Bathymetry Optimization in the Denmark Strait

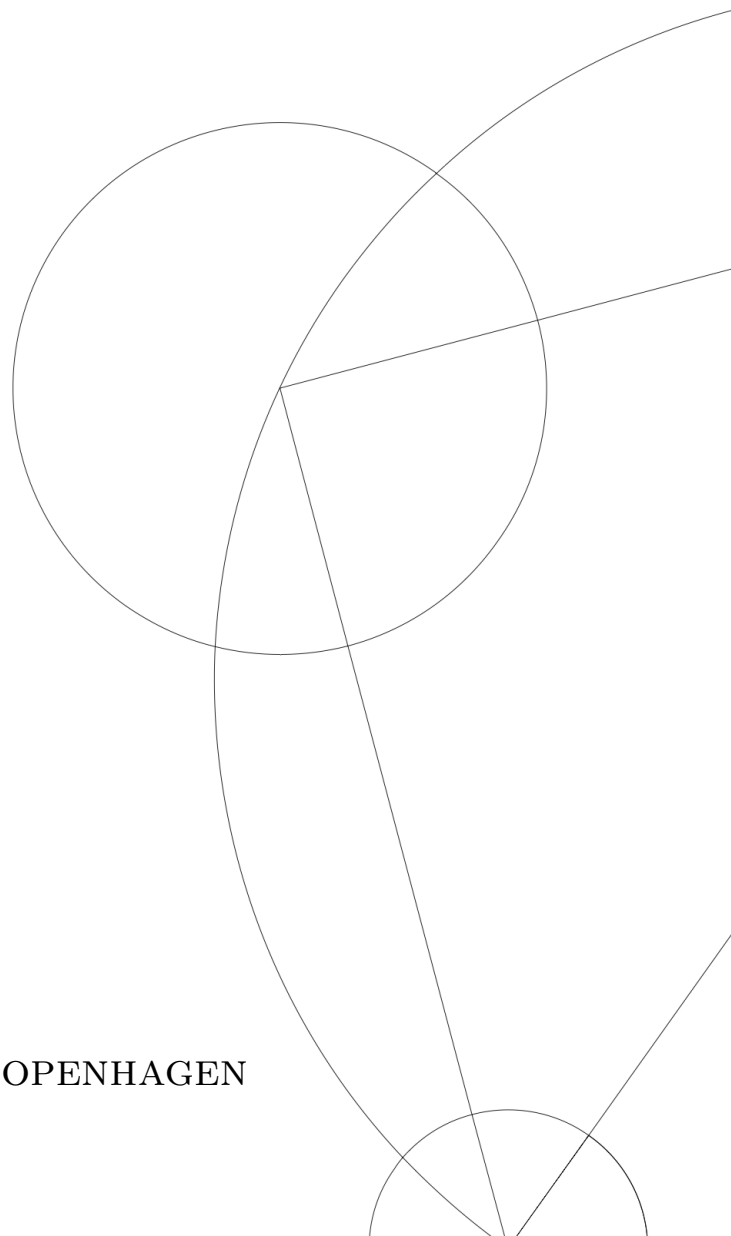
A Method for Topographic Tuning in Ocean Circulation Models

Bachelor Thesis

Written by *Nicholas O. Posborg*

June 3, 2025

Supervised by
Markus Jochum



University of Copenhagen

UNIVERSITY OF COPENHAGEN

Name of Institute:	Niels Bohr Institute
Name of Department:	Physics of Ice Climate and Earth (PICE)
Author(s):	Nicholas O. Posborg
Email:	zkg114@alumni.ku.dk
Title and subtitle:	Bathymetry Optimization in the Denmark Strait - A Method for Topographic Tuning in Ocean Circulation Models
Supervisor(s):	Markus Jochum
Handed in:	03.06.2025
Defended:	17.06.2025

Name _____

Signature _____

Date _____

Acknowledgments

I would like to thank my supervisor, Markus Jochum, for his guidance, support, and for giving me the opportunity to work on such an exciting and challenging project. His feedback and encouragement were crucial throughout this thesis.

A special thanks to Marta Agnieszka Mrozowska for her generous help with technical challenges and for introducing me to the VerOpt pipeline, which played a central role in the project. Her insight and patience were greatly appreciated.

Finally, I want to thank the rest of Team Ocean for creating a fun, collaborative, and inspiring environment to work in. It was a privilege to be part of such a fantastic group during my bachelor thesis.

Abstract

This thesis investigates how ocean bathymetry affects the Atlantic Meridional Overturning Circulation (AMOC), with a focus on the Denmark Strait. I used the Veros ocean model along with the VerOpt optimization tool to test how changes in the depth of the sill in the strait affect the overturning strength. A method was developed to change the bathymetry at specific grid points in Veros, while keeping the model stable. These modified bathymetries were then tested in a 30-year simulation, and the results were fed back into VerOpt to guide the optimization.

The results showed that changing the bathymetry does impact the AMOC strength, and that the method can automatically explore different setups and return meaningful results. Along the way, I ran into several issues, such as the fact that `kbot` values are integers (not continuous), smoothing reduces detail, and some values are repeated due to rounding. A particularly important limitation is that SLURM jobs that failed or stalled were treated as if the simulation had run and produced a bad result. This misrepresents the model behavior and can mislead the optimization. These problems are explained in the thesis, and I suggest possible improvements to address them in future work.

Finally we conclude that the project shows that it's possible to modify the bathymetry in Veros automatically and get different circulation outcomes. The method could be useful in other regions too, and is a step toward better tuning of ocean topography in climate models.

Contents

1 Introduction

The Atlantic Meridional Overturning Circulation (AMOC) is one of the major systems moving water and heat across the global ocean. It helps regulate the climate in the Northern Hemisphere by transporting warm water northward and returning cold, dense water south at depth. This circulation has a large influence on sea level, ecosystems, and even global shipping routes, so understanding it is key for both climate science and applied ocean modeling.

In models, capturing the structure and strength of the AMOC depends heavily on how well bathymetry is represented. Features like the Denmark Strait and Drake Passage are narrow and deep enough to strongly control where and how water can flow between ocean basins. During this project, I focused on the Denmark Strait because its sill controls the entry of dense water from the Nordic Seas into the Atlantic. To get realistic overturning behavior, I had to make sure the model preserved the sharp bathymetry gradients in that region.

In this thesis, I develop a method for modifying ocean bathymetry in the Veros model using the VerOpt optimization framework. The goal is to adjust the bathymetry in a physically meaningful way to better match observed overturning strength, particularly in regions where overflow processes dominate. This work also provides insight into the challenges and limitations of such an approach, and what it takes to make optimization-based bathymetry inversion a viable tool for improving large-scale ocean simulations.

To evaluate whether the method is successful, two main criteria are defined. First, the framework must be able to modify bathymetry in a controlled and automated way without compromising numerical stability in Veros. Second, the modified bathymetry should lead to measurable differences in the simulated overturning circulation, ideally bringing the model's AMOC strength closer to observed values. A successful outcome therefore involves both technical implementation and a physically meaningful model response.

2 The AMOC and The Denmark Strait

2.1 The AMOC

The Atlantic Meridional Overturning Circulation (AMOC) is one of the most important parts of the global ocean circulation. It consists of large-scale ocean currents that move heat and water between different regions, especially in the Atlantic. At the surface, warm and salty water travels northward with the Gulf Stream. When it reaches the colder regions of the North Atlantic and the Nordic Seas, the water cools, becomes denser, and sinks. This cold, dense water then flows back southward at depth, forming the lower branch of the circulation. The AMOC is essential for transporting heat from the tropics

to higher latitudes and helps regulate the climate in places like Northern Europe [3, 5].

2.2 The Denmark Strait

The Denmark Strait is one of the key locations in the AMOC system. It lies between Greenland and Iceland and is relatively narrow, only a few hundred kilometers wide. What makes it especially important is its shallow depth compared to the deep North Atlantic basin nearby. In the Denmark Strait, warm surface water moves north, while cold, dense water flows southward at depth. The sharp change in bathymetry at this location acts as a threshold that dense water must cross to reach the Atlantic. Because of this, the Denmark Strait controls how much overflow enters the AMOC [3, 6], and even small changes in its depth or shape can strongly affect the circulation strength.

3 Veros and Veropt

3.1 Veros

Veros (The Versatile Ocean Simulator) [4] is an open-source ocean model written in Python, which makes it much easier to modify than traditional Fortran-based models. It also supports both CPU and GPU execution through JAX [2], which helped a lot during this project, since I had to run many simulations back-to-back as part of the optimization loop.

Veros uses an Arakawa C-grid layout, where variables like velocity, temperature, and salinity are stored at staggered positions. This improves numerical accuracy, especially for momentum and tracer transport, and helps reduce numerical noise. The model comes with a set of predefined setups that can be adjusted, which I made use of.

In my case, I used the *Global Flexible Resolution* configuration. This setup allows finer grid spacing in specific areas. I also modified the bathymetry input file to preserve key sills and deep channels that are important for controlling dense overflow.

3.2 Veropt

Veropt [7] is a Python-based optimization framework developed specifically for use with Veros. It's still in development and not yet publicly released, so I worked with an internal version shared by the developers. I had to adapt some parts of it to fit my experiment, mainly to allow custom bathymetry input and to properly extract the AMOC strength after each Veros run.

The optimization uses a Bayesian approach. It starts by testing a few randomly chosen bathymetry values, runs the simulations, and then compares the results. In my case, the evaluation was based on how close the simulated AMOC strength came to the

observed target. Based on this, Veropt builds a surrogate model and chooses new values to test that are likely to improve performance.

I used Veropt to optimize the bathymetry by changing `kbot` values at seven selected grid points in the Denmark Strait. For each suggested modification, I ran a full Veros simulation and calculated the overturning strength from the output. The goal was to find a configuration that results in an AMOC close to 17.6 Sv, which is a typical observed value from the RAPID array at 26.5°N [8].

4 The Challenge

The grid-based structure in Veros helps simplify the ocean and makes it easier to run simulations on a computer. However, this simplification comes at the cost of losing fine details, like the sharp bathymetric features in the Denmark Strait. The challenge is how we can modify the bathymetry to better match the real shape of the strait, while still keeping the model stable and computationally efficient.

If we can successfully do this in Veros, we can treat bathymetry as a parameter and use VerOpt to explore different configurations automatically. This would allow us to test which bathymetric setup gives the most realistic overturning circulation. And it's not just limited to the Denmark Strait, this same method could be applied to other important regions too.

The main goal is to create a method for modifying the bathymetry at selected grid points, so that the simulation comes closer to how the AMOC works in the real world. If this is done right, models like Veros can better capture overflow dynamics and large-scale circulation patterns.

5 Method

5.1 Bathymetry Representation and Modification

The first step is to run Veros with the original ETOPO5 dataset [1], which contains global ocean depth information. This data is used to initialize the model and produce the bathymetry that Veros uses internally. Veros then converts the depth values into `kbot` indices, where `kbot` $= 1$ is the deepest and `kbot` $= 60$ is the shallowest. Grid points with `kbot` $= 0$ are treated as land. The depths range from around 4 meters (shallowest) to 5400 meters (deepest). The spacing between vertical levels is nonlinear and increases with depth, but Veros handles this internally. After this, Veros applies a smoothing step, which is described next.

Figure 1 shows the global ETOPO5 bathymetry. Figure 2 shows the same field after Veros has processed it, including smoothing and vertical indexing.

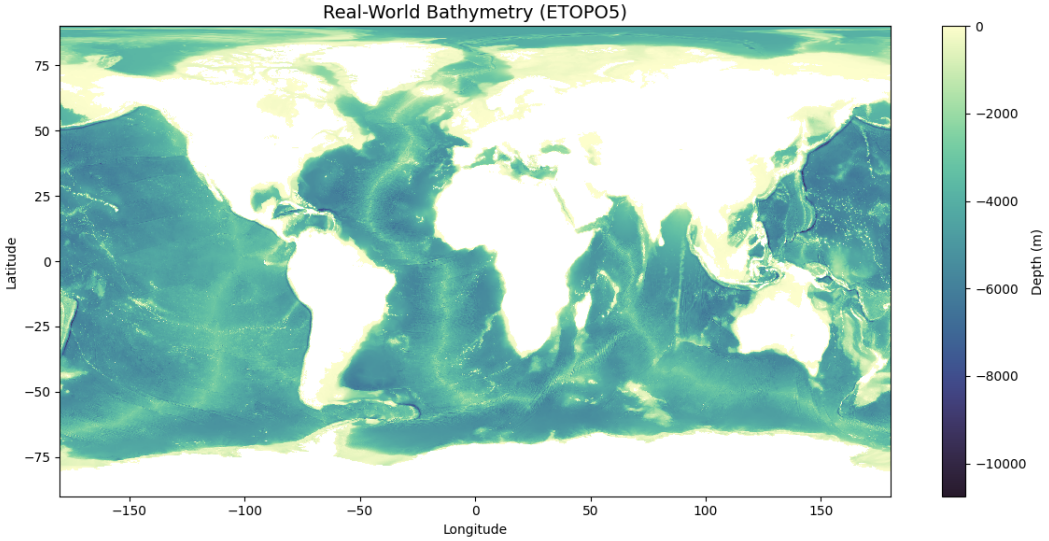


Figure 1: Global bathymetry from the ETOPO5 dataset, used as the reference depth input. The colormap uses darker shades for deeper regions.

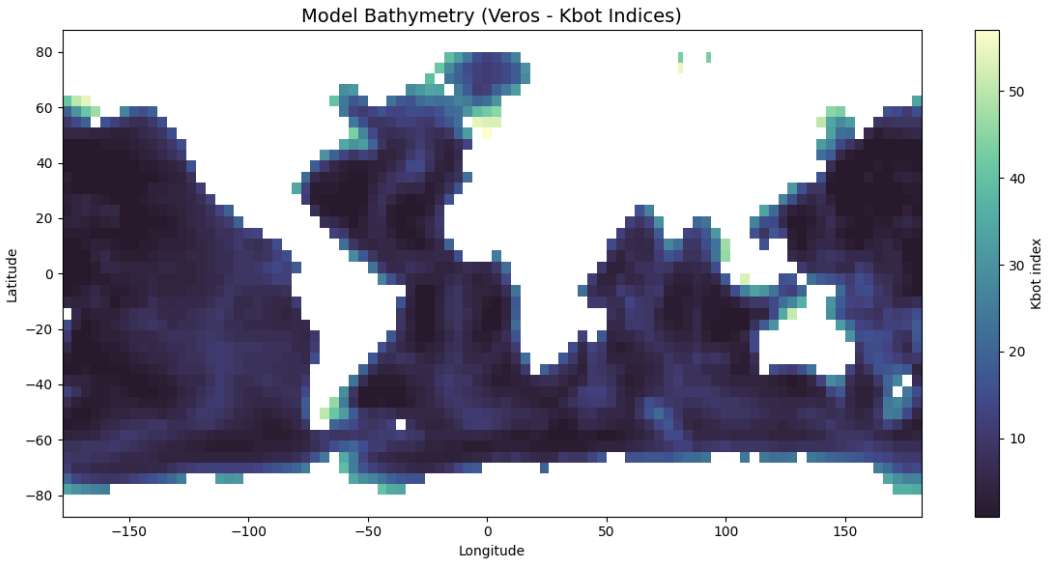


Figure 2: Model bathymetry used by Veros, expressed in vertical layer indices (`kbot`). Lower values correspond to deeper ocean points. The resolution reduction and smoothing effects are clearly visible compared to the ETOPO5 dataset.

The Veros bathymetry is then saved so it can be modified during the optimization loop. At this stage, I also choose which grid points to change. In this project, all selected

grid points are assigned the same `kbot` value for each simulation.

5.2 Smoothing Procedure

To maintain numerical stability and avoid excessively steep depth gradients, Veros applies two types of smoothing:

- First, the general Gaussian smoothing is applied to the ETOPO5 depth field before converting to `kbot`. This helps reduce sharp depth differences and improves numerical stability.
- After inserting the modified `kbot` values at the chosen points, I apply a local Gaussian smoothing with a standard deviation of $\sigma = 1$ in a 5×5 region around each point. This helps prevent crashes and allows Veros to handle a wider range of bathymetric configurations.

A localized Gaussian smoothing was chosen because the global Gaussian smoothing already used in Veros proved effective for the ETOPO5 data. By applying a similar method locally around the modified points, I aimed to maintain consistency and leverage a technique known to work well in the model.

After these steps, the model is ready to run.

5.3 Simulation and Diagnostics

Veros runs the simulation in two cycles. Each cycle covers half the total simulation time. For example, for a 30-year simulation, each cycle lasts 15 years. At the end of the run, Veros outputs an overturning file. If this file is missing, it means the model did not converge and failed early, usually around iteration 12 to 17. Failed runs are assigned a high penalty of 10^6 in the objective function.

For the successful runs, I extract the AMOC strength from the last timestep of the overturning output. The overturning is evaluated between 24°N and 34°N in latitude and from 500 m to 2000 m in depth, where the AMOC is strongest and best observed. One note is that Veros uses a different sign convention for the overturning, so the AMOC appears as a negative value in this model.

5.4 Objective Function

Now that I have the AMOC strength from the model, I can compare it to the target value. The objective function is designed to return a score that gets closer to zero as the simulation result gets closer to the target. A small amount of tolerance is built in to reduce the effect of noise. The goal is to reward parameter values that lead to more realistic simulations.

$$J(\theta) = -\left(\frac{\text{model AMOC} - \text{target AMOC}}{\text{target AMOC}}\right)^2$$

5.5 Optimization Workflow

The optimization is done using VerOpt, which applies Bayesian optimization. The steps are:

1. VerOpt suggests new `kbot` values for the selected grid points.
2. The modified bathymetry is saved.
3. A Veros simulation is started using the new bathymetry.
4. The AMOC strength is extracted and used to compute the objective value.
5. The result is logged and used to update the surrogate model.

This process repeats for a fixed number of steps. If a simulation crashes, the associated parameter is penalized. This also means stalled runs or runs that were never submitted to the cluster were penalized. Figure 3 shows the general VerOpt process from Mrozowska et al. (2025) [7], while Figure 4 shows how I implemented it in this project.

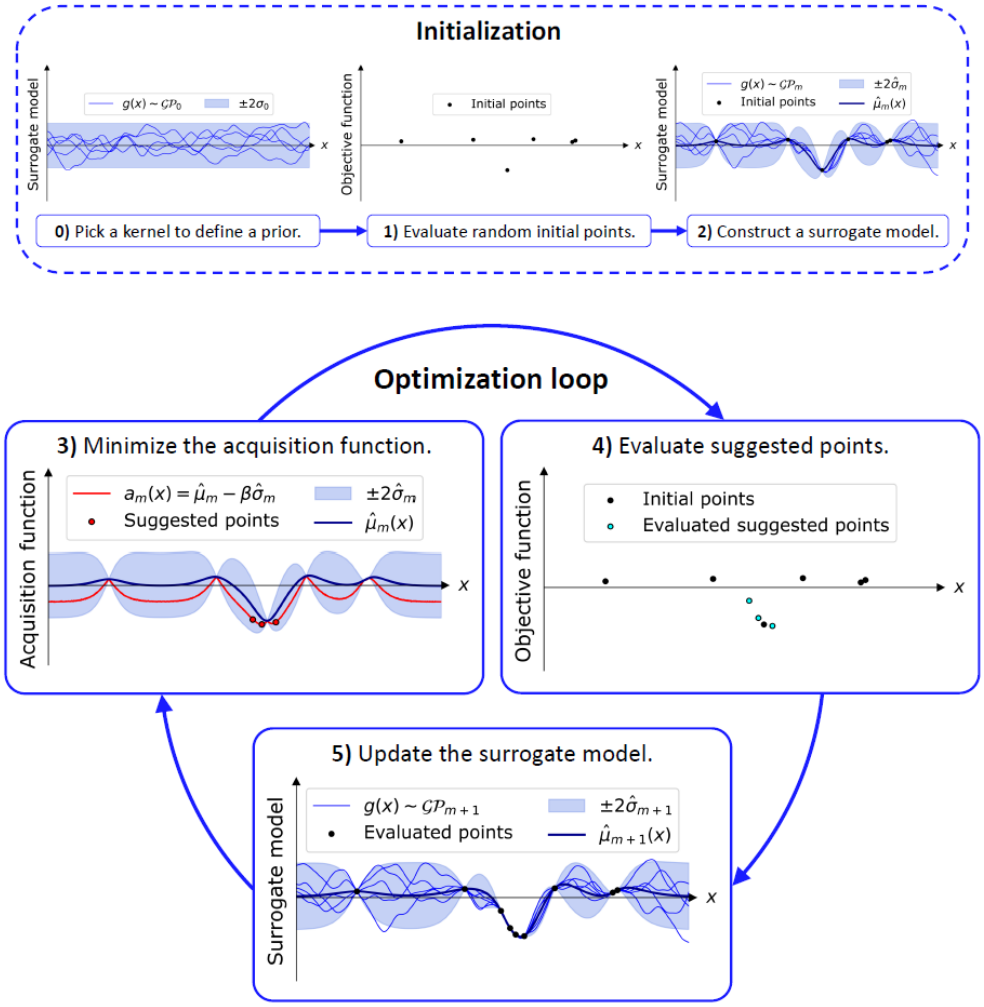


Figure 3: Bayesian optimization workflow used in VerOpt. The loop consists of (1) choosing random initial points, (2) constructing a surrogate model based on initial evaluations, (3) proposing new candidates by minimizing an acquisition function, (4) evaluating them with full Veros simulations, and (5) updating the surrogate model with new results. Optimization loop used in this project, taken from Mrozowska et al. (2025) [7].

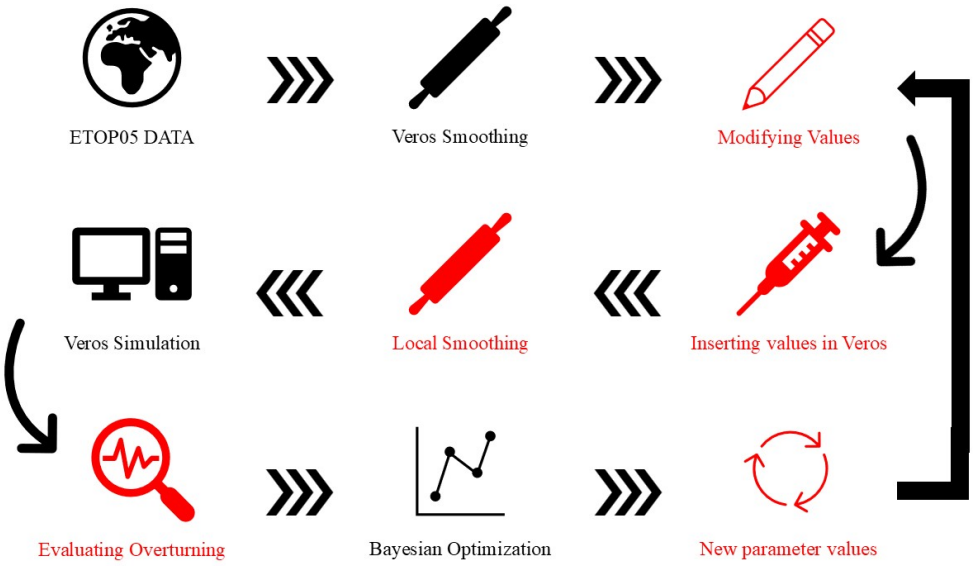


Figure 4: Overview of the custom optimization pipeline. ETOPO5 bathymetry is smoothed and modified at selected grid points before being injected into the Veros model. The simulation is then run, overturning strength is evaluated, and the objective is returned to the VerOpt optimizer, which proposes the next parameter. The red steps in the figure show the parts I worked on directly.

6 Results

The AMOC optimization experiment was conducted in three rounds: four initial random samples, followed by two Bayesian optimization steps with four simulations each. Every simulation was run for 30 years, split into two 15-year cycles.

The bathymetry was modified at seven selected coordinates in the Denmark Strait (Figure 5), chosen to control the volume of dense water entering the Atlantic and affecting the overturning circulation. These points were selected because they regulate the flow from the Nordic Seas into the Atlantic. In each simulation, the same `kbot` value was assigned to all locations, shown as red dots.

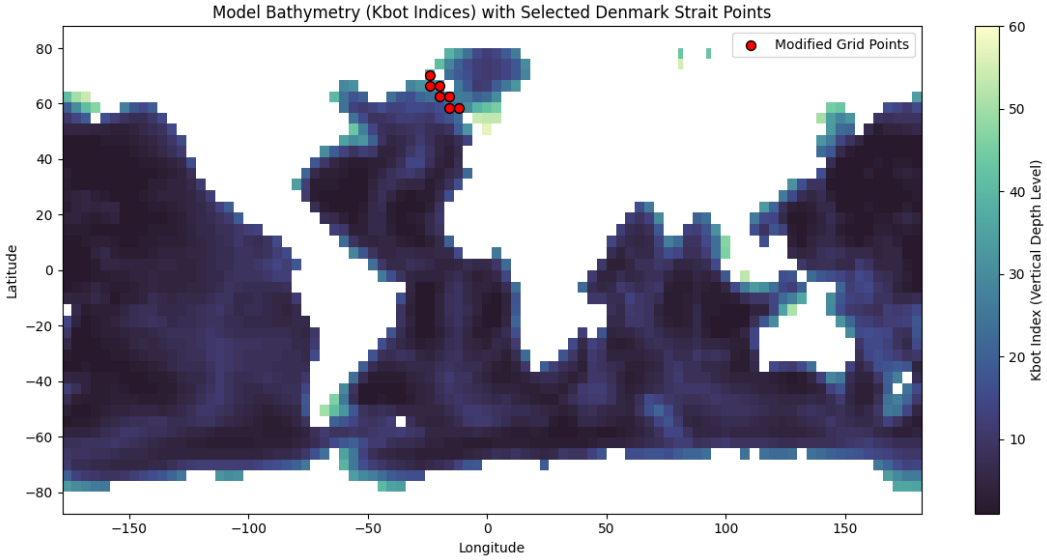


Figure 5: Coordinates in the Denmark Strait where bathymetry was modified. Red dots indicate grid cells altered during the optimization.

In total, 12 `kbot` values were tested. Some values were repeated due to resampling, and a few simulations failed either because the `kbot` was too deep, causing the model to become numerically unstable and not converge, or because the job stalled on the Aegir cluster. When a simulation failed, I assigned it a penalty value of 10^6 so the optimizer could proceed. Table 1 gives an overview of all tested values and their outcomes.

Table 1: Summary of all `kbot` values tested during optimization. Simulations are grouped by stage: initial random samples, first and second Bayesian optimization steps. Failures were assigned an objective penalty of 10^6.

#	`kbot`	Objective	Iteration	Years	Status
Initial random samples					
1	22	-0.019	5475	30.42	Completed
2	1	10^6	13	0.04	Crashed (unstable)
3	7	10^6	15	0.04	Crashed (unstable)
4	26	10^6	5475	15.21	Stalled (1 cycle)
Bayesian optimization step 1					
5	45	10^6	5475	15.21	Stalled (1 cycle)
6	27	-0.046	5475	30.42	Completed
7	55	-0.145	5475	30.42	Completed
8	8	10^6	14	0.04	Crashed (unstable)
Bayesian optimization step 2					
9	34	-0.108	5475	30.42	Completed
10	57	-0.148	5475	30.42	Completed
11	1	10^6	—	—	Repeat
12	45	10^6	—	—	Repeat

The method successfully inserted each `kbot` value into the model bathymetry and applied local smoothing to make sure it stayed stable. This preprocessing usually worked, except in cases with very deep depths that made the model crash. Figure 6 shows the final bathymetry fields, where you can clearly see the modified points in the Denmark Strait.

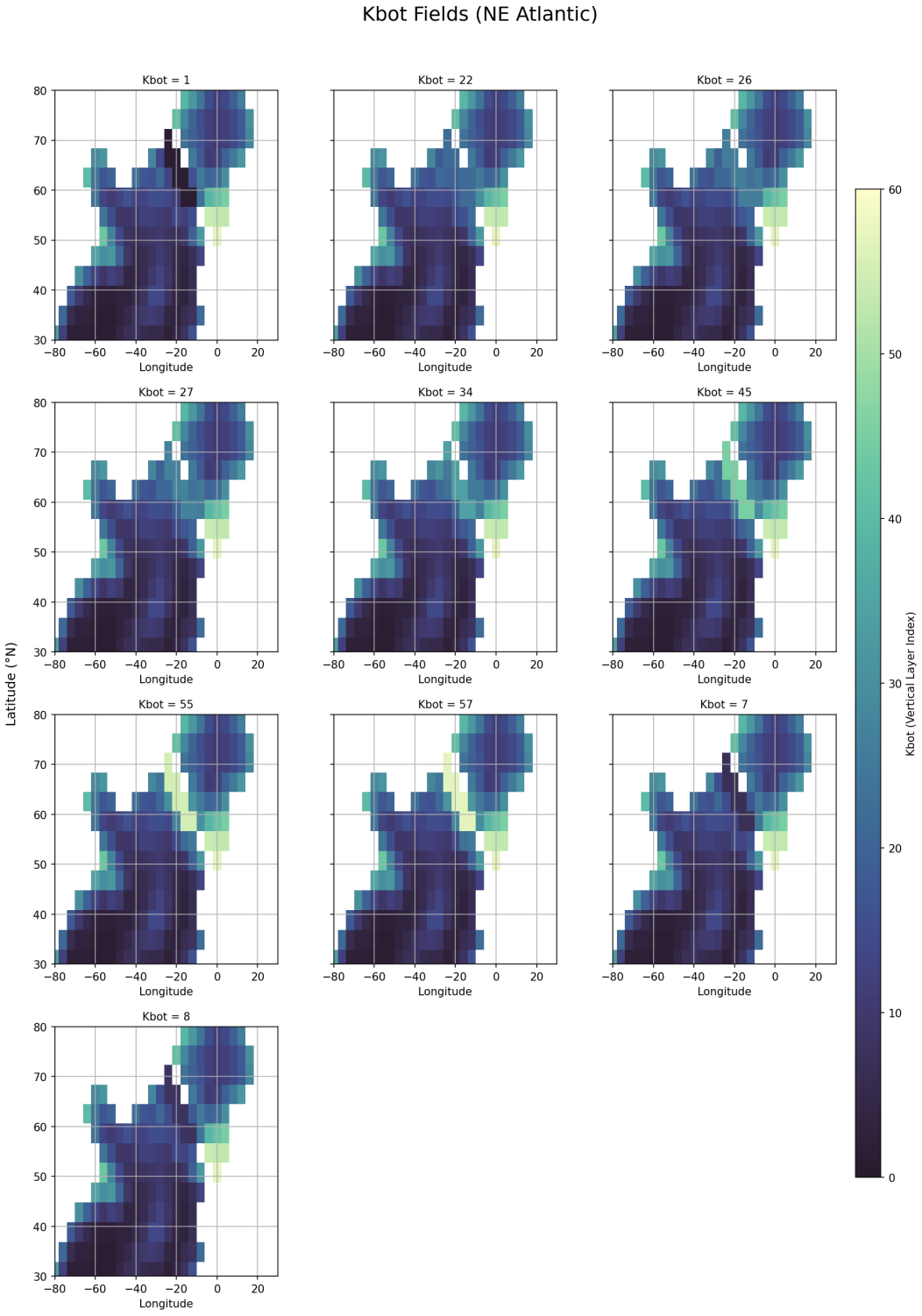


Figure 6: Final bathymetry field in terms of `kbot` layer indices. Lighter values represent shallower depths. Modified grid points are clearly visible in the Denmark Strait.

Figures 7 and 8 show how each bathymetry setup affected the Atlantic Meridional Overturning Circulation (AMOC). Each subplot shows the overturning streamfunction (`vsf_depth`) from the last timestep of the run. In Veros, positive values mean counter-clockwise circulation (northward flow in the deep ocean), while negative values mean clockwise flow like in the AMOC. The 15-year figure includes all simulations that reached the end of the first cycle, even those that later stalled or crashed. The 30-year figure only shows the simulations that completed both cycles and reached the full 30 years.

From the figures, we can see a clear difference in the AMOC strength between `kbot` 22 and `kbot` 55. The deeper blue and larger area for `kbot` 22 indicate a stronger southward flow. Since lower `kbot` values represent deeper bathymetry, this shows that deeper configurations allow more dense water to overflow the sill, enhancing the overturning circulation, especially between 0° and 60°N.

As expected from overflow theory, I saw that a deeper sill allowed denser water to pass, strengthening the overturning cell. A shallow sill acts like a bottleneck and limits the flow, which weakens the entire loop. This shows how sensitive the AMOC is to bathymetric changes, and why it's important to represent overflow regions like the Denmark Strait correctly in ocean models.

It's also worth noting that simulations for `kbot` = 26 and 45 stopped after the first 15-year cycle and didn't finish. This wasn't due to instability, but because the jobs stalled on the Aegir cluster, for reasons I couldn't fully determine.

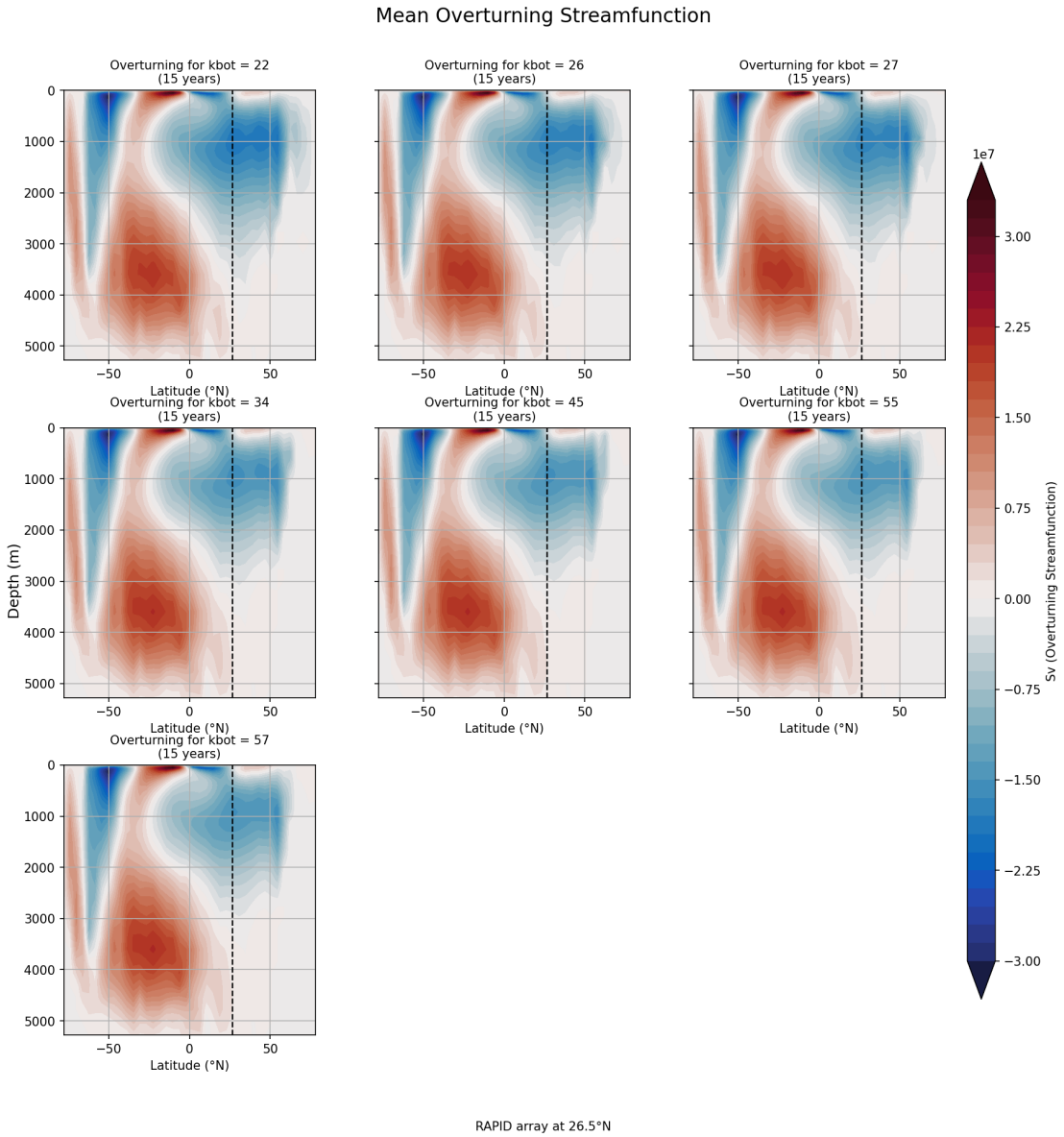


Figure 7: Overturning streamfunction (`vsf_depth`) at the end of the first 15-year cycle for all simulations that made it that far. These early profiles show initial differences in the AMOC across different `kbot` configurations. The vertical dashed line marks the RAPID array at 26.5°N [8].

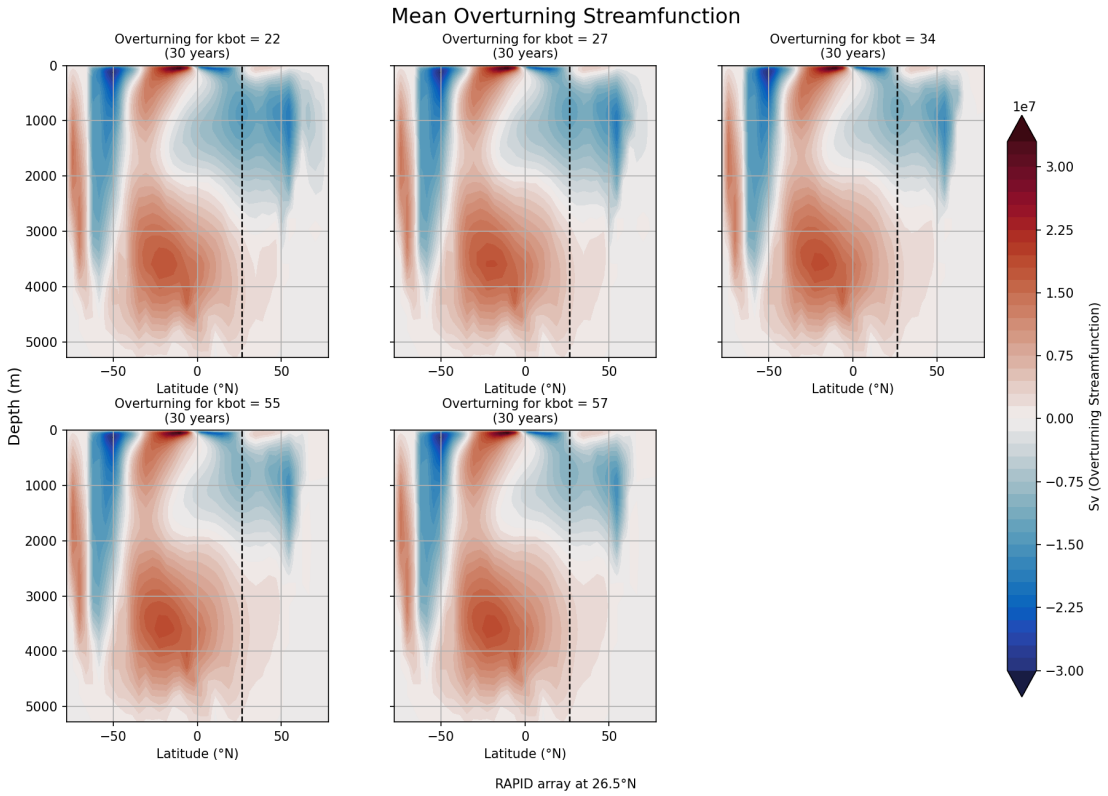


Figure 8: Overturning streamfunction (`vsf_depth`) at year 30 for simulations that completed both cycles. These show the more developed AMOC signal. The vertical dashed line marks the RAPID array at 26.5°N [8].

Overall, the optimization found bathymetry values that got the AMOC strength closer to the real-world target of 17.6 Sv. The best-performing cases were `kbot` = 22 and 27, which gave the lowest objective values. This supports that bottom topography has a big influence on large-scale circulation.

Even though each simulation was only 30 years, that was enough to see clear differences in AMOC strength and structure across the tested values. I didn't run longer simulations because of time constraints and SLURM issues, but since the goal was to test the method rather than reach perfect equilibrium, the 30-year runs were good enough for that.

7 Limitations

7.1 Smoothing Constraints

To keep the simulation stable, Veros applies Gaussian smoothing to the bathymetry. This helps prevent sharp depth changes, but it also means that small-scale features like the Denmark Strait sill become less accurate. This is especially a problem when using low-resolution models, where even small amounts of smoothing can change the shape of the strait quite a bit.

In my setup, Veros does global smoothing first, and then I apply a local smoothing around each modified `kbot` point. This is done no matter if the point is isolated or next to other points. The goal is to smooth the surrounding cells just enough so Veros doesn't crash. But this also makes it harder to test steep or very different depths, because the smoothing limits how sharp the bathymetry can be.

7.2 Non-Differentiable Step Function

Since bathymetry in Veros is controlled through the `kbot` field, which is always an integer, the parameter space becomes a step function. This is a problem for optimizers like VerOpt that are made to work with continuous parameters. VerOpt tries to improve by assuming smooth changes, but that doesn't really work when you're jumping between integers.

7.3 Parameter Rounding

Even though VerOpt suggests float values, they are always rounded to the nearest integer when applied. This means that different suggestions can end up using the same value. So you can waste simulation time testing the same bathymetry more than once, which slows things down and doesn't help the optimizer learn anything new. They will also overwrite the same folder, so that's another issue that needs to be handled.

7.4 Single Shared Parameter Across All Points

In this project, I kept it simple and used the same `kbot` value for all seven points. This made the problem easier to manage, but also made the solution space much smaller. Ideally, I would want each point to be an independent parameter. That way, VerOpt could find more complex structures, like sloped sills, instead of just flat ones.

But letting each point have its own value creates another problem: if nearby points have `kbot` values that are too far apart, the model might crash. Veros doesn't like steep jumps in bathymetry. So even if we let all the points vary freely, we would still need a way to stop the optimizer from trying values that are too different. This could be done

with some kind of penalty for sharp differences, or with a smarter optimizer that knows to avoid unstable setups.

7.5 Initial Conditions

Running Veros takes time, so it's important to think about initial conditions before starting. Each time we test a new parameter, the whole simulation has to run. That's why I did some early tests to see which `kbot` values were clearly too shallow or too deep, they crashed right away. I still kept them in my VerOpt setup because I needed to see if it could handle them properly. If you know when they crash, you can set the bounds there and save time and resources.

Another important thing is checking that the objective function actually changes based on the bathymetry. If it doesn't respond, then the optimizer can't do much. I also had to make sure the simulation was long enough to show differences in the AMOC, but not so long that it took forever to run. In the end, I settled on 30 years, which worked for my goals and available time.

7.6 SLURM Jobs

Sometimes the SLURM jobs didn't submit properly or just stalled during a run. To work around this, I added a timer: if a job didn't finish after a certain amount of time, it was canceled and given a penalty in the objective function. This allowed VerOpt to continue, but also meant that some valid simulations might have been penalized unfairly, not because the bathymetry was invalid, but because of technical issues on the cluster. This is a significant limitation, because the optimizer treats these penalty values as real signals when learning, which can distort the optimization process and steer it away from potentially good solutions. In that sense, it breaks the idea of the objective function reflecting actual model performance. Still, with limited time and recurring system instability, this was the only viable way to run a full optimization loop and get results from the method.

7.7 Numerical Stability and Runtime

All simulations were run on the Aegir cluster with one CPU core using a $4° \times 4°$ Veros setup. With that setup, one year takes about 5 minutes to simulate, so a 30-year run finishes in a little over two hours. That made it doable to run several simulations in a row.

But even with smoothing, not all bathymetry values worked. If I wanted to use a higher-resolution grid or longer simulations, this setup would start to take too long. Then I'd probably need to run on more cores or find ways to speed things up, otherwise it wouldn't be practical.

8 Outlook

To improve this framework going forward, a lot of the limitations I talked about earlier need to be looked at more closely. Each one involves a trade-off between how realistic the model is, how stable it runs, and how long it takes to simulate. Getting that balance right is important for figuring out what kind of setup is best for future experiments.

In this project, I focused on changing the depth through the `kbot` field, but another thing to look into would be the land–ocean boundaries. For example, the Denmark Strait or the Drake Passage are very narrow in reality, and they might be too wide or too smooth in the model. If I could adjust the land mask or block off certain grid cells, I might be able to represent these regions better. That could help test whether the shape of the channel, not just its depth, has a big effect on the overflow and the AMOC.

One thing that clearly needs improvement is how SLURM jobs are handled. In my current setup, sometimes a simulation would stall or not submit correctly, and VerOpt would treat it as if the bathymetry was invalid. That's not ideal. A better system would be able to check if a job actually failed because of the model or just because the cluster had issues. That way, we wouldn't throw away good parameter values by mistake.

Another thing worth testing is longer simulation time. Right now, I'm using 30-year runs, which are enough to see some differences in the overturning, but maybe not enough to reach full equilibrium. Running the simulations for longer could make the AMOC signal more stable and reliable. The downside is that it would take much more time to run each one. So if I want to go in that direction, I'll probably need to either use more cores or find other ways to speed things up.

Overall, if I can improve the job handling, test different bathymetry shapes, and maybe scale up the runtime and resolution, then this method could be used in other parts of the ocean as well.

9 Conclusion

The goal of this project was to develop a method for optimizing the bathymetry in the Denmark Strait to get a better AMOC signal in Veros. The method uses Veros' internal smoothing of the ETOPO5 bathymetry, and then applies targeted changes at specific grid points. To make sure the model stays stable, I added extra smoothing locally around those points. Once the bathymetry is modified, the files are passed into VerOpt, which tests different parameter values and tries to get the overturning strength closer to the real-world value.

The 30-year simulations showed that the method works, the bathymetry was changed as expected, and the AMOC strength clearly changed from one configuration to another. From the results, it's clear that changing bathymetry even at just seven points made

a measurable difference in AMOC strength, confirming that Veros reacts sensitively to these adjustments. It also confirmed that the depth of the sill in the Denmark Strait matters a lot when it comes to how much dense water flows into the Atlantic.

That said, there are a few things that really affect how well the method works. The initial conditions, the choice of which points to modify, the way floats are rounded to integers, and how smoothing is applied, all of these play a role in whether the model stays stable and whether the optimizer makes progress. And since VerOpt was made for smooth, continuous parameters, it struggles a bit with the step-like nature of `kbot` values. It would be better if each point could be optimized separately, but that also brings a higher risk of instability. That's something to look into in future work.

Another big issue was SLURM jobs failing or stalling. Sometimes I couldn't tell if the problem was with the bathymetry or just the cluster, and that made it hard to know if a simulation should be penalized or not.

Also, if I were to run longer simulations or increase the resolution, I'd need more computational power and maybe a better setup to keep the model from crashing. But overall, the method worked, and it could definitely be used in other places where bathymetry plays a big role in the ocean flow. It's a solid starting point for using optimization to fine-tune the ocean bottom in models like Veros.

References

[1] Etopo5 global relief model. `https://pubs.usgs.gov/of/1998/of98-801/bathy/metadata/etopo5.htm`, 1988. Accessed April 2025.

[2] James Bradbury, Roy Frostig, Peter Hawkins, Matthew Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: Composable transformations of Python+NumPy programs. `https://github.com/google/jax`, 2018. Accessed April 2025.

[3] Gokhan Danabasoglu, William G. Large, and Bruce P. Briegleb. Climate impacts of parameterized Nordic Sea overflows. *Journal of Geophysical Research: Oceans*, 115, 2010.

[4] Dion Häfner, René Løwe Jacobsen, Carsten Eden, Mads R. B. Kristensen, Markus Jochum, Roman Nuterman, and Brian Vinter. Veros v0.1 – a fast and versatile ocean simulator in pure Python. *Geoscientific Model Development*, 11(8):3299–3312, 2018.

[5] T. Kuhlbrodt, A. Griesel, M. Montoya, A. Levermann, M. Hofmann, and S. Rahmstorf. On the driving processes of the Atlantic Meridional Overturning Circulation. *Reviews of Geophysics*, 45(2), 2007.

[6] Martin Losch and Carl Wunsch. Bottom topography as a control variable in an ocean model. *Journal of Atmospheric and Oceanic Technology*, 20(11):1683–1696, 2003.

[7] Marta Agnieszka Mrozowska, James Avery, Aster Stoustrup, Roman Nuterman, Carl-Johannes Johnsen, and Markus Jochum. Fast and efficient: Bayesian optimization with gpu acceleration for ocean models. Manuscript, available from the authors, 2025.

[8] David A. Smeed, Gerard D. McCarthy, Stuart A. Cunningham, Eleanor Frajka-Williams, Darren Rayner, William E. Johns, Christopher S. Meinen, Molly O. Baringer, and Harry L. Bryden. Observed decline of the Atlantic Meridional Overturning Circulation 2004–2012. *Ocean Science*, 10(1):29–38, 2014.

Appendix

My GitHub repository with all relevant code: `https://github.com/DJNOP/Bachelor-project`

Objective Function

This script defines the `OverflowObjFun` class used by VerOpt. It modifies the `kbot` bathymetry values in a NetCDF file at selected coordinates, submits Veros simulations via SLURM, and evaluates the resulting AMOC strength from Veros output. The AMOC is compared to a target value using a relative squared error, which serves as the optimization objective.

Veros Setup Script

This is the modified Veros setup based on the `global_flexible` template. I extended the `set_topography()` method to inject modified bathymetry from a NetCDF file and apply localized smoothing. I also added `load_and_sort_bathymetry()` to preprocess external bathymetry files.

Run Script

This script runs the full optimization using VerOpt. It defines the optimization parameter (`kbot`), configures the number of evaluation points, and iteratively runs the Bayesian optimization loop, calling the objective function and saving optimizer states.

AI-Declaration

Declaration on Use of Generative AI Tools (Student)

✓ I/we have used generative AI as a tool (tick the box)
❑ I/we have NOT used generative AI as a tool (tick the box)

List which GAI tools were used, including a link to the platform (if possible):
ChatGPT (https://chat.openai.com)

Describe how generative AI was used in the assignment:

1) **Purpose (what did you use the tool for):**
   I used ChatGPT as a coding assistant to help understand the Veros, VerOpt, and SLURM tools (including their structure and specific code segments), to debug Python code, and to explore ways of organizing my optimization method.

2) **Work phase (when during the work process did you use GAI):**
   During both the development and coding phases, particularly while building, improving, and troubleshooting the simulation pipeline.

3) **What did you do with the output:**
   All code was tested and adjusted by me. ChatGPT suggestions were used as guidance and always reviewed and implemented manually. I interpreted all simulation results and included only my own analyses in the assignment.

NB: GAI-generated content used as a source in the thesis requires proper quotation and referencing. See KU Library guidelines on KUnet.