Report VSMN20

Isabella McAuley Skriver
20020123-5801

May 22, 2025

1 Introduction

1.1 Problem Domain

The program is to be used for calculating the groundwater flow of a certain domain with a barrier placed in the center. This domain is created by the user by choosing values for different parameters including lengths, pressure and permeability. The following figure illustrates an example of a given domain with marked width, height, depth and thickness. In the figure the green marks the domain, the gray marks the barrier and the blue marks the predetermined pressure placed along a boundary.

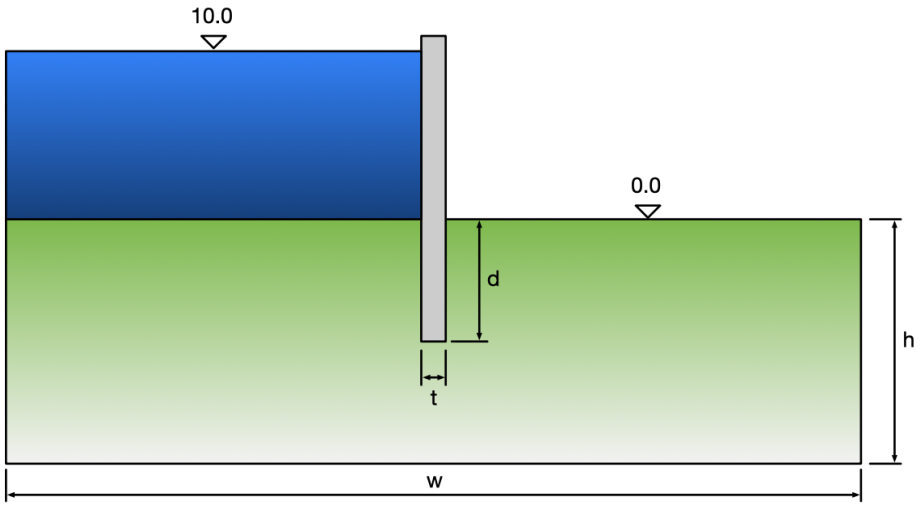


Figure 1: Example of the problem domain.

1.2 Theory

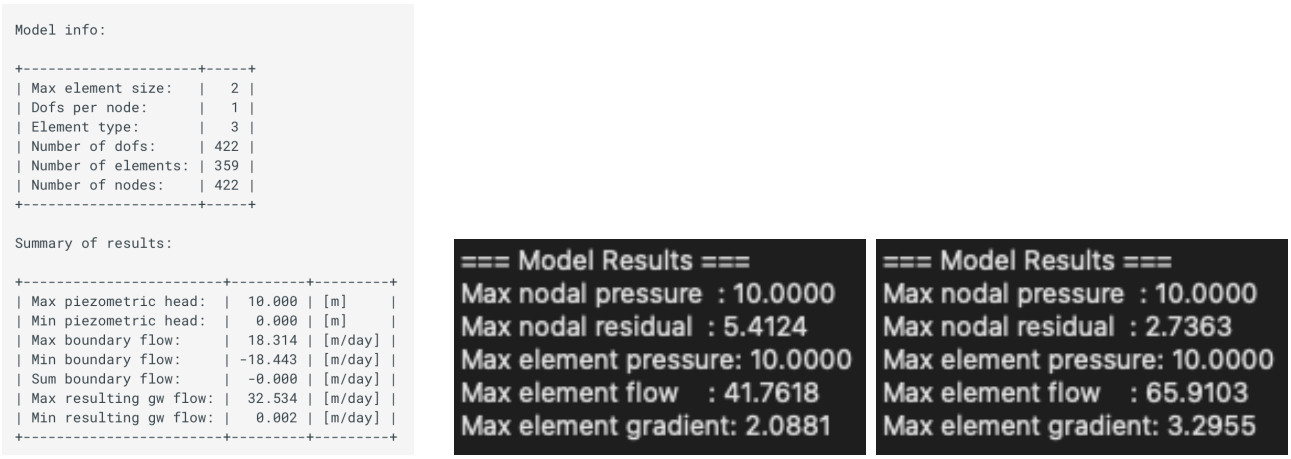
The problem at hand is computationally solved using the Finite Element Method along side the CALFEM toolbox. Initially a mesh based on the given domain is defined, the predetermined pressure and flow within the domain is set, these are also referred to as the natural and essential boundary conditions, so that finally the pressure and flow can be computed for any given point within the domain. It is assumed that the flow is equivalent to zero along the bottom edge and the left and right edges due to the nonpermeable nature of the material as well as symmetry. Along the top edge the piezometric pressure is chosen for each side of the barrier, this parameter largely controls the flow direction. A parameter study can be done, and is so by changing the values of a constant related to the barrier, recomputing the solution several times, and finally presenting it.

2 Program Description

The program is written using an object orientated programming approach and has a Model-View-Controller (MVC) architecture which means that one script calculates, one script controls the communication between the UI and the calculator and then the final participant is the UI file. Within the calculator, which in this case is called flowmodel, the script is built up of several classes; ModelParams, ModelResult, ModelVisulazation, ModelSolver and ModelReport which all contain functions corresponding to the tasks each class must fullfill. Within the communicator, called mainwindow, there are two classes; SolverThread which makes the calculator execute the calculations as well as a MainWindow which controls the rest of the actions. These actions include everything integrated within the UI system as a whole such as button pressing, changing variables, saving files and loading screens. Finally, the UI has been created separately and works together with the previous two to create a working program for calculating the groundwater flow of a certain domain.

3 Reasonableness

Using the given values and results from Worksheet 3 the programs accuracy is evaluated. Overall, the accuracy seems low but not unreasonable, this could be due to several factors including, which results are presented, how the maximum values are generated, as well as the size of the mesh. As shown in (a) the mesh size is 2, whereas the mesh size for the calculated values are never that big, this leads to inaccurate results due to the accuracy level of the calculations varying. The way the maximum values are found may also vary, the program uses the Pythagorean Theorem to calculate the largest flow whereas the given results may only use the flow in either the x - direction or the y - direction. This is a good explanation as to why the calculated values are larger than the given values.



```
Model info:

+----------------------+-----+
| Max element size:    |   2 |
| Dofs per node:       |   1 |
| Element type:        |   3 |
| Number of dofs:      | 422 |
| Number of elements:  | 359 |
| Number of nodes:     | 422 |
+----------------------+-----+

Summary of results:

+------------------------+---------+---------+
| Max piezometric head:  |  10.000 | [m]     |
| Min piezometric head:  |   0.000 | [m]     |
| Max boundary flow:     |  18.314 | [m/day] |
| Min boundary flow:     | -18.443 | [m/day] |
| Sum boundary flow:     |  -0.000 | [m/day] |
| Max resulting gw flow: |  32.534 | [m/day] |
| Min resulting gw flow: |   0.002 | [m/day] |
+------------------------+---------+---------+
```


(a) Worksheet 3 (2) (b) Mesh (1.0) (c) Mesh (0.5)

Figure 2: Comparison of Worksheet 3 and calculated results for two mesh resolutions.

4 User Manual

4.1 Startup and Inputs

When the program starts the following screen will show. From the startup screen a set of predetermined parameters for the domain are shown. These can be adjusted to the users liking according to the labels but don't need to be. The labels are self explanatory and if a value is missing when the calculation is run the program will notify the user.

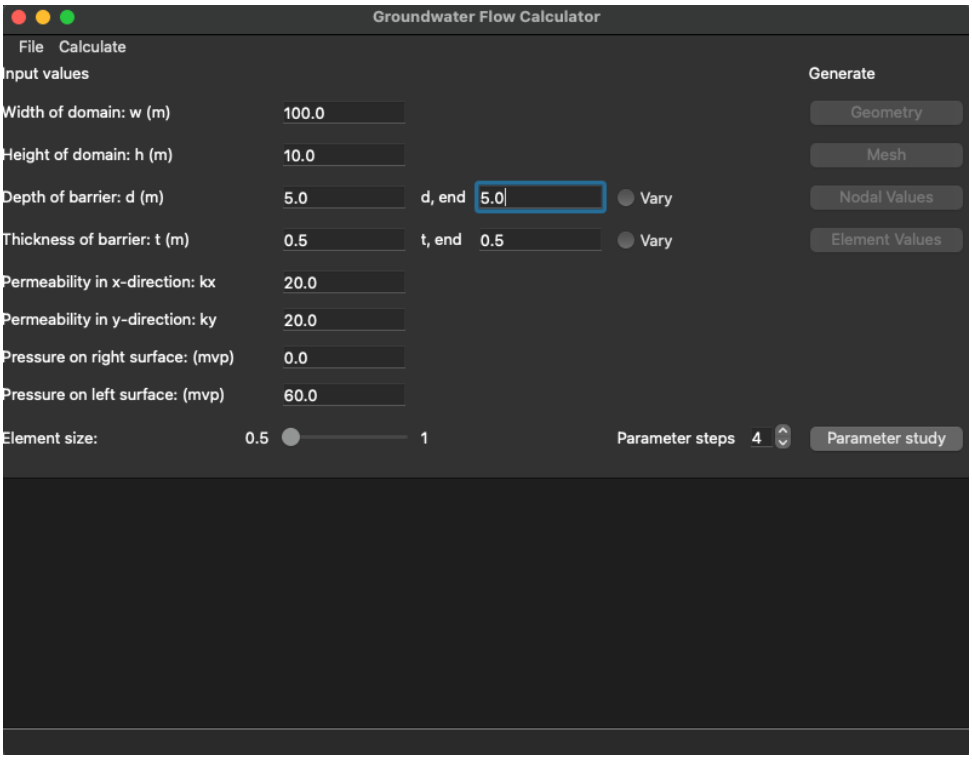


Figure 3: Startup screen.

4.2 Parameter Study

A parameter study can be computed with regards to either the depth of the barrier or the width of the barrier. This is done by the user when the program starts by selecting which parameter to vary, which values to vary between as well as how many values to study. The results are shown in the window at the bottom of the program as well as a graph comparing the resulting flow of each value. The parameter study needs to be done before executing calculations, after this the button isn't be available.

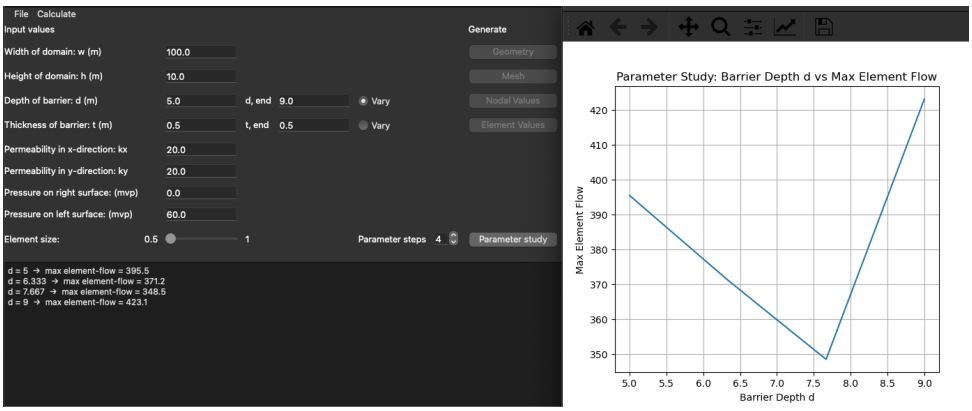


Figure 4: Parameter study.

4.3 Calculate

Within the Calculate menu there is one option, Execute. When this is pressed, the program takes the chosen variable values and computes the groundwater problem. After the calculations are done, the chosen parameters and results will print in the bottom part of the window as well as the buttons for showing the Geometry, Mesh, Nodal Values and Element Values becoming available.

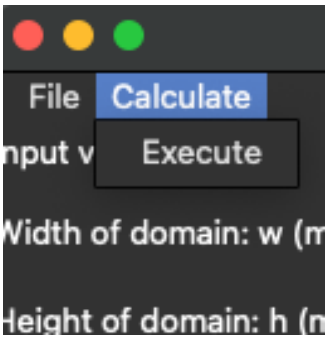
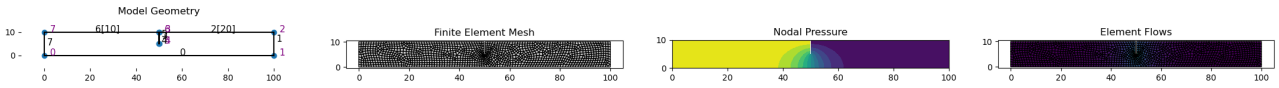


Figure 5: Calculate menu.



(a) Model geometry (b) Finite Element Mesh (c) Nodal Values (d) Element Values

Figure 6: Overview of the four possible figures to show.

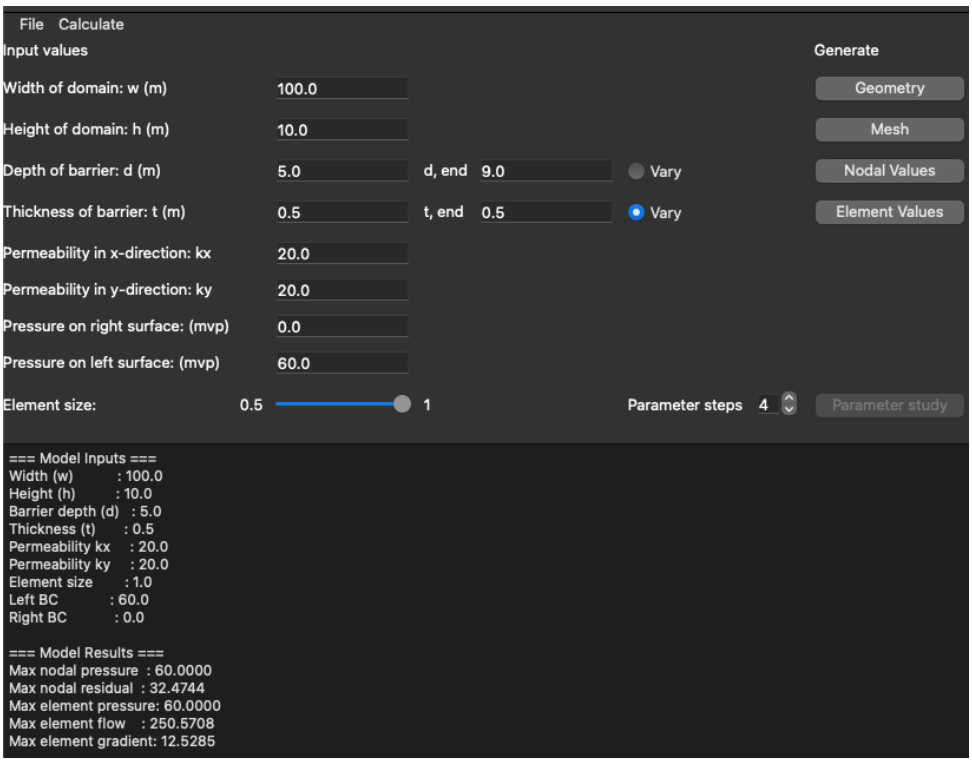


Figure 7: Program after calculating the groundwater flow.

4.4 File

In the File menu there are six options.

- New, which allows you to open up a new window for a new calculation.
- Open, which allows the user to access their files and choose a JSON file with presaved variable values.
- Save, which saves the current variable values if a JSON file already exists.
- Save as, which opens up the users library and allows the user to create a JSON file and choose where to save it.
- Export VTK, which allows the user to save a file anywhere in their library with the .VTK format which can later be opened in e.g. ParaVeiw.
- Exit, which closes the program.

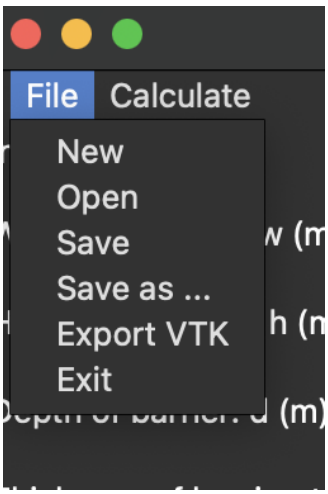


Figure 8: File menu.

5 Appendix

5.1 Main Script

```
# -*- coding: utf-8 -*-
import sys
import MAINWINDOW as mw
from qtpy.QtWidgets import QApplication

if __name__ == '__main__':

    # Create application instance
    app = QApplication(sys.argv)

    # Create and show main window
    widget = mw.MainWindow()
    widget.show()

    # Start main event loop
    sys.exit(app.exec_())
```

5.2 MainWindow Script

```
# -*- coding: utf-8 -*-
from qtpy.QtCore import Qt, QThread
from qtpy.QtWidgets import QApplication, QMainWindow, QFileDialog,
   QMessageBox, QProgressDialog
from qtpy.uic import loadUi
from qtpy.QtGui import QFont

import os
import sys
import xml.etree.ElementTree as ET
import matplotlib.pyplot as plt
import calfem.vis_mpl as cfv
import numpy as np

import FLOWMODEL as fm

def clean_ui(uifile):
    """Fix issues with Orientation:Horizontal/Vertical by creating
       _cleaned_mainwindow_5.ui"""
    tree = ET.parse(uifile)
    root = tree.getroot()
    for enum in root.findall(".//property[@name='orientation']/enum"):
        txt = enum.text or ''
        if 'Orientation::Horizontal' in txt:
            enum.text = 'Horizontal'
        elif 'Orientation::Vertical' in txt:
            enum.text = 'Vertical'
    clean_file = os.path.join(os.path.dirname(uifile), '
       CLEANED_MAINWINDOW.ui')
```

```
27      tree.write(clean_file, encoding='utf-8', xml_declaration=True)
28      return clean_file
29
30  class SolverThread(QThread):
31      def __init__(self, solver):
32          super().__init__()
33          self.solver = solver
34
35      def run(self):
36          self.solver.execute()
37
38  class MainWindow(QMainWindow):
39      """Main window for the application."""
40      def __init__(self):
41
42          """Constructor for the main window."""
43          super(QMainWindow, self).__init__()
44
45          # Set up
46          self.model_params = fm.ModelParams() # Model parameters
47          self.visualization = None # Visualization object
48          self.calc_done = False # Calculation status
49
50          # Clean UI file and load interface description
51          ui_path = os.path.join(os.path.dirname(__file__), 'MAINWINDOW.
                ui')
52          loadUi(clean_ui(ui_path), self) #loads CLEAN
53
54          # Font settings
55          mono = QFont("ComicSans", 12)
56          self.plainTextEdit.setFont(mono)
57
58          # Export VTK
59          if hasattr(self, 'actionExport_VTK'):
60              self.actionExport_VTK.triggered.connect(self.on_export_vtk
                    )
61          elif hasattr(self, 'actionExportVTK'):
62              self.actionExportVTK.triggered.connect(self.on_export_vtk)
63          else:
64              print("Warning: could not find Export VTK action in UI!")
65
66          # Menu placement in ui window
67          self.menuBar().setNativeMenuBar(False)
68
69          # Element size slider
70          self.element_size_label.setText('Element size:')
71          self.element_size_slider.setRange(50, 100)
72
73          # Set placeholders
74          placeholders = {
75              'w_text': '100.0 m',
76              'h_text': '10.0 m',
77              'd_text': '5.0 m',
```

```
78              't_text': '0.5 m',
79              'kx_text': '20.0 m/day',
80              'ky_text': '20.0 m/day',
81              'left_bc_text': '60.0 mvp',
82              'right_bc_text': '0.0 mvp'
83          }
84
85          # Set placeholder text
86          for attr, text in placeholders.items():
87              if hasattr(self, attr):
88                  widget = getattr(self, attr)
89                  widget.clear()
90                  widget.setPlaceholderText(text)
91
92          # Set default values
93          defaults = {
94              'w_text':           str(self.model_params.w),
95              'h_text':           str(self.model_params.h),
96              'd_text':           str(self.model_params.d),
97              't_text':           str(self.model_params.t),
98              'kx_text':          str(self.model_params.kx),
99              'ky_text':          str(self.model_params.ky),
100             'left_bc_text':     str(self.model_params.bc_values['
                    left_bc']),
101             'right_bc_text':    str(self.model_params.bc_values['
                    right_bc']),
102             'dEndEdit':         str(self.model_params.d),
103             'tEndEdit':         str(self.model_params.t),
104         }
105
106         # Set default values in UI
107         for attr, val in defaults.items():
108             if hasattr(self, attr):
109                 getattr(self, attr).setText(val)
110
111             self.element_size_slider.setValue(int(self.model_params.
                    el_size_factor * 100))
112
113         # Set default values for the parameter study
114         if hasattr(self, 'paramStep'):
115             self.paramStep.setValue(4)
116
117         # Clear checked radio buttons
118         if hasattr(self, 'paramVaryDRadio'):
119             self.paramVaryDRadio.setChecked(False)
120         if hasattr(self, 'paramVaryTRadio'):
121             self.paramVaryTRadio.setChecked(False)
122
123         # Disable buttons initially
124         for btn in (self.show_geometry_button,
125                     self.show_mesh_button,
126                     self.show_nodal_values_button,
127                     self.show_element_values_button):
```

```
128             btn.setEnabled(False)
129
130         # Connect menu actions
131         self.new_action.triggered.connect(self.on_new)
132         self.open_action.triggered.connect(self.on_open)
133         self.save_action.triggered.connect(self.on_save)
134         self.save_as_action.triggered.connect(self.on_save_as)
135         self.exit_action.triggered.connect(self.close)
136         self.execute_action.triggered.connect(self.on_execute)
137         self.paramButton.clicked.connect(self.on_execute_param_study)
138
139         # Connect visualization buttons
140         self.show_geometry_button.clicked.connect(self.
               on_show_geometry)
141         self.show_mesh_button.clicked.connect(self.on_show_mesh)
142         self.show_nodal_values_button.clicked.connect(self.
               on_show_nodal_values)
143         self.show_element_values_button.clicked.connect(self.
               on_show_element_values)
144
145         # Slider only updates element_size_factor
146         self.element_size_slider.valueChanged.connect(self.
               on_element_size_change)
147
148         self.model_params = None
149         self.model_results = None
150
151         self.show()
152         self.raise_()
153
154     def update_model(self):
155         """Read UI fields into model_params and update boundary
               conditions."""
156
157         # Ensure we have a ModelParams to write into
158         if not self.model_params:
159             self.model_params = fm.ModelParams()
160
161         # Define the mapping
162         fields = [
163             ('w_text',       'w',        'Width of domain (w)'),
164             ('h_text',       'h',        'Height of domain (h)'),
165             ('d_text',       'd',        'Depth of barrier (d)'),
166             ('t_text',       't',        'Thickness of barrier (t)'),
167             ('kx_text',      'kx',       'Permeability in x (kx)'),
168             ('ky_text',      'ky',       'Permeability in y (ky)'),
169             ('left_bc_text', 'left_bc',  'Left surface pressure (mvp)
                   '),
170             ('right_bc_text','right_bc', 'Right surface pressure (mvp)
                   '),
171         ]
172
173         invalid = []
```

```
174
175         # Read values from UI fields and set them in model_params
176         for widget_name, param_name, label in fields:
177             widget = getattr(self, widget_name, None)
178             txt = widget.text().strip() if widget else ''
179             try:
180                 value = float(txt)
181             except Exception:
182                 invalid.append(label)
183             else:
184                 setattr(self.model_params, param_name, value)
185
186         # Warnings for invalid inputs
187         if invalid:
188             QMessageBox.warning(
189                 self,
190                 'Invalid input',
191                 'Please enter valid numbers for:\n' + '\n'.join(
                        invalid)
192             )
193             return False
194
195         # Propagate into bc_values
196         if hasattr(self.model_params, 'bc_values'):
197             self.model_params.bc_values['left_bc']  = self.
                    model_params.left_bc
198             self.model_params.bc_values['right_bc'] = self.
                    model_params.right_bc
199
200         # Properties
201         mp = self.model_params
202         mp.D = np.array([[mp.kx, 0], [0, mp.ky]])
203         mp.el_size_factor = self.element_size_slider.value() / 100.0
204
205         return True
206
207     def on_new(self):
208         self.__init__()
209         self.paramButton.setEnabled(True)
210
211     def on_open(self):
212         """Open a model file and load its parameters into the UI."""
213
214         # Open file dialog to select a model file
215         fn, _ = QFileDialog.getOpenFileName(self, 'Open model', '', '
                Model files (*.json)')
216         if not fn: return
217         mp = fm.ModelParams()
218         try:
219             mp.load(fn)
220         except Exception as e:
221             QMessageBox.critical(self, 'Error', f'Load failed: {e}')
222             return
```

```
223
224         # Set the model parameters in the UI
225         for param, attr in [('w','w_text'),
226                             ('h','h_text'),
227                             ('d','d_text'),
228                             ('t','t_text'),
229                              ('kx','kx_text'),
230                              ('ky','ky_text'),
231                              ('left_bc','left_bc_text'),
232                              ('right_bc','right_bc_text')]:
233
234             if not hasattr(self, attr):
235                 continue
236             if param in mp.bc_values:
237                 text = str(mp.bc_values[param])
238             else:
239                 text = str(getattr(mp, param, ''))
240             getattr(self, attr).setText(text)
241
242         self.model_params = mp
243         self.model_results = None
244         for btn in (self.show_geometry_button, self.show_mesh_button,
245                     self.show_nodal_values_button, self.
                        show_element_values_button):
246             btn.setEnabled(False)
247
248     def on_save(self):
249         """Save model parameters to the current file or prompt for a
               new file."""
250         # Check if model_params is None or if update_model() fails
251         if not self.model_params:
252             QMessageBox.warning(self, 'Warning', 'Nothing to save or
                    invalid data.')
253             return
254         fn = getattr(self.model_params, 'filename', None)
255         if fn:
256             try:
257                 self.model_params.save(fn)
258             except Exception:
259                 self.on_save_as()
260         else:
261             self.on_save_as()
262
263     def on_export_vtk(self):
264         # Ensure we've actually run a solve (and have a solver to
               export from)
265         if not hasattr(self, 'solver') or self.solver is None:
266             QMessageBox.warning(self, "Nothing to export",
267                                 "Please run the simulation first.")
268             return
269
270         # Prompt user for filename
271         fn, _ = QFileDialog.getSaveFileName(
```

```
272             self,
273             "Export VTK File",
274             "",
275             "VTK files (*.vtk);; All files (*)"
276         )
277         if not fn:
278             return
279 
280         # Delegate to the solver's export method
281         try:
282             self.solver.export_vtk(fn)
283         except Exception as e:
284             QMessageBox.critical(self, "Export Failed",
285                                  f"Could not write VTK:\n{e}")
286             return
287 
288         QMessageBox.information(self, "Export Successful",
289                                 f"Wrote VTK file:\n{fn}")
290 
291     def on_save_as(self):
292         """Prompt for a file name and save model parameters to that
               file."""
293         # Ensure we have a model_params to save into
294         if not self.model_params:
295             self.model_params = fm.ModelParams()
296 
297         # Prompt for filename
298         fn, _ = QFileDialog.getSaveFileName(self, 'Save As', '', '
               Model files (*.json)')
299 
300         # If Cancel is pressed or no filename is provided
301         if not fn:
302             return
303 
304         # Save the model parameters to the specified file
305         try:
306             _ = self.update_model()
307             self.model_params.save(fn)
308             self.model_params.filename = fn
309         except Exception as e:
310             QMessageBox.critical(self, 'Error', f'Save failed: {e}')
311 
312     def on_execute(self):
313         """Run solver unless already executed; prompt to start new
               model if so"""
314         # Prevent re-execution
315         if self.model_results is not None:
316             QMessageBox.warning(
317                 self,
318                 'Execution Already Run',
319                 'To generate another domain create a new file.'
320             )
321             return
```

```
322
323         # UI values into model_params
324         if not self.update_model():
325             return
326
327         # Disable UI until solver finishes
328         self.calc_done = False
329         self.setEnabled(False)
330
331         # Create fresh result & solver
332         self.model_results = fm.ModelResult()
333         self.solver = fm.ModelSolver(self.model_params, self.
              model_results)
334
335         # Show   please   wait  dialog
336         progress = QProgressDialog("Running simulation ", None, 0,
              0, self)
337         progress.setWindowTitle("Please wait")
338         progress.setWindowModality(Qt.ApplicationModal)
339         progress.setCancelButton(None)
340         progress.setMinimumDuration(0)
341         progress.show()
342
343         # Launch the solver in its thread
344         self.solverThread = SolverThread(self.solver)
345         self.solverThread.finished.connect(progress.close)
346         self.solverThread.finished.connect(self.on_solver_finished)
347         self.solverThread.start()
348
349     def on_solver_finished(self):
350         """Handle completion of the solver thread."""
351         self.setEnabled(True)
352
353         # Caluclation finished
354         self.calc_done = True
355
356         # Recreate visualization object
357         self.visualization = fm.ModelVisualization(self.model_params,
              self.model_results)
358
359         for btn in (self.show_geometry_button, self.show_mesh_button,
360                     self.show_nodal_values_button, self.
                        show_element_values_button):
361             btn.setEnabled(True)
362
363                 # --- Build a neat, monospaced summary
364
365         self.paramButton.setEnabled(False)
366
367         mp = self.model_params
368         mr = self.model_results
369
370         lines = []
```

```
371         lines.append("=== Model Inputs ===")
372         # list of (label, value) tuples in desired order
373         inputs = [
374             ("Width (w)"       , mp.w),
375             ("Height (h)"      , mp.h),
376             ("Barrier depth (d)", mp.d),
377             ("Thickness (t)"   , mp.t),
378             ("Permeability kx" , mp.kx),
379             ("Permeability ky" , mp.ky),
380             ("Element size"    , mp.el_size_factor),
381             ("Left BC"         , mp.bc_values["left_bc"]),
382             ("Right BC"        , mp.bc_values["right_bc"]),
383         ]
384         # align the colon at column  twenty for neatness
385         for label, val in inputs:
386             lines.append(f"{label:<20s}: {val}")
387 
388         lines.append("")  # blank separator
389         lines.append("=== Model Results ===")
390         results = [
391             ("Max nodal pressure"  , mr.max_nodal_pressure),
392             ("Max nodal residual"  , mr.max_nodal_flow),
393             ("Max element pressure", mr.max_element_pressure),
394             ("Max element flow"    , mr.max_element_flow),
395             ("Max element gradient", mr.max_element_gradient),
396         ]
397         for label, val in results:
398             lines.append(f"{label:<20s}: {val:.4f}")
399 
400         # set into the UI
401         self.plainTextEdit.setPlainText("\n".join(lines))
402 
403     def on_show_geometry(self):
404         """Display the geometry of the model."""
405 
406         if not self.calc_done or self.visualization is None:
407             QMessageBox.warning(self, 'No Data', 'Please run the
                calculation first.')
408             return
409         self.visualization.show_geometry()
410 
411     def on_show_mesh(self):
412         """Display the finite element mesh of the model."""
413 
414         if not self.calc_done or self.visualization is None:
415             QMessageBox.warning(self, 'No Data', 'Please run the
                calculation first.')
416             return
417         self.visualization.show_mesh()
418 
419     def on_show_nodal_values(self):
420         """Display nodal values of the model."""
421 
```

```
422         if not self.calc_done or self.visualization is None:
423             QMessageBox.warning(self, 'No Data', 'Please run the
                    calculation first.')
424             return
425         self.visualization.show_nodal_values()
426 
427     def on_show_element_values(self):
428         """Display element values of the model."""
429 
430         if not self.calc_done or self.visualization is None:
431             QMessageBox.warning(self, 'No Data', 'Please run the
                    calculation first.')
432             return
433         self.visualization.show_element_values()
434 
435     def on_element_size_change(self, value):
436         """Update the element size factor based on the slider value
                ."""
437         if self.model_params is None:
438             self.model_params = fm.ModelParams()
439         self.model_params.el_size_factor = value / 100.0
440 
441     def on_execute_param_study(self):
442         """Run a parameter study, either on depth (d) or thickness (t)
                , and log results."""
443         # Params from the UI
444         if not self.update_model():
445             return
446 
447         # Decide which parameter to vary
448         if self.paramVaryDRadio.isChecked():
449             var_name = 'd'
450             start_val = self.model_params.d
451             try:
452                 end_val = float(self.dEndEdit.text())
453             except ValueError:
454                 QMessageBox.warning(self, 'Invalid Input',
455                                     'Depth  e n d value  must be a
                                            number.')
456                 return
457             xlabel = 'Barrier Depth d'
458         elif self.paramVaryTRadio.isChecked():
459             var_name = 't'
460             start_val = self.model_params.t
461             try:
462                 end_val = float(self.tEndEdit.text())
463             except ValueError:
464                 QMessageBox.warning(self, 'Invalid Input',
465                                     'Thickness  e n d value  must be a
                                            number.')
466                 return
467             xlabel = 'Barrier Thickness t'
468         else:
```

```
469             QMessageBox.warning(self, 'Parameter Study',
470                                 'Please check   Vary    d   or   Vary
                                        t   to enable a sweep.')
471             return
472
473         # Steps from the UI
474         n_steps = self.paramStep.value()
475         if n_steps < 2:
476             QMessageBox.warning(self, 'Invalid Input',
477                                 'Number of steps must be at least 2.')
478             return
479
480         vals = np.linspace(start_val, end_val, n_steps)
481
482         # Progress dialog
483         progress = QProgressDialog(f"Running parameter study of {
               var_name}   ",
484                                    "Abort", 0, n_steps-1, self)
485         progress.setWindowTitle("Please wait")
486         progress.setWindowModality(Qt.WindowModal)
487         progress.setMinimumDuration(0)
488         progress.show()
489
490         # Prepare for plotting
491         self.plainTextEdit.clear()
492         flows = []
493
494         # Abort button
495         for i, v in enumerate(vals):
496             progress.setValue(i)
497             QApplication.processEvents()
498             if progress.wasCanceled():
499                 break
500
501             # Copy all other params into a fresh ModelParams
502             base = self.model_params
503             p = fm.ModelParams()
504             p.w = base.w
505             p.h = base.h
506             p.d = v if var_name == 'd' else base.d
507             p.t = v if var_name == 't' else base.t
508             p.kx = base.kx
509             p.ky = base.ky
510             p.bc_markers = base.bc_markers
511             p.bc_values= base.bc_values.copy()
512             p.load_markers = base.load_markers
513             p.load_values = base.load_values.copy()
514             p.el_size_factor = base.el_size_factor
515
516             # Solve the model
517             mr = fm.ModelResult()
518             solver = fm.ModelSolver(p, mr)
519             solver.execute()
```

```
520
521             mf = mr.max_element_flow
522             flows.append(mf)
523
524             # Log into the plainTextEdit
525             self.plainTextEdit.appendPlainText(
526                 f"{var_name} = {v:.4g}       max element-flow = {mf:.4
                        g}"
527             )
528
529         progress.setValue(n_steps-1)
530         progress.close()
531
532         cfv.figure()
533         plt.clf()
534         plt.plot(vals[:len(flows)], flows)
535         plt.xlabel(xlabel)
536         plt.ylabel('Max Element Flow')
537         plt.title(f'Parameter Study: {xlabel} vs Max Element Flow')
538         plt.grid(True)
539         cfv.show()
540
541 if __name__ == '__main__':
542     app = QApplication(sys.argv)
543     window = MainWindow()
544     sys.exit(app.exec_())
```

5.3 Flowmodel Script

```
1  # -*- coding: utf-8 -*-
2  import json
3  import pyvtk as vtk
4
5  import calfem.core as cfc
6  import calfem.geometry as cfg
7  import calfem.mesh as cfm
8  import calfem.vis_mpl as cfv
9  import calfem.utils as cfu
10
11 import matplotlib.pylab as plt
12 import tabulate as tab
13 import numpy as np
14
15 class ModelParams:
16     """Class defining parametric model properties"""
17     def __init__(self):
18
19         # Version tracking
20         self.version = 1
21
22         # Geometric parameters
23         self.w = 100.0 # Width of domain
```

```
24          self.h = 10.0 # Height of domain
25          self.d = 5.0 # Depth of barrier
26          self.t = 0.5 # Thickness of barrier
27          self.ep = [self.t, int(2)] # Element properties
28
29          # Material properties
30          self.kx = 20.0 # Permeability in x-direction
31          self.ky = 20.0 # Permeability in y-direction
32          self.D = np.array([[self.kx, 0], [0, self.ky]]) # Permeability
                matrix
33
34          # Mesh control
35          self.el_size_factor = 0.5 # Elements size in mesh
36
37          # Boundary conditions and loads
38          self.bc_markers = {
39              "left_bc": 10, # Marker for left boundary
40              "right_bc": 20 # Marker for right boundary
41          }
42
43          self.bc_values = {
44              "left_bc": 60.0, # Value for left boundary
45              "right_bc": 0.0  # Value for right boundary
46          }
47
48          self.load_markers = {
49          }
50
51          self.load_values = {
52          }
53
54      def geometry(self):
55          """Create and return a geometry instance based on defined
                parameters"""
56
57          # Use shorter variable names for readability
58          w = self.w
59          h = self.h
60          t = self.t
61          d = self.d
62
63          # Create a geometry object
64          g = cfg.Geometry()
65
66          # Define points for the geometry
67          g.point([0, 0]) # Point 0: Bottom left corner
68          g.point([w, 0]) # Point 1: Bottom right corner
69          g.point([w, h]) # Point 2: Top right corner
70          g.point([w/2+t/2, h]) # Point 3: Top right corner of barrier
71          g.point([w/2+t/2, h-d]) # Point 4: Bottom right corner of
                barrier
72          g.point([w/2-t/2, h-d]) # Point 5: Bottom left corner of
                barrier
```

```
73          g.point([w/2-t/2, h]) # Point 6: Top left corner of barrier
74          g.point([0, h]) # Point 7: Top left corner
75
76          # Define splines connecting the points
77          g.spline([0, 1])
78          g.spline([1, 2])
79          g.spline([2, 3], marker=self.bc_markers["right_bc"])
80          g.spline([3, 4])
81          g.spline([4, 5])
82          g.spline([5, 6])
83          g.spline([6, 7], marker=self.bc_markers["left_bc"])
84          g.spline([7, 0])
85
86          # Define the surface using the spline indices
87          g.surface([0, 1, 2, 3, 4, 5, 6, 7])
88
89          return g
90
91      def save(self, filename):
92          """Save input to file."""
93
94          model_params = {} # Create a dictionary to store model
               parameters
95          model_params["version"] = self.version
96          model_params["t"] = self.t
97          model_params["ep"] = self.ep
98          model_params["w"] = self.w
99          model_params["h"] = self.h
100         model_params["d"] = self.d
101         model_params["kx"] = self.kx
102         model_params["ky"] = self.ky
103         model_params["D"] = self.D.tolist() # Convert numpy array to
               list for JSON compatibility
104         model_params["el_size_factor"] = self.el_size_factor
105         model_params["bc_markers"] = self.bc_markers
106         model_params["bc_values"] = self.bc_values
107         model_params["load_markers"] = self.load_markers
108         model_params["load_values"] = self.load_values
109
110         # Write the model parameters to a JSON file
111         ofile = open(filename, "w")
112         json.dump(model_params, ofile, sort_keys = True, indent = 4)
113         ofile.close()
114
115     def load(self, filename):
116         """Read input from file."""
117
118         # Read the model parameters from a JSON file
119         ifile = open(filename, "r")
120         model_params = json.load(ifile)
121         ifile.close()
122
123         self.version = model_params["version"]
```

```
124         self.t = model_params["t"]
125         self.ep = model_params["ep"]
126         self.w = model_params["w"]
127         self.h = model_params["h"]
128         self.d = model_params["d"]
129         self.kx = model_params["kx"]
130         self.ky = model_params["ky"]
131         self.D = np.array(model_params["D"]) # Convert list back to
              numpy array
132         self.el_size_factor = model_params["el_size_factor"]
133         self.bc_markers = model_params["bc_markers"]
134         self.bc_values = model_params["bc_values"]
135         self.load_markers = model_params["load_markers"]
136         self.load_values = model_params["load_values"]
137
138 class ModelResult:
139     """Class for storing results from calculations."""
140
141     def __init__(self):
142
143         # Initialize attributes for mesh and geometry
144         self.loads = None
145         self.bcs = None
146         self.edof = None
147         self.coords = None
148         self.dofs = None
149         self.bdofs = None
150         self.boundary_elements = None
151         self.geometry = None
152
153         # Initialize attributes for results
154         self.a = None
155         self.r = None
156         self.ed = None
157         self.es = None
158         self.et = None
159
160         self.flow = None
161         self.pressure = None
162         self.gradient = None
163
164         self.max_nodal_flow = None
165         self.max_nodal_pressure = None
166         self.max_element_flow = None
167         self.max_elemetn_pressure = None
168         self.max_element_gradient = None
169
170 class ModelVisualization:
171     """Class for visualizing model geometry, mesh, and results"""
172
173     def __init__(self, model_params, model_result):
174         """Constructor"""
175
```

```
176         # Store references to model parameters and results
177         self.model_params = model_params
178         self.model_result = model_result
179 
180         # Store references to visualization windows
181         self.geom_fig = None
182         self.mesh_fig = None
183         self.node_value_fig = None
184         self.element_value_fig = None
185 
186     def show_geometry(self):
187         """Display model geometry"""
188 
189         # Create a new figure
190         cfv.figure()
191         cfv.clf()
192 
193         # Draw Geometry
194         cfv.draw_geometry(
195             geometry = self.model_params.geometry(),
196             draw_points=True,
197             label_points=True,
198             label_curves=True,
199             title="Model Geometry"
200         )
201 
202         cfv.show_and_wait()
203 
204     def show_mesh(self):
205         """Display Finite Element Mesh"""
206 
207         # Create a new figure
208         cfv.figure()
209         cfv.clf()
210 
211         # Draw Mesh
212         cfv.draw_mesh(
213             coords=self.model_result.coords,
214             edof=self.model_result.edof,
215             dofs_per_node=self.model_result.dofs_per_node,
216             el_type=self.model_result.el_type,
217             filled=True,
218             title="Finite Element Mesh"
219         )
220 
221         cfv.show_and_wait()
222 
223     def show_nodal_values(self):
224         """Display Nodal Pressure"""
225 
226         # Create a new figure
227         cfv.figure()
228         cfv.clf()
```

```
229 |
230 |         # Draw Nodal Pressure
231 |         cfv.draw_nodal_values(
232 |             self.model_result.a,
233 |             coords=self.model_result.coords,
234 |             edof=self.model_result.edof,
235 |             title="Nodal Pressure"
236 |         )
237 |
238 |         cfv.show_and_wait()
239 |
240 |     def show_element_values(self):
241 |         """Display Element Flows"""
242 |
243 |         # Create a new figure
244 |         cfv.figure()
245 |         cfv.clf()
246 |
247 |         # Draw element flows
248 |         cfv.draw_element_values(
249 |             self.model_result.flow,
250 |             coords=self.model_result.coords,
251 |             edof=self.model_result.edof,
252 |             dofs_per_node=self.model_result.dofs_per_node,
253 |             el_type=self.model_result.el_type,
254 |             title="Element Flows"
255 |         )
256 |
257 |         cfv.show_and_wait()
258 |
259 |     def wait(self):
260 |         """Wait for user to close all visualization windows"""
261 |
262 |         cfv.show_and_wait()
263 |
264 | class ModelSolver:
265 |     """Class for solving the finite element model"""
266 |
267 |     def __init__(self, model_params, model_result):
268 |         self.model_params = model_params
269 |         self.model_result = model_result
270 |
271 |     def execute(self):
272 |         """Perform mesh generation and finite element computation"""
273 |
274 |         # Create shorter references to input variables
275 |         ep = self.model_params.ep
276 |         kx = self.model_params.kx
277 |         ky = self.model_params.ky
278 |         D = self.model_params.D
279 |
280 |         # Get geometry and store it
281 |         geometry = self.model_params.geometry()
```

```
282         self.model_result.geometry = geometry
283 
284         # Set up mesh generation
285         el_type = 3
286         dofs_per_node = 1
287 
288         # Create mesh generator
289         mesh = cfm.GmshMeshGenerator(geometry)
290 
291         # Configure mesh generator
292         mesh.el_type = el_type
293         mesh.dofs_per_node = dofs_per_node
294         mesh.el_size_factor = self.model_params.el_size_factor
295         mesh.return_boundary_elements = True
296 
297         # Generate mesh
298         coords, edof, dofs, bdofs, element_markers, boundary_elements
               = mesh.create()
299 
300         # Store mesh data in results
301         self.model_result.coords = coords
302         self.model_result.edof = edof
303         self.model_result.dofs = dofs
304         self.model_result.bdofs = bdofs
305         self.model_result.element_markers = element_markers
306         self.model_result.boundary_elements = boundary_elements
307         self.model_result.el_type = el_type
308         self.model_result.dofs_per_node = dofs_per_node
309 
310         # Create global stiffness matrix and load vector
311         n_dofs = np.max(dofs)
312         K = np.zeros((n_dofs, n_dofs))
313         f = np.zeros((n_dofs, 1))
314 
315         # Global stiffness matrix
316         nDofs = np.size(dofs) # Number of global degrees of freedom
317         ex, ey = cfc.coordxtr(edof, coords, dofs) # Extract
               coordinates of elements
318         K = np.zeros([nDofs, nDofs]) # Global stiffness matrix
319 
320         n_el = edof.shape[0] # Number of elements
321         ep = np.tile(self.model_params.ep, (n_el, 1)).astype(object)
322 
323         # Assemble global stiffness matrix
324         for i, (eltopo, elx, ely) in enumerate(zip(edof, ex, ey)):
325             thickness = float(ep[i][0])
326             integration_rule = int(ep[i][1])
327             el_ep = [thickness, integration_rule]
328             Ke = cfc.flw2i4e(elx, ely, el_ep, D)
329             cfc.assem(eltopo, K, Ke)
330 
331         # Global load vector
332         f = np.zeros([nDofs, 1])
```

```
333
334         # Boundary conditions
335         bc = np.array([], int)
336         bcVal = np.array([], int)
337
338         # Apply boundary conditions
339         for name, marker in self.model_params.bc_markers.items():
340             value = self.model_params.bc_values.get(name, 0.0)
341             bc, bcVal = cfu.applybc(bdofs, bc, bcVal, marker, value)
342
343         # Solve the system of equations
344         a, r = cfc.solveq(K, f, bc, bcVal)
345         ed = cfc.extractEldisp(edof, a)
346
347         # Calculate element flows
348         flow = [] # List to store flow values
349         gradient = [] # List to store gradient values
350
351         for i in range(edof.shape[0]):
352             el_ep = [float(ep[i][0]), int(ep[i][1])]
353             es, et, eci = cfc.flw2i4s(ex[i, :], ey[i, :], el_ep, D, ed
                    [i, :])
354             flow.append(np.sqrt(es[0, 0]**2 + es[0, 1]**2))
355             gradient.append(np.sqrt(et[0, 0]**2 + et[0, 1]**2))
356
357         # Maximal flow, pressure, gradient for nodes and elements
358         max_nodal_pressure = np.max(np.abs(a))
359         max_nodal_flow = np.max(np.abs(r))
360         max_element_pressure = np.max(np.abs(ed))
361         max_element_flow = np.max(np.abs(flow))
362         max_element_gradient = np.max(np.abs(gradient))
363
364         # Store results in model_result and model_params
365         self.model_result.loads = list(zip(bc, bcVal))
366         self.model_result.bcs = list(zip(bc, bcVal))
367         self.model_result.edof = edof
368         self.model_params.coord = coords
369         self.model_params.dof = dofs
370         self.model_params.elem = np.arange(edof.shape[0])
371
372         self.model_result.a = a
373         self.model_result.r = r
374         self.model_result.ed = ed
375         self.model_result.es = es
376         self.model_result.et = et
377
378         self.model_result.flow = flow
379         self.model_result.gradient = gradient
380
381         self.model_result.max_nodal_flow = max_nodal_flow
382         self.model_result.max_nodal_pressure = max_nodal_pressure
383         self.model_result.max_element_flow = max_element_flow
384         self.model_result.max_element_pressure = max_element_pressure
```

```
385         self.model_result.max_element_gradient = max_element_gradient
386
387     def run_parameter_study(self):
388         """Run a parameter study by varying the barrier depth"""
389
390         # Parameters to vary
391         d_values = np.linspace(1.0, 9.0, 9)
392         max_flow_values = []
393
394         # Run simulation for each value
395         for d in d_values:
396             print(f"Simulating with barrier depth d = {d:.2f}...")
397             # Create model with current parameter
398             model_params = ModelParams()
399             model_params.d = d  # Set current barrier depth
400
401             # Other parameters remain constant
402             model_params.w = 100.0
403             model_params.h = 10.0
404             model_params.t = 0.5
405             model_params.kx = 20.0
406             model_params.ky = 20.0
407
408             # Create result storage and solver
409             model_result = ModelResult()
410             model_solver = ModelSolver(model_params, model_result)
411
412             # Run the simulation
413             model_solver.execute()
414
415             # Compute the norm of the flow vector for each element
416             flow_norms = np.linalg.norm(model_result.es, axis=1)
417
418             # Store the maximum flow for this configuration
419             max_flow_values.append(np.max(flow_norms))
420             print(f"Maximum flow value: {np.max(flow_norms):.4f}")
421
422
423         # Plot the results
424         plt.figure(figsize=(10, 6))
425         plt.plot(d_values, max_flow_values, 'o-', linewidth=2)
426         plt.grid(True)
427         plt.xlabel('Barrier Depth (d)')
428         plt.ylabel('Maximum Flow')
429         plt.title('Parameter Study: Effect of Barrier Depth on Maximum
                Flow')
430         plt.savefig('parameter_study.png')
431         plt.show()
432
433         # Return results for further analysis
434         return d_values, max_flow_values
435
436     def export_vtk(self, filename):
```

```
437         """Export  n o d e pressures  and  c e l l flows  to a VTK file."""
438 
439         print(f"Exporting results to {filename!r}...")
440 
441         # Import geometry, mesh and flow
442         points = self.model_result.coords.tolist()
443         polygons = (self.model_result.edof[:, 1:] - 1).tolist()
444         point_data = vtk.PointData(
445             vtk.Scalars(self.model_result.a.tolist(), name="pressure")
446         )
447         cell_data = vtk.CellData(
448             vtk.Scalars(self.model_result.flow, name="flow")
449         )
450 
451         # Create VTK structure
452         structure = vtk.PolyData(points=points, polygons=polygons)
453         vtk_data  = vtk.VtkData(structure, point_data, cell_data)
454         vtk_data.tofile(filename, "ascii")
455 
456         print("VTK export complete.")
457 
458 class ModelReport:
459     """Class for presenting input and output parameters in report form
        ."""
460 
461     def __init__(self, model_params, model_result):
462         self.model_params = model_params
463         self.model_result = model_result
464         self.report = ""
465 
466     def clear(self):
467         self.report = ""
468 
469     def add_text(self, text=""):
470         self.report+=str(text)+"\n"
471 
472     def __str__(self):
473         self.clear()
474         self.add_text()
475         self.add_text("-------------- Model Inputs
            ----------------------------------")
476         self.add_text()
477         self.add_text(
478             tab.tabulate([
479             ["t", self.model_params.t],
480             ["w", self.model_params.w],
481             ["h", self.model_params.h],
482             ["d", self.model_params.d],
483             ["kx", self.model_params.kx],
484             ["ky", self.model_params.ky],
485             ["Element size", self.model_params.el_size_factor],
486             ["Left boundary", self.model_params.bc_values.get("left_bc
                ", "N/A")],
```

```
487             ["Right boundary", self.model_params.bc_values.get("
                    right_bc", "N/A")]
488             ],
489             headers=["Parameter", "Value"],
490             numalign="right",
491             floatfmt=".1f",
492             tablefmt="psql"
493             )
494         )
495 
496         self.add_text()
497         self.add_text("-------------- Model results
                    ---------------------------------")
498         self.add_text()
499         self.add_text(
500             tab.tabulate(
501             [[
502                 self.model_result.max_nodal_pressure,
503                 self.model_result.max_nodal_flow,
504                 self.model_result.max_element_pressure,
505                 self.model_result.max_element_flow,
506                 self.model_result.max_element_gradient
507             ]],
508             headers=[
509                 "Max Nodal Pressure",
510                 "Max Nodal Flow",
511                 "Max Element Pressure",
512                 "Max Element Flow",
513                 "Max Element Gradient"
514             ],
515             numalign="right",
516             floatfmt=".4f",
517             tablefmt="psql"
518             )
519         )
520 
521         return self.report
```

5.4 UI file

```
1     <?xml version="1.0" encoding="UTF-8"?>
2 <ui version="4.0">
3  <class>GroundwaterflowCalculator</class>
4  <widget class="QMainWindow" name="GroundwaterflowCalculator">
5   <property name="geometry">
6    <rect>
7     <x>0</x>
8     <y>0</y>
9     <width>800</width>
10    <height>600</height>
11   </rect>
12  </property>
```

```
13  <property name="windowTitle">
14   <string>Groundwater Flow Calculator</string>
15  </property>
16  <widget class="QWidget" name="centralwidget">
17   <layout class="QVBoxLayout" name="verticalLayout">
18    <item>
19     <layout class="QGridLayout" name="gridLayout">
20      <item row="3" column="4">
21       <widget class="QLineEdit" name="dEndEdit"/>
22      </item>
23      <item row="4" column="0">
24       <widget class="QLabel" name="t_label">
25        <property name="text">
26         <string>Thickness of barrier: t (m)</string>
27        </property>
28       </widget>
29      </item>
30      <item row="3" column="0">
31       <widget class="QLabel" name="d_label">
32        <property name="text">
33         <string>Depth of barrier: d (m)</string>
34        </property>
35       </widget>
36      </item>
37      <item row="1" column="7">
38       <widget class="QPushButton" name="show_geometry_button">
39        <property name="text">
40         <string>Geometry</string>
41        </property>
42       </widget>
43      </item>
44      <item row="8" column="0">
45       <widget class="QLabel" name="left_bc_label">
46        <property name="text">
47         <string>Pressure on left surface: (mvp)</string>
48        </property>
49       </widget>
50      </item>
51      <item row="1" column="0">
52       <widget class="QLabel" name="w_label">
53        <property name="text">
54         <string>Width of domain: w (m)</string>
55        </property>
56       </widget>
57      </item>
58      <item row="9" column="3">
59       <widget class="QLabel" name="label_4">
60        <property name="text">
61         <string>1</string>
62        </property>
63       </widget>
64      </item>
65      <item row="4" column="2">
```

```
66          <widget class="QLineEdit" name="t_text"/>
67         </item>
68         <item row="7" column="2">
69          <widget class="QLineEdit" name="right_bc_text"/>
70         </item>
71         <item row="3" column="3">
72          <widget class="QLabel" name="label">
73           <property name="text">
74            <string>d, end</string>
75           </property>
76          </widget>
77         </item>
78         <item row="5" column="2">
79          <widget class="QLineEdit" name="kx_text"/>
80         </item>
81         <item row="3" column="2">
82          <widget class="QLineEdit" name="d_text"/>
83         </item>
84         <item row="9" column="6">
85          <widget class="QSpinBox" name="paramStep"/>
86         </item>
87         <item row="3" column="5">
88          <widget class="QRadioButton" name="paramVaryDRadio">
89           <property name="text">
90            <string>Vary</string>
91           </property>
92          </widget>
93         </item>
94         <item row="4" column="4">
95          <widget class="QLineEdit" name="tEndEdit"/>
96         </item>
97         <item row="5" column="0">
98          <widget class="QLabel" name="kx_label">
99           <property name="text">
100           <string>Permeability in x-direction: kx</string>
101          </property>
102         </widget>
103        </item>
104        <item row="6" column="0">
105         <widget class="QLabel" name="ky_label">
106          <property name="text">
107           <string>Permeability in y-direction: ky</string>
108          </property>
109         </widget>
110        </item>
111        <item row="4" column="7">
112         <widget class="QPushButton" name="show_element_values_button">
113          <property name="text">
114           <string>Element Values</string>
115          </property>
116         </widget>
117        </item>
118        <item row="9" column="7">
```

```
119         <widget class="QPushButton" name="paramButton">
120          <property name="text">
121           <string>Parameter study</string>
122          </property>
123         </widget>
124        </item>
125        <item row="3" column="7">
126         <widget class="QPushButton" name="show_nodal_values_button">
127          <property name="text">
128           <string>Nodal Values</string>
129          </property>
130         </widget>
131        </item>
132        <item row="9" column="0">
133         <widget class="QLabel" name="element_size_label">
134          <property name="text">
135           <string>Element size 0.5</string>
136          </property>
137         </widget>
138        </item>
139        <item row="9" column="5">
140         <widget class="QLabel" name="label_2">
141          <property name="text">
142           <string>Parameter steps</string>
143          </property>
144         </widget>
145        </item>
146        <item row="8" column="2">
147         <widget class="QLineEdit" name="left_bc_text"/>
148        </item>
149        <item row="6" column="2">
150         <widget class="QLineEdit" name="ky_text"/>
151        </item>
152        <item row="1" column="2">
153         <widget class="QLineEdit" name="w_text"/>
154        </item>
155        <item row="4" column="3">
156         <widget class="QLabel" name="label_3">
157          <property name="text">
158           <string>t, end</string>
159          </property>
160         </widget>
161        </item>
162        <item row="0" column="0">
163         <widget class="QLabel" name="input_values_label">
164          <property name="text">
165           <string>Input values</string>
166          </property>
167         </widget>
168        </item>
169        <item row="7" column="0">
170         <widget class="QLabel" name="right_bc_label">
171          <property name="text">
```

```
172           <string>Pressure on right surface: (mvp)</string>
173          </property>
174         </widget>
175        </item>
176        <item row="2" column="7">
177         <widget class="QPushButton" name="show_mesh_button">
178          <property name="text">
179           <string>Mesh</string>
180          </property>
181         </widget>
182        </item>
183        <item row="2" column="0">
184         <widget class="QLabel" name="h_label">
185          <property name="text">
186           <string>Height of domain: h (m)</string>
187          </property>
188         </widget>
189        </item>
190        <item row="9" column="2">
191         <widget class="QSlider" name="element_size_slider">
192          <property name="orientation">
193           <enum>Qt::Orientation::Horizontal</enum>
194          </property>
195         </widget>
196        </item>
197        <item row="0" column="7">
198         <widget class="QLabel" name="generation_label">
199          <property name="text">
200           <string>Generate</string>
201          </property>
202         </widget>
203        </item>
204        <item row="4" column="5">
205         <widget class="QRadioButton" name="paramVaryTRadio">
206          <property name="text">
207           <string>Vary</string>
208          </property>
209         </widget>
210        </item>
211        <item row="2" column="2">
212         <widget class="QLineEdit" name="h_text"/>
213        </item>
214        <item row="9" column="1">
215         <widget class="QLabel" name="label_5">
216          <property name="text">
217           <string>0.5</string>
218          </property>
219         </widget>
220        </item>
221       </layout>
222      </item>
223      <item>
224       <widget class="QPlainTextEdit" name="plainTextEdit"/>
```

```
    </item>
   </layout>
  </widget>
  <widget class="QMenuBar" name="menubar">
   <property name="geometry">
    <rect>
     <x>0</x>
     <y>0</y>
     <width>800</width>
     <height>24</height>
    </rect>
   </property>
   <widget class="QMenu" name="menu_file">
    <property name="title">
     <string>File</string>
    </property>
    <addaction name="new_action"/>
    <addaction name="open_action"/>
    <addaction name="save_action"/>
    <addaction name="save_as_action"/>
    <addaction name="actionExport_VTK"/>
    <addaction name="exit_action"/>
   </widget>
   <widget class="QMenu" name="menu_calculate">
    <property name="title">
     <string>Calculate</string>
    </property>
    <addaction name="execute_action"/>
   </widget>
   <addaction name="menu_file"/>
   <addaction name="menu_calculate"/>
  </widget>
  <widget class="QStatusBar" name="statusbar"/>
  <action name="new_action">
   <property name="text">
    <string>New</string>
   </property>
  </action>
  <action name="open_action">
   <property name="text">
    <string>Open</string>
   </property>
  </action>
  <action name="save_action">
   <property name="text">
    <string>Save</string>
   </property>
  </action>
  <action name="save_as_action">
   <property name="text">
    <string>Save as ...</string>
   </property>
  </action>
```

```
  <action name="exit_action">
   <property name="text">
    <string>Exit</string>
   </property>
  </action>
  <action name="execute_action">
   <property name="text">
    <string>Execute</string>
   </property>
  </action>
  <action name="actionExport_VTK">
   <property name="text">
    <string>Export VTK</string>
   </property>
  </action>
 </widget>
 <resources/>
 <connections/>
</ui>
```