tag{3.8}$$

Below a tabulated nomenclature over the symbols used in the formulas above is provided.

Symbol	**Explaination**
i	Index of current cell
ngh_n	Index of neighbouring cell n of i
u_i^t	Oil concentration in cell i at current time t
u_i^{t+1}	Oil concentration in cell i at time $t+1$
F_i	Total oil flux of cell i
A_i	Area of cell i
η_{i,ngh_n}	Product of edge normal and edge length (scaled normal)
$[(x_{fg_{min}}, x_{fg_{max}}), (y_{fg_{min}}, y_{fg_{max}})]$	Borders for fishing grounds
$[x_i, y_i]$	Center point coordinates for triangle i
Oil_{fg}^{t+1}	Total amount of oil in the fishing grounds at time $t+1$
v_i and v_{ngh_n}	Vector flow for center points i and ngh_n

3.2 Program architecture

The program is divided into a set of independent modules: `config.py` interprets and validates user input from Command Line Interface (CLI), while `controller.py` handles these inputs and folder structures. `Mesh`, `Cell` and `Simulation` makes up the model itself. `Mesh` reads in the geometry, creates cells via `CellFactory` and builds topology. `Simulation` makes instances of `Mesh` to calculate fluxes and update oil concentrations over time. This structure provides clear areas of responsibility and makes it easy to change specific parts of the system without intereferring with the other components. These relationships are summarized in the UML diagram in Figure 3.1, which illustrates how the main classes interact.

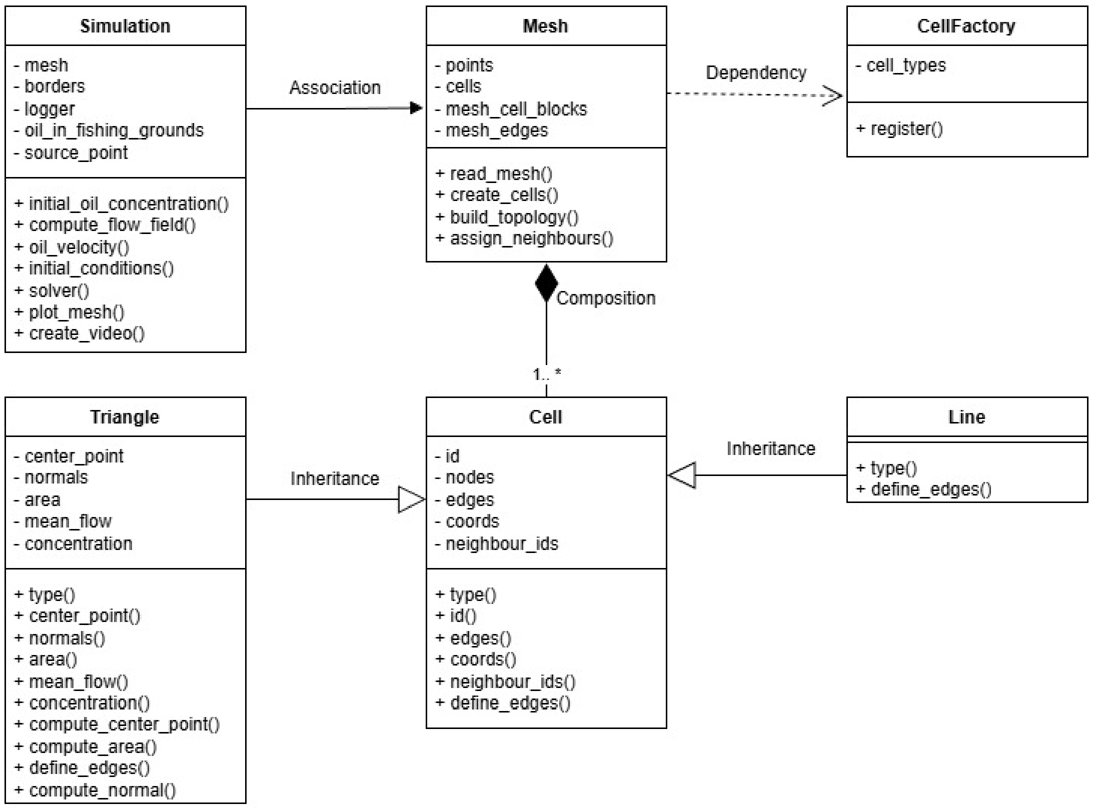


Figure 3.1: UML-diagram of the class structure

3.2.1 Data encapsulation and extendability

Common data for cells lies in the abstract base class `Cell`, while `Triangle` and `Line` provides specified attributes for themselves. This is reflected in the UML diagram, which shows inheritance from `Cell` as well as the composition between `Mesh` and `Cell`. As can also be seen in the diagram, `Simulation` only know `Mesh` and is not connected to the cells internal details, which makes it possible to easily add new cell types without needing to change the solver (e.g., a quadrangle). `CellFactory` is placed outside the inheritance hierarchy and creates new cells for `Mesh`. This provides both encapsulation of data and possibilities for extendability.

3.2.2 Performance considerations

Some concrete examples are provided here. Geometrical data such as area, normals and center points are only calculated and saved once in `Triangle` so they dont need to be calculated for each iteration. A temporary structure is used for concentrations, before they are saved back to the cells. NumPy is used where it makes sense to focus on performance on behalf of flexibility.

3.2.3 Testing and validation

Different tests has been conducted, including integration tests and acceptance tests in the code, however, a special emphasis was put on unit testing on the geometry and topology of the cells in the mesh. Figure 3.2 displays the minimal test mesh that was created in order to validate correct neighbouring cells, cell ID's, node ID's; both visually and manually numerically.

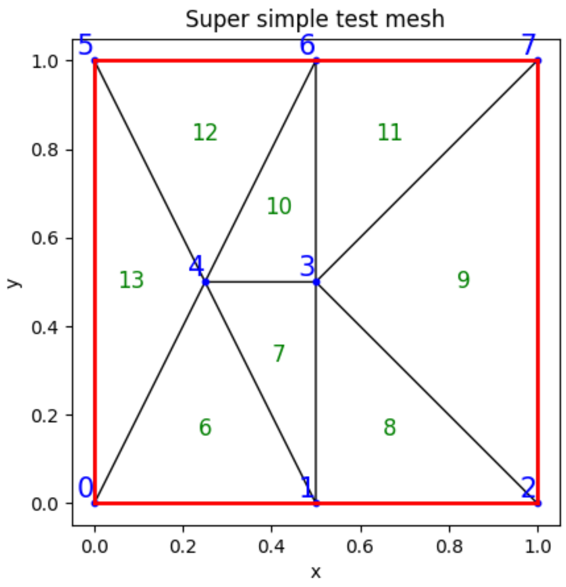


Figure 3.2: Simple test mesh for topology testing

Furthermore, Figure 3.3 displays the geometrical testing, looking into single triangle cells. This test entailed the length and direction of normals, as well as center points and areas.

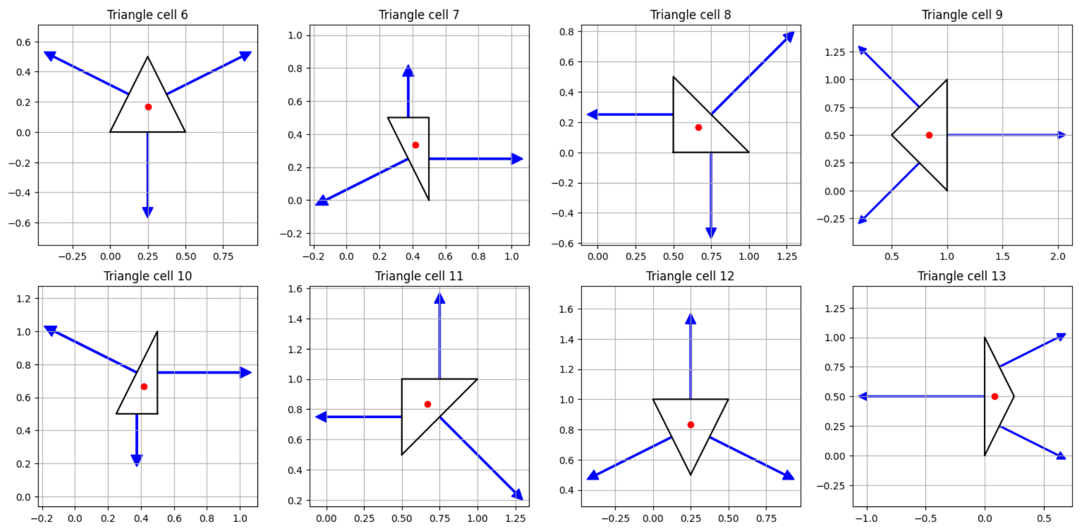


Figure 3.3: Isolated triangles from simple test mesh

Chapter 4

Agile development

Saying that the group was overwhelmed in the initial stages of the project is an understatement. In order for the group to deliver a top-quality result, and at the same time maximize learning outcome for each member, a clear guiding framework on how to tackle the problem was essential.

4.1 Problem approach

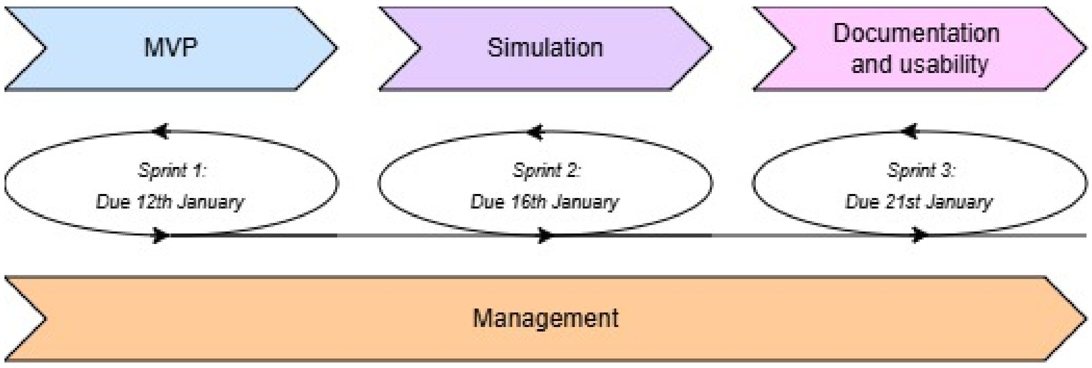
The clear defined problem statement in Chapter 1 was prerequisite to decompose the problem into smaller ones. The problem was broken down into four main components: the raw logic behind the oil movement in mesh (1), implementation of this logic into a full working simulation program (2), the delivery of the program for an end user (3) and a governing component overseeing and facilitating the development (4).

4.2 Agile methods

From an agile or scrum perspective, the four components can be viewed as product increments or user stories. They can be labeled as below:

- *Minimum Viable Product (MVP)*: The purpose of this product increment was to create a minimum working product, using nothing but functions to create a working oil flow simulation.
- *Simulation*: The goal of this product increment was to implement the logic into a fully working program using Object Oriented Programming methods, to ensure data encapsulation, extendability and efficency.
- *Documenation & Usability*: The focus of this product increment was to ensure that everything was properly documented, organized, tested, intuitive and done according to the task description.
- *Management*: Whether or not management is a product increment can be discussed, but for learning purposes, it was included. Management was running in the background continuously, facilitating the other user stories and made sure the necesary infrastructure was in place (`GitLab`) and ensuring alignment between all members.

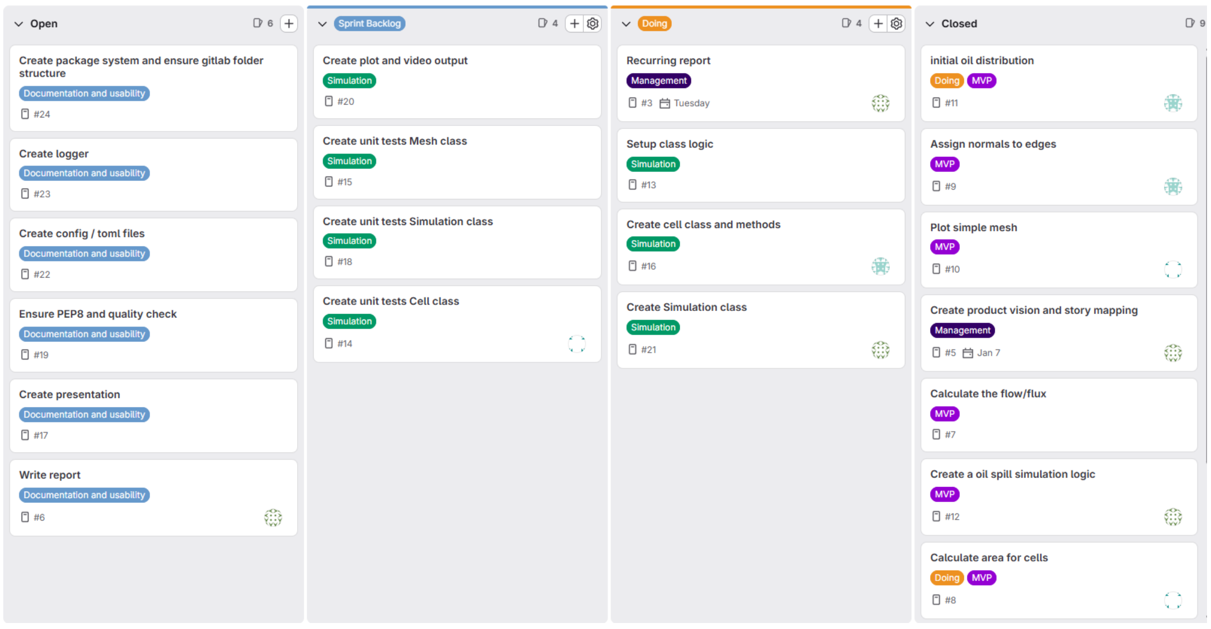
Figure 4.2 is a hybrid of EPIC, story mapping and process orientation and was modified according to the group needs. Here each product increment was assigned to a sprint.



MVP
Simulation
Documentation and usability
Sprint 1: Due 12th January
Sprint 2: Due 16th January
Sprint 3: Due 21st January
Management

Figure 4.1: Process story board with tasks assigned to MVP, Simulation, Documenation and management

Operationally, the group had daily meetups and sprint reviews as well as updating the story board on `GitLab`. This can be seen in Figure 4.2, where *specific tasks* are assigned to each product increment as *labels* in *Open, Sprint Backlog, Doing* and *Closed*. The group also frequently switched roles in order to be exposed to all areas of the projects.



Open 6

- Create package system and ensure gitlab folder structure
  Documentation and usability
  #24
- Create logger
  Documentation and usability
  #23
- Create config / toml files
  Documentation and usability
  #22
- Ensure PEP8 and quality check
  Documentation and usability
  #19
- Create presentation
  Documentation and usability
  #17
- Write report
  Documentation and usability
  #6

Sprint Backlog 4

- Create plot and video output
  Simulation
  #20
- Create unit tests Mesh class
  Simulation
  #15
- Create unit tests Simulation class
  Simulation
  #18
- Create unit tests Cell class
  Simulation
  #14

Doing 4

- Recurring report
  Management
  #3 Tuesday
- Setup class logic
  Simulation
  #13
- Create cell class and methods
  Simulation
  #16
- Create Simulation class
  Simulation
  #21

Closed 9

- initial oil distribution
  Doing MVP
  #11
- Assign normals to edges
  MVP
  #9
- Plot simple mesh
  MVP
  #10
- Create product vision and story mapping
  Management
  #5 Jan 7
- Calculate the flow/flux
  MVP
  #7
- Create a oil spill simulation logic
  MVP
  #12
- Calculate area for cells
  Doing MVP
  #8

Figure 4.2: Snapshot of the story board taken around mid-course

Chapter 5

Results

This chapter goes through the results from the simulation with two sensitivity analyses. One for time discretization and a second one for the geometry of the fishing grounds. The aim is to examine how the simulation result is impacted by choice of input parameters and how the area of interest is defined. Not all input parameters will be relevant (i.e., `writeFreq`) but are still kept for illustrative purposes.

5.1 Sensitivity analysis: time discretization

The numerical stability is dependent on `delta_t`. To investiage this, three simulation cases, with input parameters in the table below, were ran with variable `nSteps`, hence creating a varible `delta_t`. See Figure 5.1, 5.2 and 5.3 for results

Case	`nSteps`	`tEnd`	`delta_t`	`meshName`	`borders`	`writeFreq`
A	500	0.5	0.001	`bay.msh`	[0.0, 0.45], [0.0, 0.2]	20
B	50	0.5	0.01	`bay.msh`	[0.0, 0.45], [0.0, 0.2]	20
C	5	0.5	0.1	`bay.msh`	[0.0, 0.45], [0.0, 0.2]	20

- Case A: Acts as a referance configuration with small `delta_t` which gives a stable development of the oil.
- Case B: Simulation stays stable even though `delta_t` is increased, but the solution becomes increasingly numerical diffuse.
- Case C: `delta_t` is increased further to provoke numerical instability. The oil concentration breaks physics already after a few iterations and big parts of the mesh now has unrealistic values.

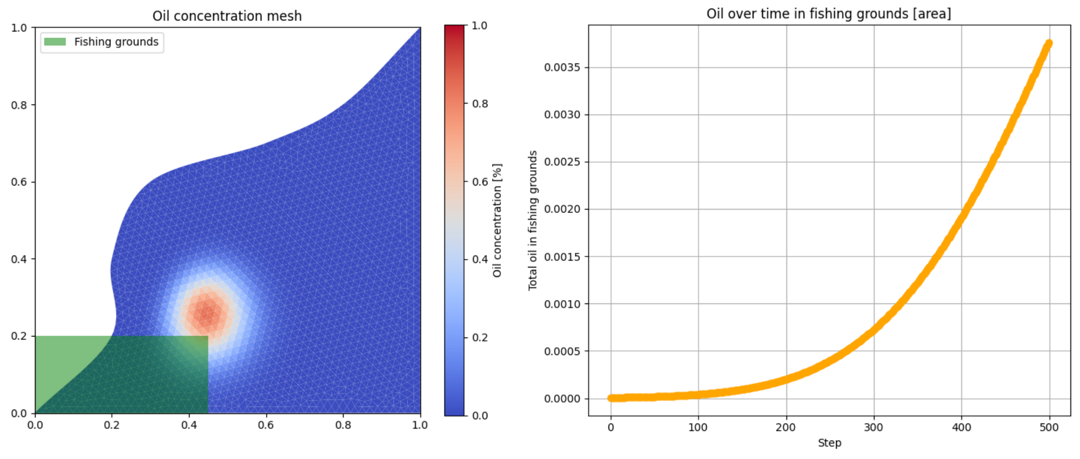


Figure 5.1: Case A, `delta_t`=0.001

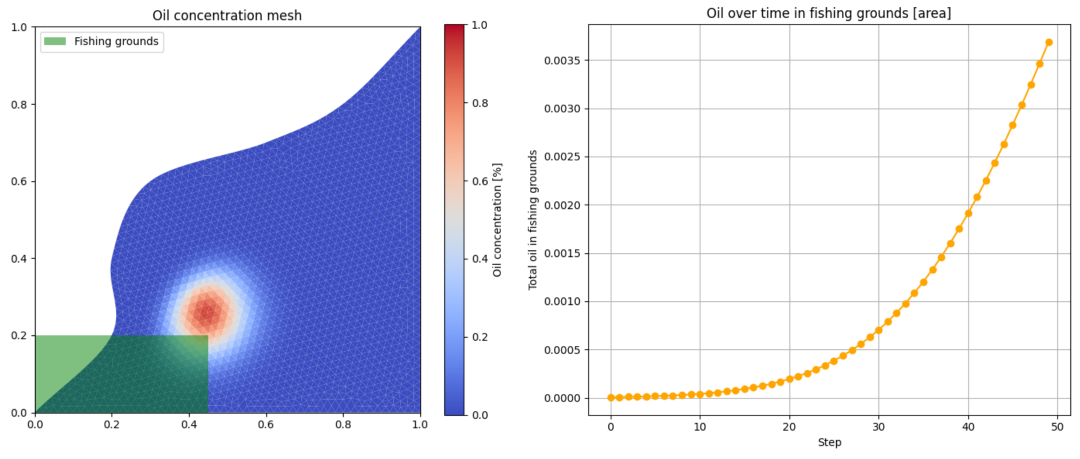


Figure 5.2: Case B, `delta_t`=0.01

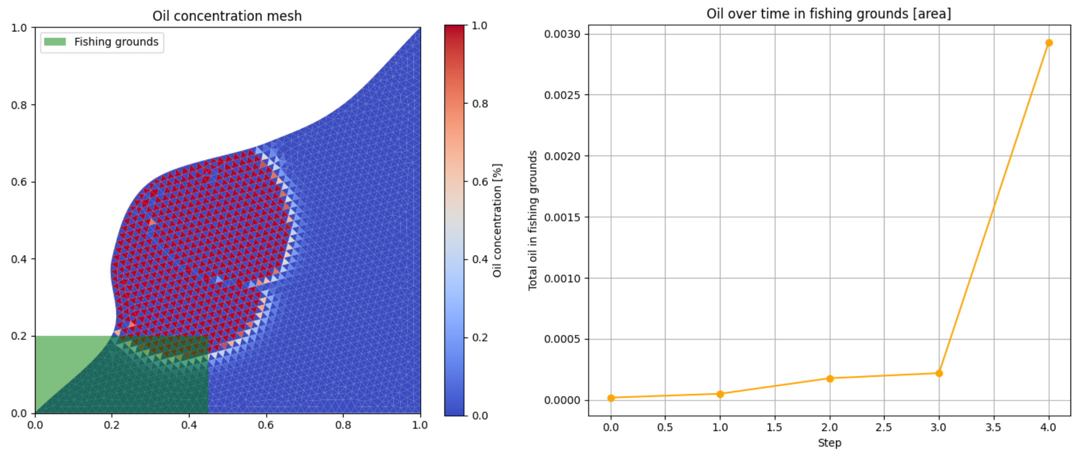


Figure 5.3: Case C, `delta_t`=0.1

5.2 Sensitivity analysis: geometry (fishing ground borders)

The oil amount in the fishing grounds is strongly dependent on where the borders of the grounds are defined in relation to the source point. To test this, the `borders` were variable while all other parameters were fixed. Figures 5.4, 5.5, 5.6 shows how the total oil amount in the fishing grounds is impacted if we geometrically change the `borders` while all other input stays equal.

Case	`nSteps`	`tEnd`	`delta_t`	`meshName`	`borders`	`writeFreq`
D	500	0.5	0.001	`bay.msh`	[0.0, 0.45], [0.0, 0.2]	20
E	500	0.5	0.001	`bay.msh`	[0.2, 0.5], [0.3, 0.6]	20
F	500	0.5	0.001	`bay.msh`	[0.5, 0.8], [0.4, 0.8]	20

- Case D: Again, acts as a reference and is down stream of the oil source point. The oil amount can be seen as starting at zero and is increasing as the oil drifts into the borders
- Case E: The borders of the fishing grounds are now placed around the source point. This gives high initial oil concentration, but as the oil is transported downstream the oil amount is reduced over time.
- Case F: The fishing grounds are now placed upstream of the source point but still exposed while it drifts by. We can clearly see a rising oil amount until a top, which then descends.

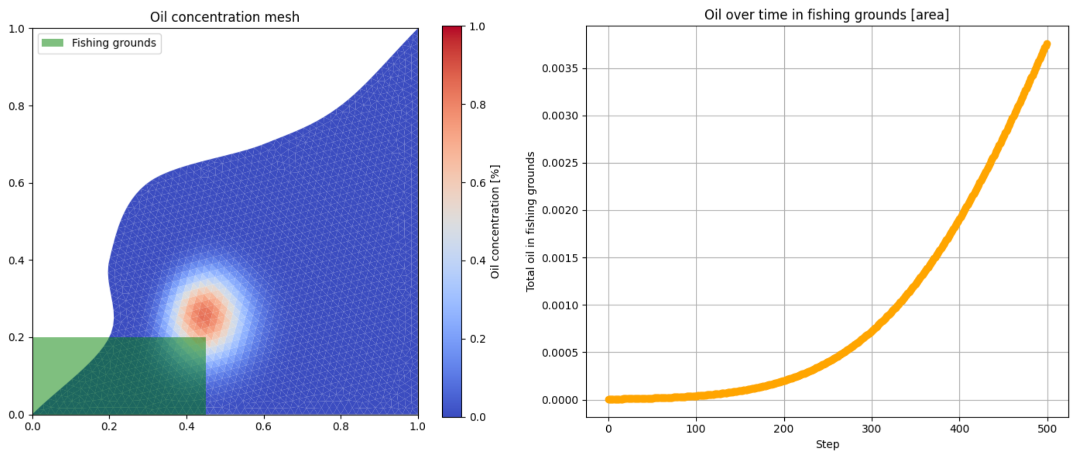


Figure 5.4: Case D, `borders`=[0.0, 0.45], [0.0, 0.2]

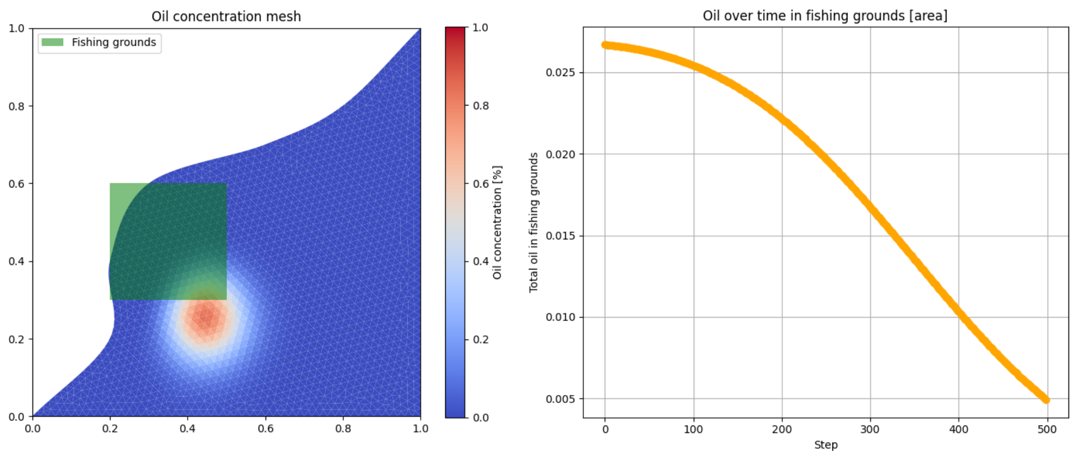


Figure 5.5: Case E, `borders`=[0.2, 0.5], [0.3, 0.6]

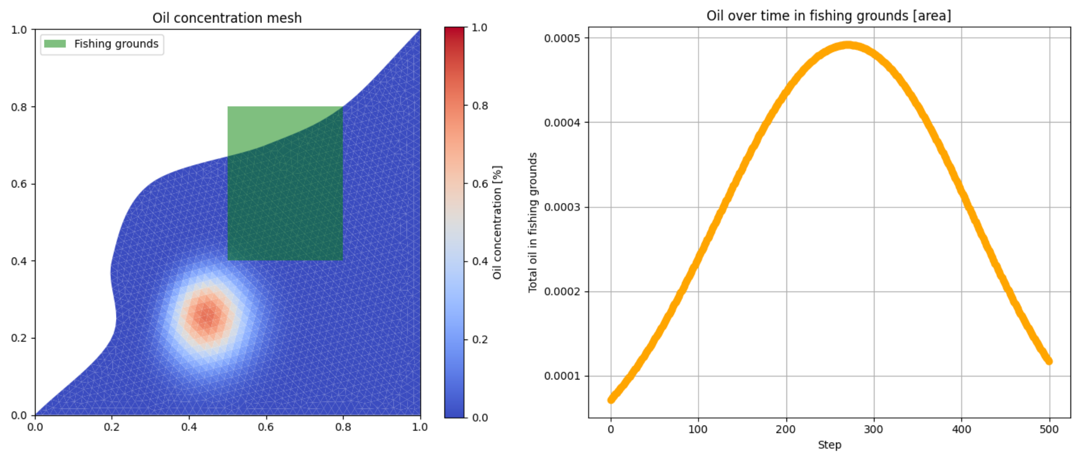


Figure 5.6: Case F, `borders`=[0.5, 0.8], [0.4, 0.8]

5.3 Boundary behavoiur (edge oil accumulation)

It is important to test the boundary behavoiur of the simulation because the bordering triangles does not act as normal triangles. They are not part of the flux calculation which will result in accumulation at the border, and without testing, this can be misinterpreted as realistic physical behaviour.

Case	`nSteps`	`tEnd`	`delta_t`	`meshName`	`borders`	`writeFreq`
G	500	2	0.004	`bay.msh`	[0.0, 0.45], [0.0, 0.2]	20

In Figure 5.7 one can observe that the oil is accumulating at the bottom border in a line of triangles with very high concentrations. The underlying vector flow of the ocean from Equation **??** is pushing the oil towards the border but can not cross it. The same phenomenon is visible in the total oil in the fishing grounds, keeping a constantly high amount of oil even after the oil should have passed.

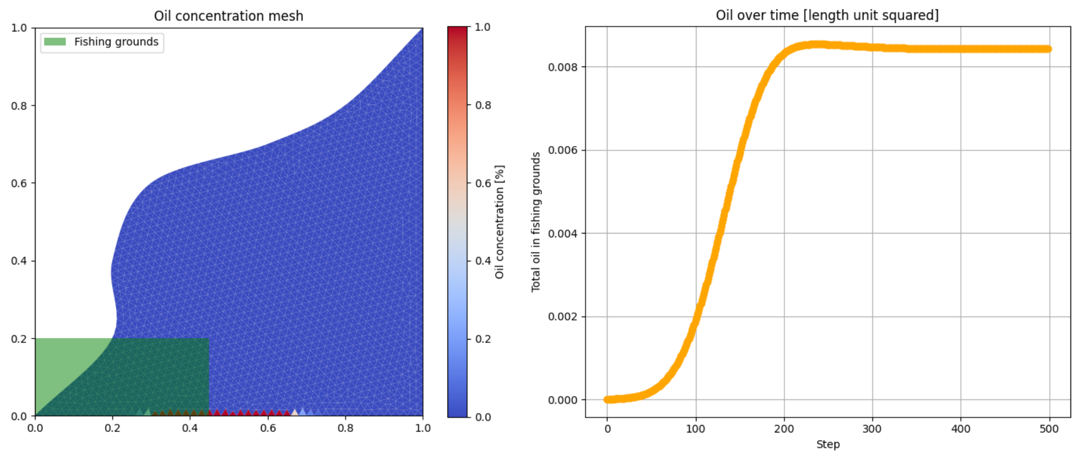


Figure 5.7: Case G, oil accumulation at bottom bordering triangles

5.4 Discussion

Both the sensitivity analyses underlines that the model is sensitive for both time steps and fishing grounds borders. Big `delta_t`'s creates diffusion or instability. Recalling from equation 3.3, we extract the ratio between $\frac{\Delta t}{A_i}$. If Δt is a relatively large number compared to A_i, the flux can potentially end up in unrealistic proportions.

In addition, different placements of the fishing grounds borders results in very different oil developments over time. These findings underlines that both the choice of `delta_t` and defintion of borders has to be carefully considered in order to achieve meaningful simulation results.

Furthermore, as can be seen in Figure 5.7 the oil accumulates in the triangles edging the border of the mesh. This is because line cells are at the border which by defintion can not contain oil. This is a intentional limitation of this model, and for future work it would be interesting to let the oil flow outside the borders of the mesh.

Chapter 6

Conclusion

This report has presented a complete numerical model for simulation oil transport through a triangular mesh, based on flux calculations and a predefined vector field in the ocean. Through modular program architecture, distinct division of responsibility between classes and intentional use of data encapsulation, a structured, extendable code has been developed ready for further use.

The results clearly shows that the model is sensitive for both time steps and geometrical delimation of the fishing grounds. For big time steps, diffusion and numerical instability, while small changes in the fishing ground borders can cause big impacts in total oil amount. These findings underlines the importance of choosing reasonable input parameters. In addition, the analysis shows that oil is accumulated against the mesh border, an intentional choice that should be adressed in further work if the oil should be allowed to leave the mesh.

In summary, the implementation fulfill the criteria in terms of functionality, data encapsulation and extendability, which is a solid foundation for further and more complex use.