CPS Final Report

Patrick O'Boyle & Lukas Negron

1. How to Run Our Code

We are using Mathematica 11.3, but it will also work on the newest version.

A new demo file contains different network setups to showcase the features and mechanics implemented since Report 2. First, go to our Project Code folder, open up the **Project2.5.nb** file, and evaluate all sections except the "Start-Up" section at the bottom. Evaluate the initialization cells.

Open the following notebooks:
NegronO'BoyleFinalReport/Project Code/**NewDemos**
NegronO'BoyleFinalReport/Project Code/**DemoModelView**

DemoModelView is a notebook that contains a readout of the structure of the current network. NewDemos contains many different demos that showcase features of our project.

How to run the demos:
Each demo has an orange heading with the demo number and a description. Within the demo cell are commented sub cells that deal with building/processing the network.

To see what the demos are doing, execute a sub cell, and then alt+tab so that you can see the DemoModelView file. After you see how the network has changed, alt+tab back to the Demo file and execute the next sub cell. Once you have executed all the sub cells for a particular demo, execute the **resetMasterPathList** function in the DemoModelView file to reset the structure of the network. Resetting the masterPathList is important because different demos showcase different networks.

If you want to see a readout of the basic structure of the current network, execute the **updateSimplePaths** function in the DemoModelView notebook. This shows the name of the path, the name of the previous path, and the name of the next path. To see a cleaned-up full readout of the network, execute the **printMPL[]** command. We tried to integrate these different functions under a single dynamic with parameter passing for simplicity, but couldn't figure it out.

Demos build upon previous demos

The demos tend to use commands/concepts established in previous ones. Previous demos and the instructions for running them are in the Report 2 submission.

Why we have two different files for testing the demos

We originally had the masterPathList dynamic view at the top of the demo file, but as the file got larger, scrolling became annoying, so we decided to split the view of the structure into the DemoModelView notebook so we could switch to it using alt+tab from the Demo notebook.

NEW INSTRUCTIONS FOR WORKING UI:

Warning: It is highly recommend to have worked through the Report 2 demos along with the new demos before using the UI. You aren't required to enter any commands; everything is done through a user interface, but it gives you a background for how everything is working.

After running through the NewDemos file, go to the Project2.5 notebook and run the "Start-Up" section in order to boot up the UI.
There's a ton of network characteristics you can modify during and after the build phase. Here's a series of instructions for a sample build with explanations to help explain what's going on. Restarting the program manually is done by re-executing "Start-Up".

When you hit start, the first thing you'll see is a setup question. Choose general. The setup will take longer but the general setup has more physics to tinker with such as fully customizable pumps and pressurized reservoirs. The dialog popups you will start seeing contain important information about how to set a part up and what the information means.

Next you have to define the input reservoir. Enter {0.5, 0.3, 10000, 1000, 900} into the box. 10000 pascals seems like a lot but it's not, because they're usually used in low-pressure situations. Pascals are used because everything is nicer when it's all in SI units.

Choose "Sure" for the pump. If you select "Nah", that means you'll have a gravity based system. Making a gravity based system means you have to be more careful. If your system is gravity-based but you're transporting water upwards, fluid transfer won't work. Gravity based systems must go down.

Enter {0.75, 1000, 25} for the pump information. Typical efficiencies are 0.7-0.8 so 0.75 is reasonable.

The view is about to be created. A box that displays the currently selected sequence just opened. A dialog popup also appeared for setting up sequence1, the origin sequence. Enter the height of seq1 in meters. 10 is a good starting height.

Now, there should be a black menu, a green menu, the popup with the currently selected sequence, and the view window. The black menu will be replaced with a white one later on after you hit the “Done” button. The green menu is always present. Let’s talk about the buttons in the green menu. Save and load are self explanatory and work for the session you’re in. If you want to load a save, restart the program using reset. You’ll be taken through the initialization phases again but these don’t matter. You can just put 1’s in for all the values to quickly get back to the green menu. If you then hit load, it brings back the network you saved earlier. Main menu and reset do exactly what they say, same with show global properties display button. Show sequence properties pulls up the properties of the selected sequence. You select sequences by clicking the buttons in the view.

Show sequence properties is extremely important. It lets you look at important information like flow rate, velocity, and what parts are contained inside the sequence.

Seq1 is selected. Click the attach sequence button to attach a sequence to it. Now you need to enter its height. Enter 15 meters. The attach sequence button is on the black menu. The black menu is used to build the network.

Be careful not to loose track of the white, green, and black menus. They can’t be alt-tabbed to. They usually end up under other windows you may have up.

Click on the seq2 button in the view to select it. Now attach a split. Enter {{1,1,1}, {True}} to create one seq above the split, a continuation, and a seq below the split. {True} will merge the first pairing (top and the middle). Now the heights of the new sequences, starting from the top, have to be entered. Enter 20, 13, and 16 for the height prompts. Now we already have a fairly complicated pipe structure! 20 is the height for the upper sequence you created off of the split. 13 is for the lower. You don’t input a height for the middle sequence because it’s the same height as the lead-in sequence. The help button in “attach split” doesn’t do anything.

Now, start inputting parts. Select seq1 and click the “add pipes” button to input a pipe of any size you wish. (something under 1 would make sense. A 1 meter pipe is very large). Now, use the seq buttons in the view to start adding pipes, valves, and gates to the rest of the sequences.

Seq6 and seq8 will be where the outputs go. Select seq6. Push “Attach output reservoir” and configure it as you wish or enter {0.3,0, 1, 3, 0}.Then do the same for seq8. Note- if you pressurize a reservoir, the density of the upper fluid shouldn’t be 0. Instead, it should be some density that’s lower than the density of the fluid you’re pumping throughout the system. Ex: The bottom fluid density you entered earlier for the input reservoir is 1000 (water). You can check this by selecting seq1 and clicking the display seqs properties button. Now, if you have a pressurized output container, the density of that fluid (upper fluid) should be less than 1000.

If ALL the sequences are filled with atleast 1 pipe/valve/gate, you can go ahead and click the “Done” button to move into the next phase. It won’t work if these criteria aren’t met.

When you hit done, the system is processed and a white menu pops up with new options (black menu is gone now). If you want to modify the gauge pressure for seq8’s reservoir, click the white “modify gauge pressure” button and enter {“seq8”, 1000} to set the gauge pressure to 1000 Pascals. Any command that asks you to manually enter a sequence name will require you to enter it as a string. It’s important that you enter {“seq8”, 1000} and not {seq8, 1000}.

If you put in a gate or valve, select the sequence of where you put it and click “Show seq properties” to determine what index that gate or valve is at in the parts list. For a vertical readout of {P, 5m}, {P, 5m}, {G, 0, 0} the gate is at index 3. Knowing the index is important if you want to open/close the gate/valve.

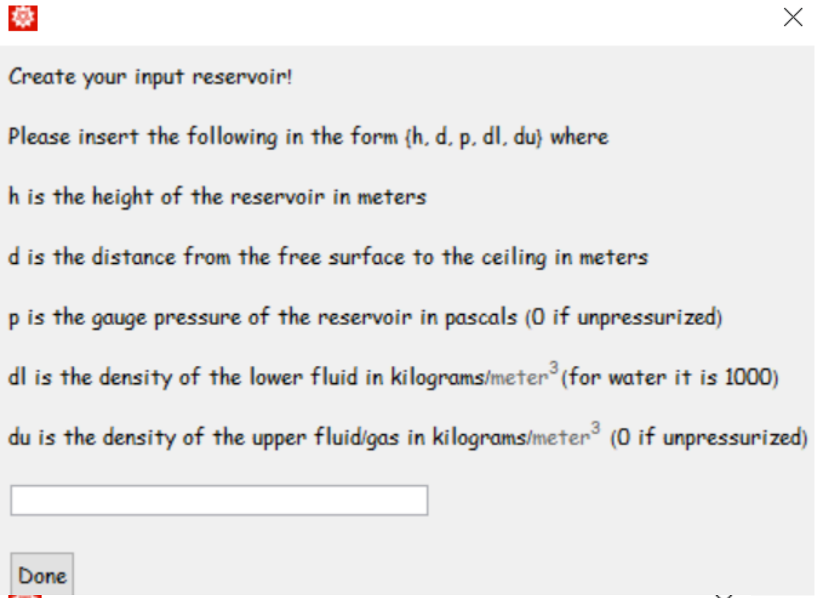
The change comp diameter button in the white menu operates on valves and gates. If you give it the index of a gate, it will close it. For a valve, you will set the new diameter, which must be less than the original max diameter. If you accidentally give “change comp diameter” an index for a pipe, nothing will happen.

If you want to transfer fluid, you input the seconds you want to pass. Because you transferred fluid, you probably want to see an output reservoir fill up. You do this by clicking on the “Show sequence properties” button to see the current volume of the reservoir. The output reservoir will be in the sequences end reference, shown in the properties. The current volume of an output reservoir is in index 5 of the “GenOut” reservoir list. The other “meters cubed” number designates the max volume, which will prevent you from transferring more than that amount of water. So if you want to see fluid transfer in action, you select a sequence with an output (a purple disk). In this case, seq6 and seq8 have outputs. Then you click “display sequence properties”. Now you can look at the 5th index of the GenOut reservoir list to see the current volume of water. After that, click the fluid transfer button to move water. Then, you can click “display seq props” again to see the change in volume.

An important feature to note is the network CAN be changed between water transfers. You can use the “change comp diameter” button to close a gate/valve at a given index for a selected sequence. Then the system will be reprocessed. This means that if you close a gate that’s inside a sequence connected to an output, and you try to transfer fluid, no water will fill the output in that sequence. Valves and gates can be placed anywhere, so you’re able to cut off certain parts of the network if you decide to place them in sequences that come before merges.s

2. Project Description

Fluid Mechanics for English Majors is an interactive plumbing network designer. The user defines the network structure, inserts parts, and then modifies those parts to observe changes within the system based on fluid mechanics processes. The project is essentially a physics simulation that the user builds from the bottom-up.



Create your input reservoir!

Please insert the following in the form {h, d, p, dl, du} where

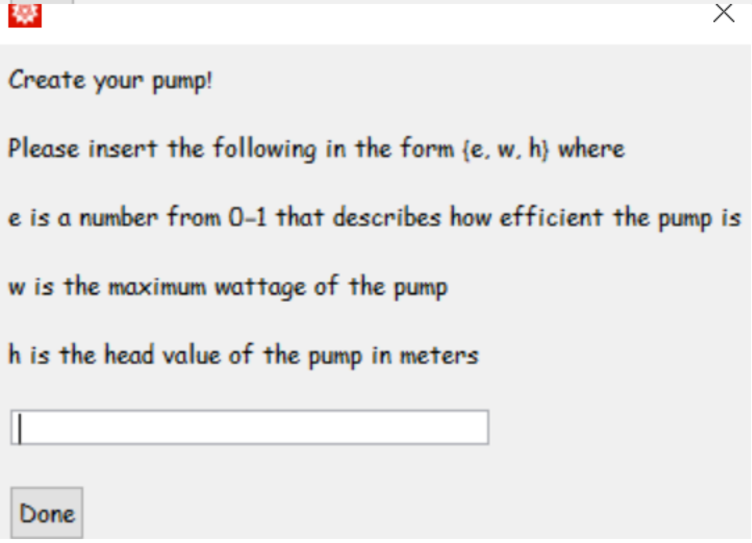
h is the height of the reservoir in meters

d is the distance from the free surface to the ceiling in meters

p is the gauge pressure of the reservoir in pascals (0 if unpressurized)

dl is the density of the lower fluid in kilograms/meter^3 (for water it is 1000)

du is the density of the upper fluid/gas in kilograms/meter^3 (0 if unpressurized)



Create your pump!

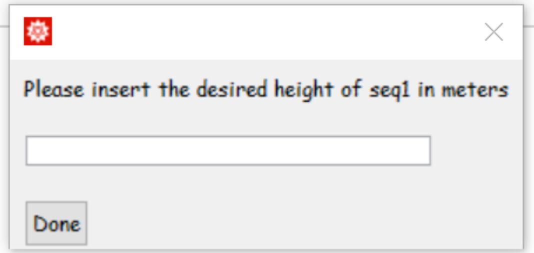
Please insert the following in the form {e, w, h} where

e is a number from 0-1 that describes how efficient the pump is

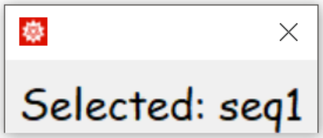
w is the maximum wattage of the pump

h is the head value of the pump in meters

After hitting the start button on the menu screen, the user selects either “General” or “Infinite” for the network setup. The output reservoirs (where water is transferred to) in an infinite setup are unbounded, and the general setup features more advanced reservoirs and pumps that allow for more interesting manipulation. The screens above show input reservoir creation (the reservoir the water starts off in) and pump creation. The values assigned to the input reservoir and pump are important because they determine the starting flow rate into the system (how much water enters the system per second). The input reservoir and pump configurations also influence pressure calculations and if the pump is powerful enough to successfully distribute water into the network. Your pump may be too weak, especially if the network is very tall, or if the output reservoirs are heavily pressurized, adding resistance to flow. It is important to note that the pump is optional; systems can be based on gravity. However, the same principle applies. Depending on the structure of the network, the force of gravity may not be enough to transfer fluid. Another important observation is the line “density of the lower fluid” in the “Create your input reservoir” image. The fluid you pump into the system isn’t necessarily water. You have the freedom to use any fluid, as long as you enter its correct density.



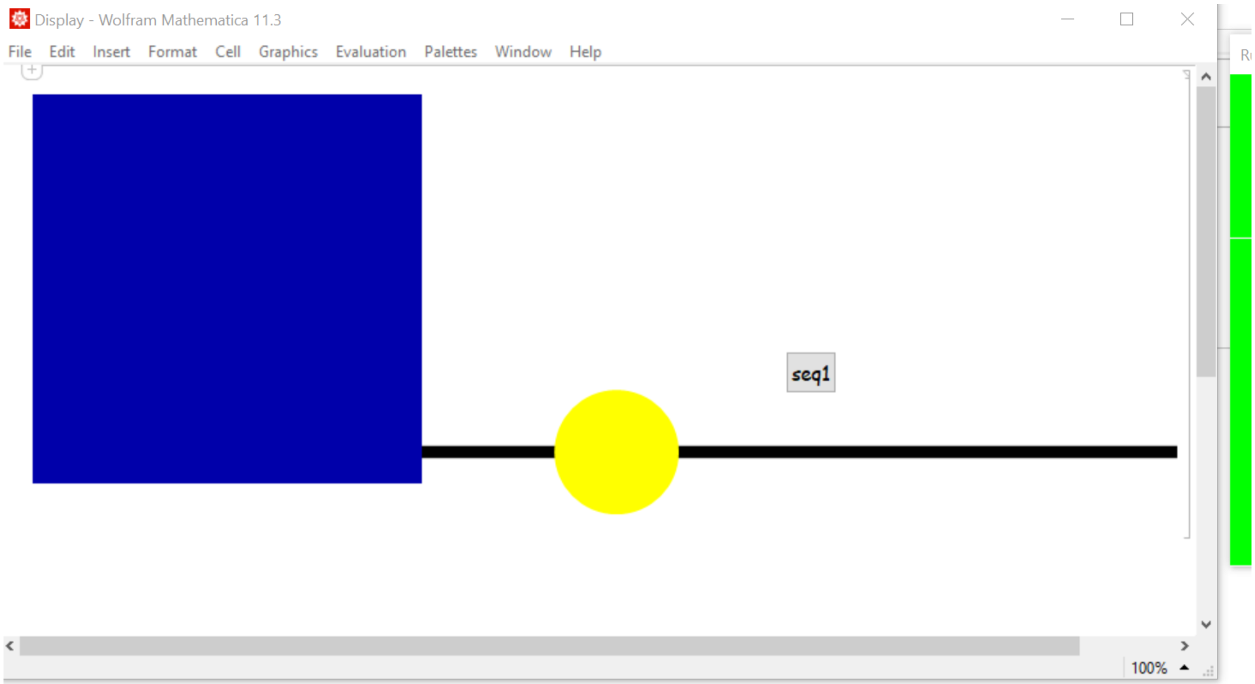
Please insert the desired height of seq1 in meters



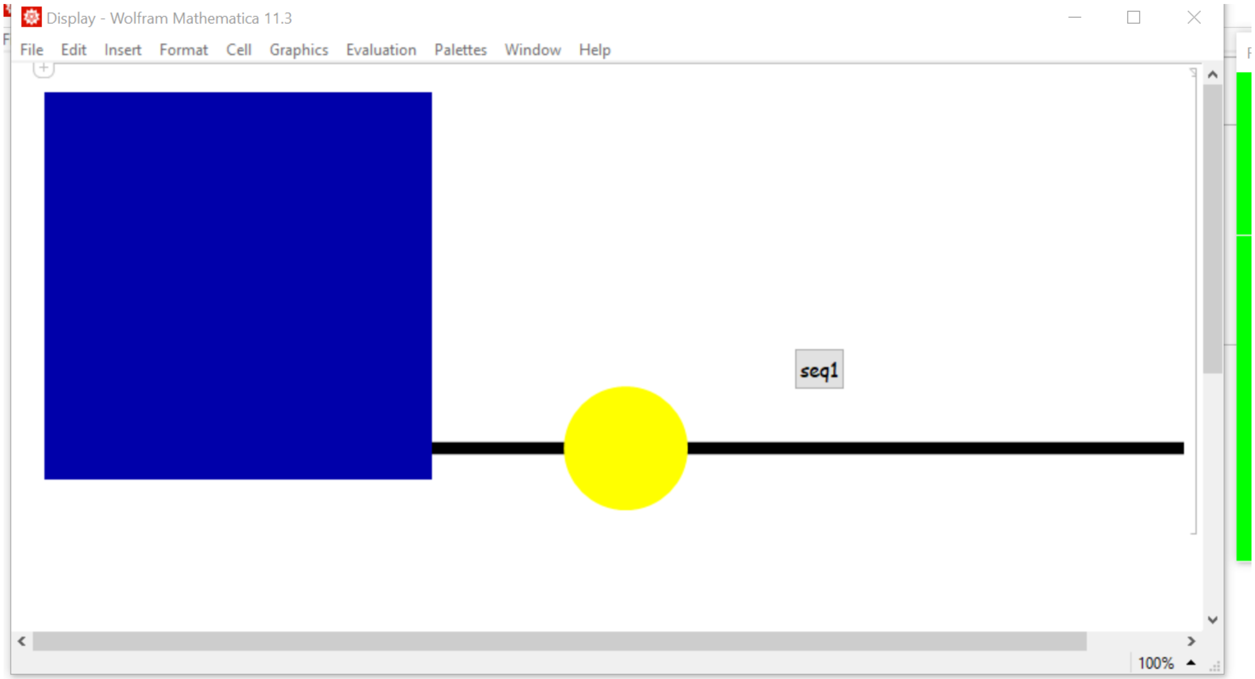
Selected: seq1

Now the interface is beginning to appear. Seq1, the origin, contains the optional pump and customized input reservoir. A tab is also open that displays the currently selected sequence. This is important because buttons such as “attach sequence”, “attach split”, and “add pipes” operate on the currently selected

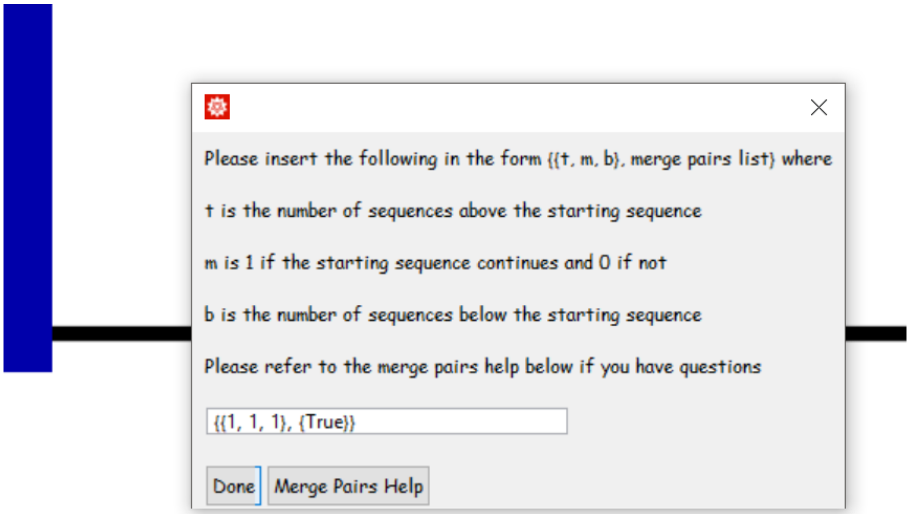
sequence.


Display - Wolfram Mathematica 11.3
File Edit Insert Format Cell Graphics Evaluation Palettes Window Help
seq1
100%

Here is the initial view. The black menu shown on the right is a list of build options that allows the user to add pipes, valves, and gates to sequences. The structure of the overall network is decided by the attach sequence and split commands. Attach output reservoir will put an output reservoir at the end of a sequence and pull up a menu that allows the user to customize it. When the build is finally done, the black menu will be replaced by a white menu that allows for modifications to the build.


Display - Wolfram Mathematica 11.3
File Edit Insert Format Cell Graphics Evaluation Palettes Window Help
seq1
100%

The green menu doesn’t go away. It allows you to save the current setup and this save persists for the duration of your Mathematica session. Main menu takes you back to the main menu, and reset scraps everything, starting off from the very beginning where you were initializing the setup. Show global properties lists environmental properties important for calculations. Show sequence properties is the most important button on the green menu because it allows you to see how the structure changes.


Please insert the following in the form {{t, m, b}, merge pairs list} where

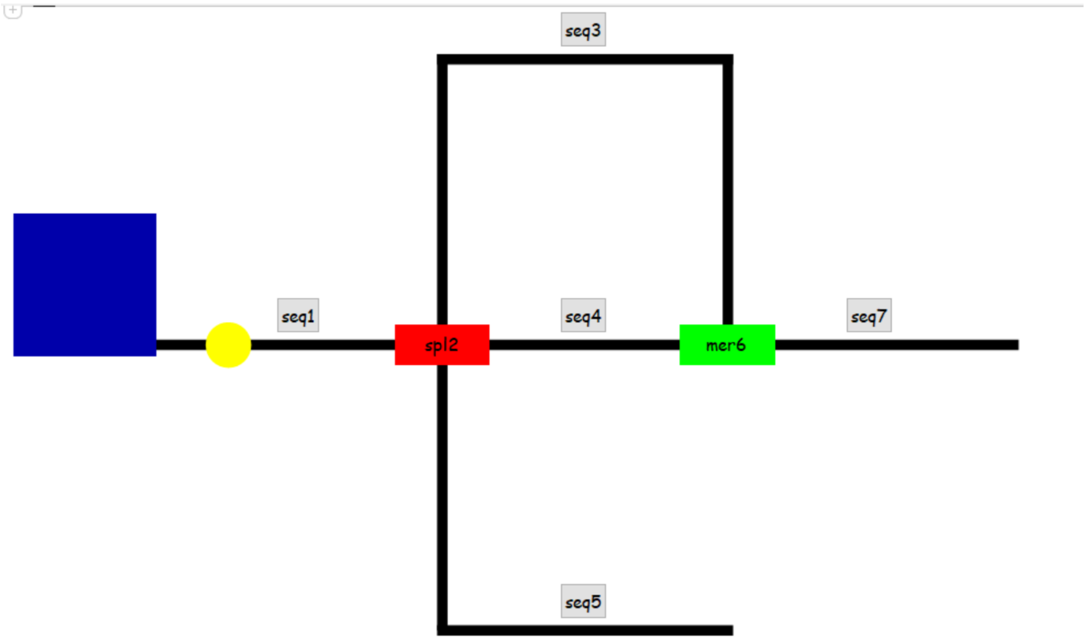
t is the number of sequences above the starting sequence

m is 1 if the starting sequence continues and 0 if not

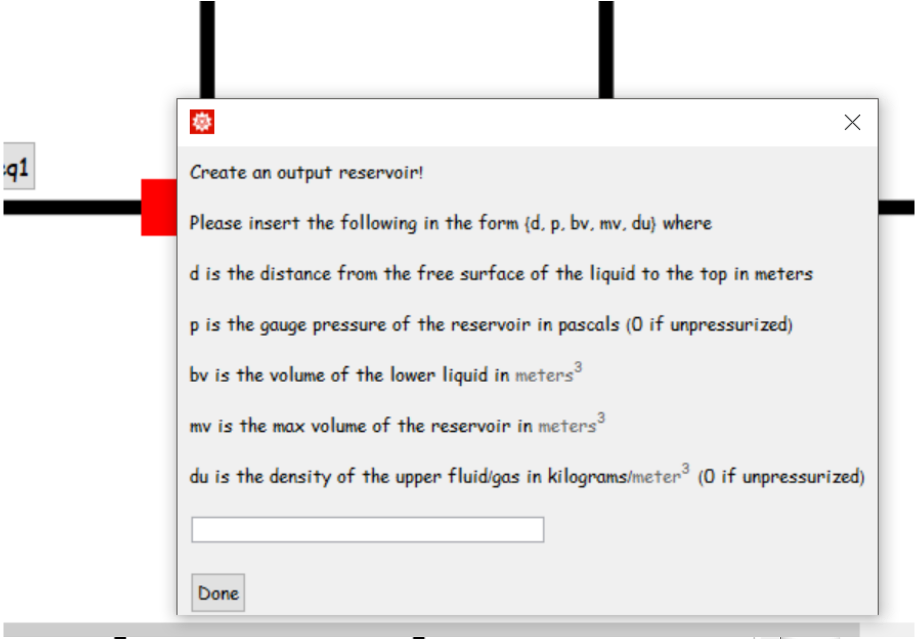
b is the number of sequences below the starting sequence

Please refer to the merge pairs help below if you have questions

{{1, 1, 1}, {True}}



The above images show split creation and the view that is produced after the split is attached and the branches are configured (this is done by dialog popups). By adding configured sequences and splits, the user is able to design complicated networks fairly quickly. The explanation for {t,m,b} is found in the dialog above. Merge pairs is a list of boolean expressions that allows the user to connect any combination of adjacent pairs of sequences that come off a split. In the case above. The first pair (seq3 +seq4) are being merged because the value is True. The view reflects this.



Create an output reservoir!

Please insert the following in the form {d, p, bv, mv, du} where

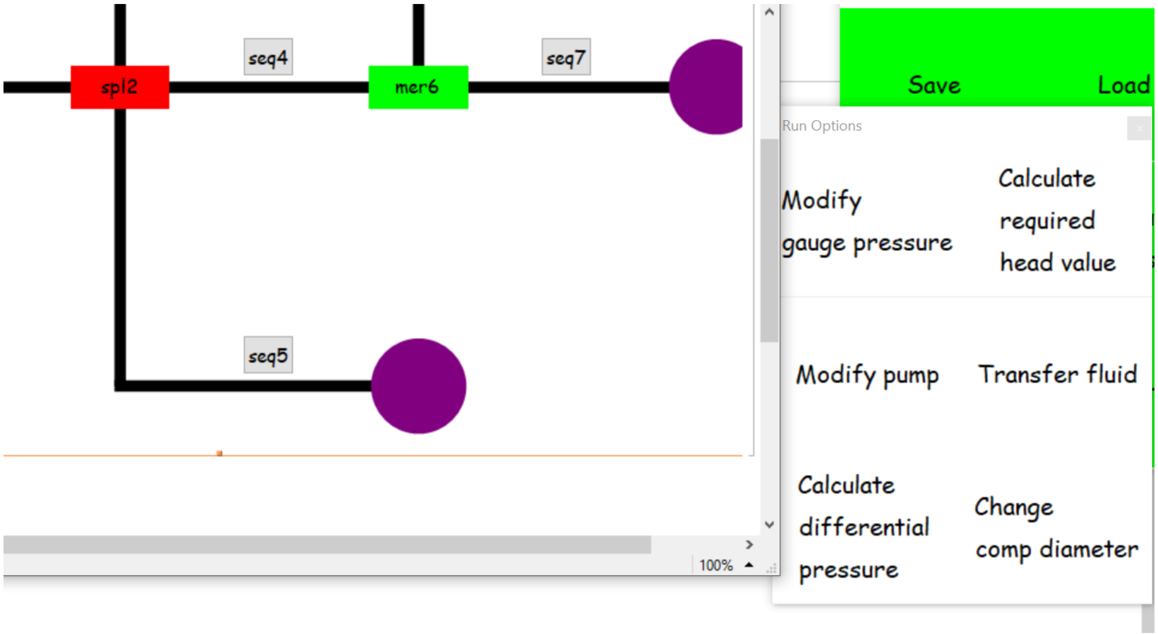
d is the distance from the free surface of the liquid to the top in meters

p is the gauge pressure of the reservoir in pascals (0 if unpressurized)

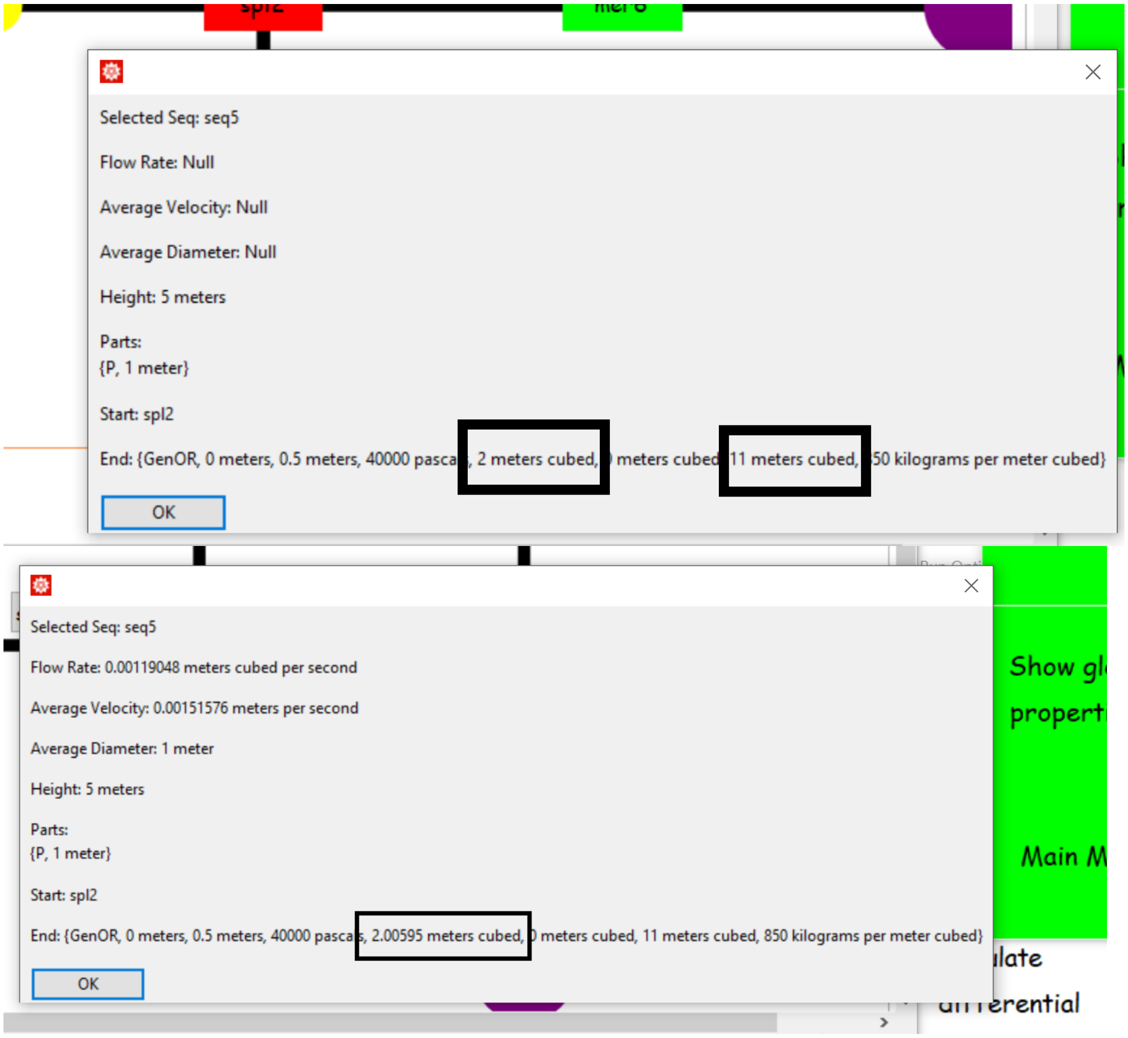
bv is the volume of the lower liquid in meters^3

mv is the max volume of the reservoir in meters^3

du is the density of the upper fluid/gas in kilograms/meter^3 (0 if unpressurized)



In addition to pipes, gates, and valves, output reservoirs can be mapped to sequences. These are customizable and will fill up with water. The properties of the output reservoir will affect how the network is processed. After the network is filled up with parts, it can be finalized by clicking the “Done” button. Clickign done will replace the black menu with the white menu. The system is processed, physics calculations are applied to determine properties such as flow rate and velocity, and the network will be processed each time a new change is made to it. This means changes can be made on the fly, even after fluid has been transferred. The white menu will allow you to modify your pump, find differential pressure (which will tell you if there are pressure increases or drops between points on the network), change comp diameter (modifying gates and valves), modify gauge pressure (pressurize/depressurize reservoirs), and transfer fluid from the input reservoir to the outputs. Additonally, the required work capacity of the system is calculated and returned by the calculate required head value button. Your pump’s head value (work capacity) must match the system’s required head value in order for fluid transfer to work. For gravity-based systems, the required head value must be less than or equal to 0, or else the transfer fluid button will notify you that your pump is too weak.



Selected Seq: seq5

Flow Rate: Null

Average Velocity: Null

Average Diameter: Null

Height: 5 meters

Parts:
{P, 1 meter}

Start: spl2

End: {GenOR, 0 meters, 0.5 meters, 40000 pasca, 2 meters cubed, meters cubed 11 meters cubed, 50 kilograms per meter cubed}

Selected Seq: seq5

Flow Rate: 0.00119048 meters cubed per second

Average Velocity: 0.00151576 meters per second

Average Diameter: 1 meter

Height: 5 meters

Parts:
{P, 1 meter}

Start: spl2

End: {GenOR, 0 meters, 0.5 meters, 40000 pasca, 2.00595 meters cubed, meters cubed, 11 meters cubed, 850 kilograms per meter cubed}

The above images are the displayed sequence properties of seq5 before and after fluid transfer. Before fluid transfer, 2 meters cubed of liquid was already in the output reservoir. After the user clicks the “Fluid transfer” button in the white menu and determines the time of the transfer (in seconds), the volume of fluid entering all the outputs is determined. The settings of gates and valves in the system will determine if flow is cut off or not. Flow can also be constricted based on how parallel networks are processed. A sequence with a greater average diameter will have more water enter it in comparison to the rest of the sequences in the parallel network. A parallel network is essentially all the branches that come off of a split.

3. Research- Algorithms/Formulas

The following formulas are implemented in our project to carry out physics calculations and processing. These formulas are implemented and combined in different ways to accomplish certain tasks. For example, Bernoulli’s is used along with reservoir pressure to figure out how powerful a given pump must be to successfully transfer water.

References are listed in #9, the references section.

Cross-sectional area:
The cross-sectional area of a pipe with diameter d can be calculated with the formula pi/4 $*d^2$.
So when d=4 meters, the cross-sectional area will be 12.5664 m^2.

Flow rate, cross-sectional area, velocity:
Q=AV. Q= flow rate, A= cross-sectional area, V= velocity
If the cross-sectional area is 2 $m^2$and the velocity is 3 m/sec, then the flow rate is 6 m^3/sec.

Pump head-value (capacity to do work):
This is an application of Bernoulli's Principle.
The equation below is a rearrangement of Bernoulli's Principle to solve for headValue. A and B are reservoirs.

$$\frac{\text{pressureB}}{\text{densityLiq}*\text{gravity}} + \text{heightB} - \frac{\text{pressureA}}{\text{densityLiq}*\text{gravity}} - \text{heightA}=\text{headValue}$$

pressureA/B refers to the pressure exerted on the free surfaces (fluid surface level) of reservoirs A and B. densityLiq is the density of the fluid you are pumped through the system.

For a reservoir A exposed to the atmosphere 5 meters above the ground, and a reservoir B exposed to the atmosphere 10 meters above the ground, and using water at standard density as the fluid, the headValue of the pump will be 5 meters. (Exposure to atmosphere makes the pressureA/B variables 0, gauge pressure would be used otherwise).

Calculating pump flow rate

$$\frac{\text{watts}*\text{efficiency}}{\text{densityOfLiquid}*\text{gravity}*\text{hPump}} = \text{flowRate}$$

efficiency is typically between 0.7 and 0.8.
For a pump that is supplied with 1000W, pumps water at a standard density, a pump efficiency of 0.75, and a head value of 5 meters, the flow rate will be 0.0153061 m^3/sec

Calculating flow across members in a split
The flow rate for each sequence that comes off a split is based off the proportions of each sequences average diameter.
For sequences with average diameters 2m, 3m, 4m, and 5m. The resulting flow rates will be 10/7 m^3/sec, 15/7 m^3/sec, 20/7 m^3/sec, 25/7 m^3/sec. First, you find the unit flow by dividing the original flow in by the total of the diameters. Then you multiply the unit flow value by each diameter to find the flow rate across that member.

Differential pressure

Application of Bernoulli’s Principle.

You can rearrange Bernoulli’s Principle to find the difference in pressure.

$$\frac{\text{densityLiq} * \left(\text{v2}^2 - \text{v1}^2\right)}{2} + \text{densityLiq} * \text{gravity} * (\text{height2} - \text{height1})$$

Example- The fluid being pumped is water at standard density. Water flows through sequence 1 at a velocity of 5m/sec and sequence 2 at a velocity of 10m/sec. Sequence 1 is at a height of 2 meters and sequence 2 is at a height of 2 meters. The resulting differential pressure in pascals is 37500. This means pressure increasing from sequence 1 to sequence 2 by 37500 Pa.

Calculating pressure on a reservoir’s free surface

densityOfUpperFluid*gravity*ceilingDis+gaugePressure

The density*g*ceilingDistance part of the equation finds the pressure that the upper liquid exerts on the lower liquid in pascals. This means if we have oil resting on top of water in a reservoir, we can figure out the pressure exerted on the water that is being pumped into the system by adding the oil’s pressure to the pressure set by the user for the gauge.

Example- The fluid resting on top of the fluid being pumped is water at a standard density. The distance to the top of the reservoir is 1 meter, and the gauge pressure is 1000 Pascals. The total pressure exerted on the bottom fluid’s free-surface is 10800 Pa.

Calculating flow rate for a reservoir that uses gravity (no pump)

V=Sqrt[2*g*h]

For a reservoir with a 2 meter height to the fluid’s free surface, the velocity of fluid exiting the reservoir is 6.26099m/s. This value can be multiplied by the cross-sectional area of the pipe that the fluid travels out of to figure out the flow rate into the rest of the system.

Water of different temperatures merging- 2nd increment

You can average the temperatures of two merging members to figure out the resulting temperature. So a a sequence with 10 degrees C merged with another sequence 16 degrees C will result in the temperature of the sequence they converge to being 13 degrees C.

Water coming out of a heater- 2nd increment

watts = density*flowRate*specificHeat*(T2-T1)

specificHeat = (T2+T1)/2

For water going into a heater with a constant kinetic energy, you can use the above expressions. Watts is the electricity supplied to the heater, density is the density of the fluid pumped through, and specific heat quantity is needed for anything dealing with measuring changes in temperature. So if water at a standard density is being pumped through a heater supplied with a temperature of 10 degrees C and a target temperature of 20 degrees C for water with a flow rate of 1 m^3/sec, you will have to supply 150,000W to the heater.

4. Files

A pictures folder in the notebook directory stores the menu screen image for the UI. Other than that, we have no file structure. One could be implemented for persistent saving, however.

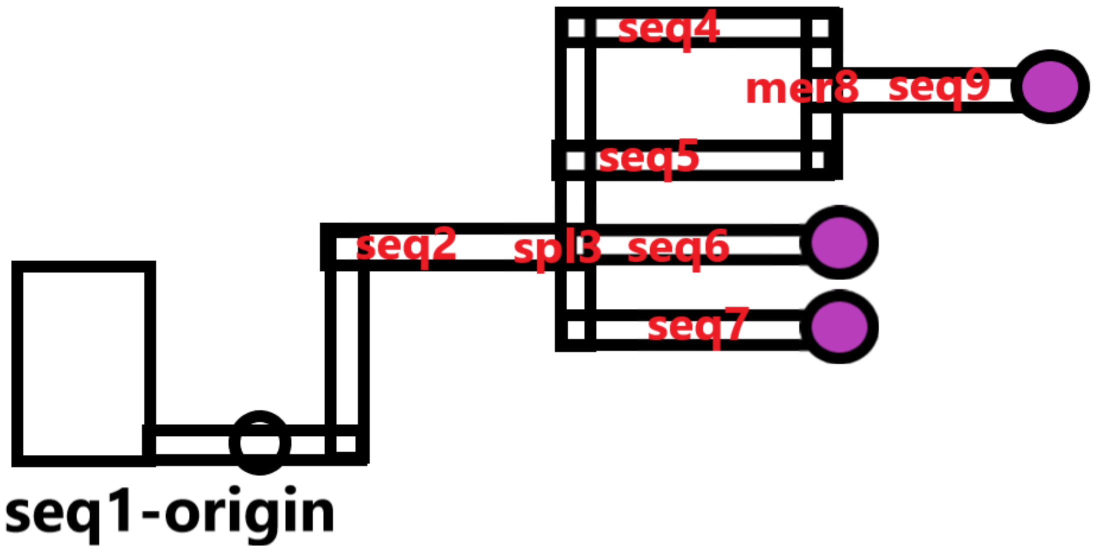
5. Data Structures

Our project uses lists and maps.

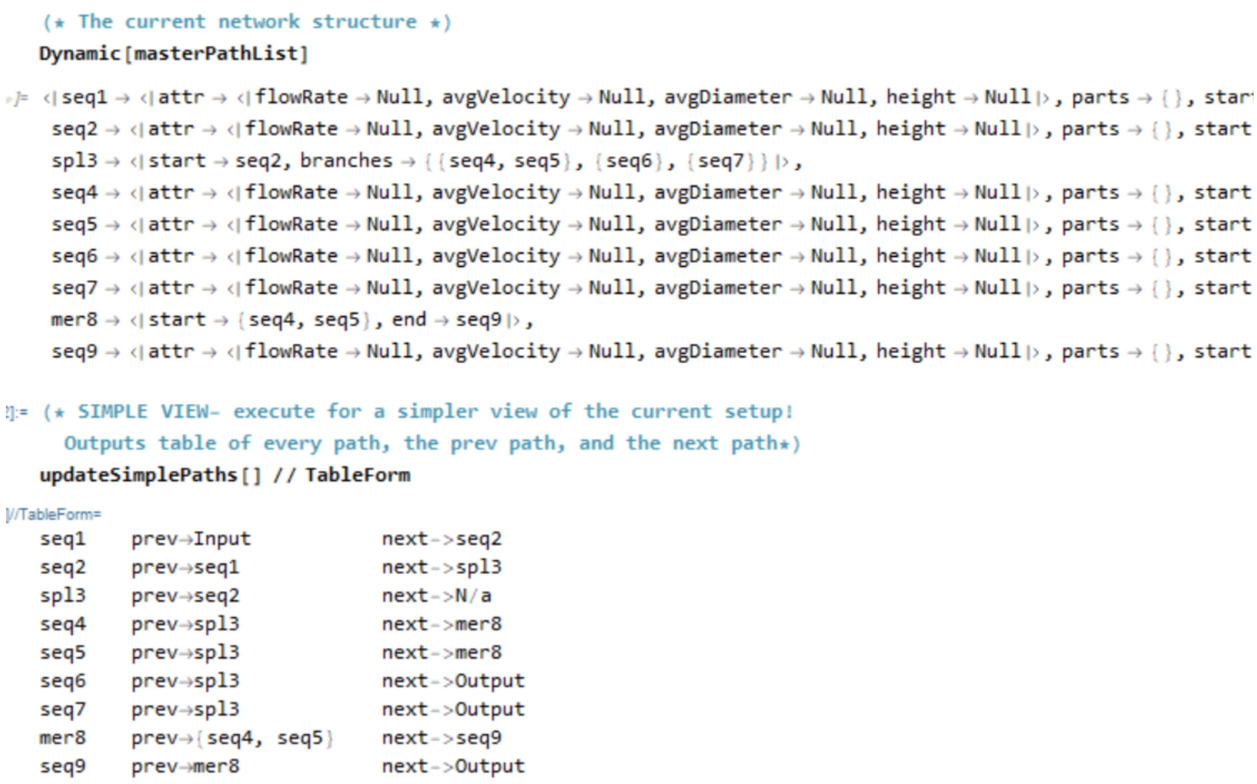
Maps:

We figured out that we needed a doubly-linked structure to represent the piping network, because paths have inputs and outputs. There are very complicated linked-list structures posted on StackExchange, but we discovered that we could use associations to achieve the same results. Associations are composed of key-value pairs. An example of how we use maps can be seen by going through the “Background Info” demo in our Report2Demo file. Here’s the demo output so you don’t have to re-run it:

*diagram added for better explanation



The plumbing network diagram above is represented by the simplified model shown in the image below. seq1 points to seq2 in its end reference. seq2 points to seq1 in its start reference and seq2 points to spl3 in its end reference. seq4, 5, 6, and 7 are all branches of spl3, so they are mapped to spl3's branches key. mer8 combines seq4 and seq5, so its start pointer is of course {seq4, seq5} and a sequence must come off of a merge, and that sequence is seq9, which mer8 has a reference to in its end pointer. Though output reservoirs aren't mapped, they would be inserted where the null mappings are for the end references in seq6, 7 and 9. This is how the relationships between the sequences are defined. Splits and merges don't contain any important attributes or parts; they define how the sequences are linked together for the overall network. The linking of the structure is important because each sequence has an output flow rate that is conditionally processed based on the next structure in the
map.



```
   (* The current network structure *)
   Dynamic[masterPathList]

]= <|seq1 → <|attr → <|flowRate → Null, avgVelocity → Null, avgDiameter → Null, height → Null|>, parts → {}, star
    seq2 → <|attr → <|flowRate → Null, avgVelocity → Null, avgDiameter → Null, height → Null|>, parts → {}, start
    spl3 → <|start → seq2, branches → {{seq4, seq5}, {seq6}, {seq7}}|>,
    seq4 → <|attr → <|flowRate → Null, avgVelocity → Null, avgDiameter → Null, height → Null|>, parts → {}, start
    seq5 → <|attr → <|flowRate → Null, avgVelocity → Null, avgDiameter → Null, height → Null|>, parts → {}, start
    seq6 → <|attr → <|flowRate → Null, avgVelocity → Null, avgDiameter → Null, height → Null|>, parts → {}, start
    seq7 → <|attr → <|flowRate → Null, avgVelocity → Null, avgDiameter → Null, height → Null|>, parts → {}, start
    mer8 → <|start → {seq4, seq5}, end → seq9|>,
    seq9 → <|attr → <|flowRate → Null, avgVelocity → Null, avgDiameter → Null, height → Null|>, parts → {}, start

]:= (* SIMPLE VIEW- execute for a simpler view of the current setup!
     Outputs table of every path, the prev path, and the next path*)
   updateSimplePaths[] // TableForm

]//TableForm=
   seq1    prev→Input              next->seq2
   seq2    prev→seq1               next->spl3
   spl3    prev→seq2               next->N/a
   seq4    prev→spl3               next->mer8
   seq5    prev→spl3               next->mer8
   seq6    prev→spl3               next->Output
   seq7    prev→spl3               next->Output
   mer8    prev→{seq4, seq5}       next->seq9
   seq9    prev→mer8               next->Output
```

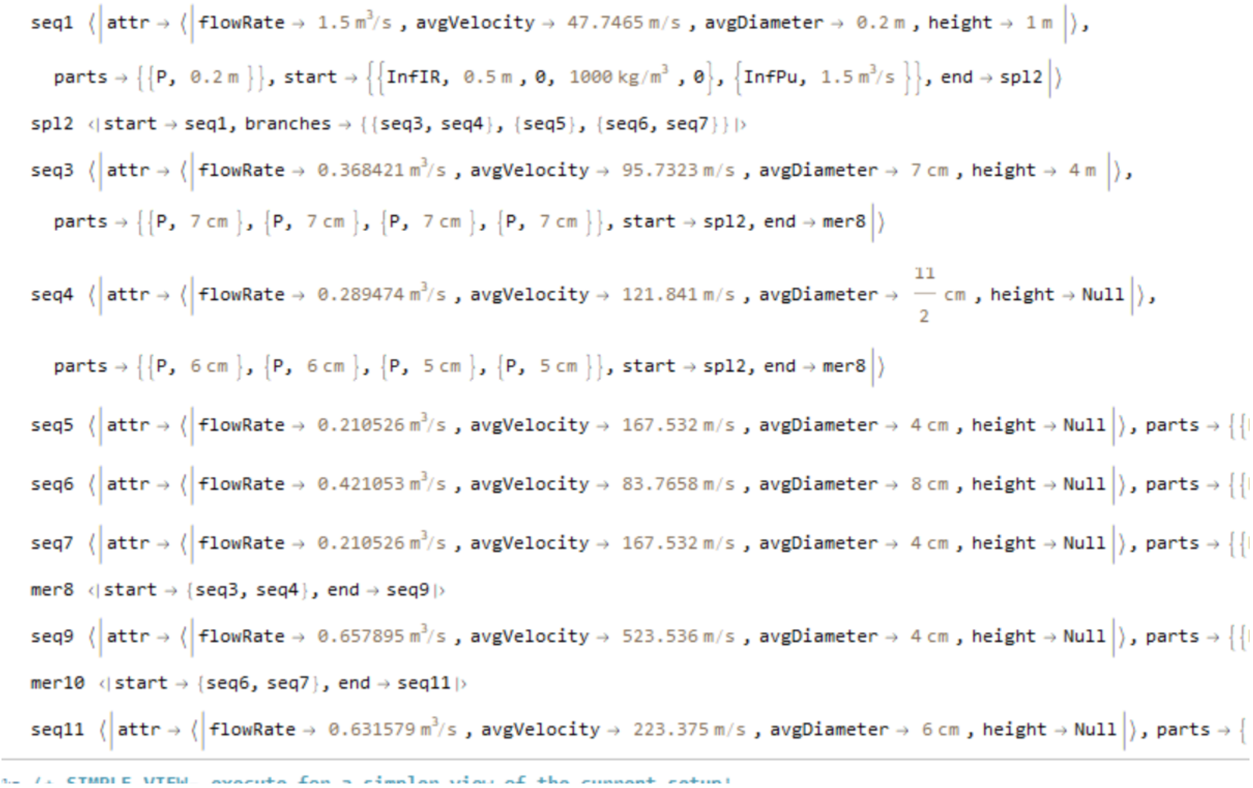
;

masterPathList contains a list of associations. These associations are our paths (seq1, seq2, and so on...). This allows us to quickly grab path information by running the command masterPathList[“-pathKey”]. Paths refer to another sublist of associations that contain path characteristics, such as where it starts, the parts it contains, and the environmental attributes like flowRate. If we wanted to access a sequence’s partsList, we could simply run masterPathList[“seqExample”, “parts”]. This is much simpler and more readable than using part commands on one gigantic list full of sublists. Sequences, splits, and merges are the path types, and they all have slightly different structures within their association list. These differences are shown in the above image and are explained in the comments for the functions that create those path types. These comments can be found in Project.nb in the Model section under the Path Creation subheading.

We also use an identical structure for our globalConstants map. This allows us to easily retrieve and change environmental constants like gravity that we use in our calculations. The masterPathList and globalConstants structures can be found in the Project.nb file at the beginning, under the headings “Constants Dynamic Display” and “MasterPathList Dynamic Display + simplePaths View”.

Lists:

We use standard lists within our masterPathList association map list. We used them to help model merges, splits, and components. Here’s the output from Demo4 as an example:



```
seq1 ⟨|attr → ⟨|flowRate → 1.5 m^3/s , avgVelocity → 47.7465 m/s , avgDiameter → 0.2 m , height → 1 m |⟩,
  parts → {{P, 0.2 m }}, start → {{InfIR, 0.5 m , 0, 1000 kg/m^3 , 0}, {InfPu, 1.5 m^3/s }}, end → spl2|⟩
spl2 ⟨|start → seq1, branches → {{seq3, seq4}, {seq5}, {seq6, seq7}}|⟩
seq3 ⟨|attr → ⟨|flowRate → 0.368421 m^3/s , avgVelocity → 95.7323 m/s , avgDiameter → 7 cm , height → 4 m |⟩,
  parts → {{P, 7 cm }, {P, 7 cm }, {P, 7 cm }, {P, 7 cm }}, start → spl2, end → mer8|⟩
                                                                                    11
seq4 ⟨|attr → ⟨|flowRate → 0.289474 m^3/s , avgVelocity → 121.841 m/s , avgDiameter → ── cm , height → Null|⟩,
                                                                                    2
  parts → {{P, 6 cm }, {P, 6 cm }, {P, 5 cm }, {P, 5 cm }}, start → spl2, end → mer8|⟩
seq5 ⟨|attr → ⟨|flowRate → 0.210526 m^3/s , avgVelocity → 167.532 m/s , avgDiameter → 4 cm , height → Null|⟩, parts → {{
seq6 ⟨|attr → ⟨|flowRate → 0.421053 m^3/s , avgVelocity → 83.7658 m/s , avgDiameter → 8 cm , height → Null|⟩, parts → {{
seq7 ⟨|attr → ⟨|flowRate → 0.210526 m^3/s , avgVelocity → 167.532 m/s , avgDiameter → 4 cm , height → Null|⟩, parts → {{
mer8 ⟨|start → {seq3, seq4}, end → seq9|⟩
seq9 ⟨|attr → ⟨|flowRate → 0.657895 m^3/s , avgVelocity → 523.536 m/s , avgDiameter → 4 cm , height → Null|⟩, parts → {{
mer10 ⟨|start → {seq6, seq7}, end → seq11|⟩
seq11 ⟨|attr → ⟨|flowRate → 0.631579 m^3/s , avgVelocity → 223.375 m/s , avgDiameter → 6 cm , height → Null|⟩, parts → {
(* SIMPLE VIEW, execute for a simpler view of the current setup!
```

;

Sequences contain a “parts” mapping that points to a list that contains components. Components are represented by a sublist. For example, in seq1, the pipe is represented by the sublist {“P”, 0.2m}. Index 1 of a component identifies its type. Because all of our current components have a diameter of some sort, we’ve been using index 2 as the standard index for the current diameter of the part. Our other components: valves, gates, input reservoirs, and output reservoirs follow this structure but add more to it. Valves and gates have an additional index for the max diameter. Input reservoirs and output reservoirs also contain indexes that refer to values used in physics calculations such as gaugePressure and the densities of the two liquids in the reservoir, if two liquids exist. More information on component list structure can be found in the comments in the Project.nb file under the Components section.

Splits contain a “branches” mapping that points to a partitioned list of created sequences that the split diverges into. The first partition is the list of sequences above the split. The middle partition will contain a sequence if there is on that continues straight from the split. The last partition is the list of sequences created below the split. spl2 in the above image is an example.
The “start” mapping for merges points to a list of sequences that converge. mer8 and mer10 in the above image are examples.

6. Solution Framework

Overview

We broke our project into two overall components: design and physics. Design includes anything related to the construction of the network. The physics component refers to the math that's applied to the design component.

We followed the Model-Controller-View principle as we designed the project. We began by implementing math functions (under the Calculations heading) so we could calculate important values like velocity through a pipe, differential pressure, pressure exerted on a free-surface for a reservoir, and so on. Then, after figuring out how to represent the piping network with a model, we started with the masterPathList as a global storage list for different sequences, splits, and merges. The creation of the different path types interacts with the calculations through the processing functions. The path types hold attributes such as flowRate, avgDiameter, and avgVelocity that are modified and updated by the processing functions. The processing functions make use of the math from Calculations in order to distribute flow, calculate velocity, and so on. There's also a Components section that includes our code for adding different parts like reservoirs, pipes, and valves to a chosen path in the model. Component characteristics like diameter, height, and output flow rate are read inside the model by the processing functions in order to carry out actions like assigning flow rate values, calculating velocity, finding differential pressure, and transferring water.

Important Functions

Dynamic[globalConstants]

Outputs a live display of the globalConstants list, which contains constants used in calculations. The globalConstant list can be easily added to and the variables can be modified to change the results of calculations. Gravity is used in many of the calculations and can be modified. However this feature is not included in the UI. calling globalConstants["gravity"] and setting it equal to a new value will change it and affect all the calculations that depend on it directly or indirectly.

Dynamic[masterPathList]

The masterPathList stores our model. The dynamic wrapping outputs a live display of the masterPathList, which holds the structure of the current network. As paths and components are added or modified, the display of the masterPathList will update.

mapNewInfInputReservoir & mapNewUnboundedOutputReservoir

mapNewInfInputReservoir will add an infinite input reservoir to the beginning of the network at the origin sequence. The reservoir has customizable properties that allow us to model gauge pressure, atmospheric pressure, and the affects of having a floating liquid above the fluid you're transporting.
mapNewUnboundedOutputReservoir will add an unbounded reservoir at the end map of a sequence. The reservoir has customizable properties like the infinite input reservoir, but has a changing volume and a variable that determines how the graduations will work. The number of lines and how much liquid each line designates can be customized, along with the size.

addInfPump

Adds an infinite pump to the origin sequence. The infinite pump has a customizable flow rate (parameter). This will determine the flow rate into the rest of the network.

addPipes

Adds a given number of pipes with a given diameter to a sequence. Affects a sequence’s average diameter and velocity. Parameters determine how many pipes are added and their diameters.

addValve

Adds a special type of pipe to a sequence that has a modifiable diameter. Affects a sequence’s flow out along with its average diameter and velocity. Parameters determine the maximum diameter and the current diameter.

addGate

Adds a special type of pipe to a sequence that can either be opened or closed. Affects a sequence’s flow out along with its average diameter and velocity. Parameter determines the maximum diameter.

createPathKey

The basis of our key naming system for the masterPathList. It takes the type of path it is and an optional number as parameters. When any path is created, whether it’s a merge, sequence, or split, this function is called to assign its name. Names are created in the form “type” + “count”, count referring to the current size of the masterPathList + 1. The optional number parameter allows us to set custom numbers for a key rather than having it use the size of the masterPathList.

createSeq, createSpl, createMer

These functions are the foundations for our model. All three path types have start and end mappings that allow the structure to be doubly-linked, which is important for processing. createSeq creates a key that’s mapped to a list of attributes and list of components. createSpl models a generalized split by creating a key that’s mapping to a partitioned list of sequences. createSpl also calls createMer to preemptively build merges for the partitioned sequences. createMer models generalized merges by reading truth values for merge pairs given to the createSpl function. Merges are mapped to a list of sequences that converge.

processSysMain

This is the umbrella processing function that takes care of reading, updating, and calculating of the flowRate, avgDiameter, and avgVelocity attributes for the network’s sequences. It handles the starting flow rate for systems with/without pumps and the transfer of flow through the system based on how the sequences, splits, and merges are combined. It also uses the changing avgDiameter of the sequences to calculate the average velocity of fluid through each sequence. processSysMain is the function that’s called whenever a change is made to the network that demands attributes to be updated.

findDiffPress

Finds the differential pressure between two sequences using Bernoulli’s Principle. The two sequences along with the density of the fluid travelling in them is given for the parameters. The function is used to see how pressure changes throughout the network.

fillUnOutput

Fills an unbounded output reservoir from an infinite input based on the flow into the output and time (parameters). It’s a fluid transfer function.

findHeadVal

Uses Bernoulli’s Principle to figure out how high a pump’s capacity for work needs to be in order to pump water throughout the network. It’s an umbrella function and uses the heights of the sequences along with the pressure from reservoirs applied to Bernoulli’s equation.

headValCheck

Works for gravity-based and pump-based networks. headValCheck looks at the required work capacity that the network demands for water transfer. If the pump/gravity is enough to successfully move water, true is returned. Else, false is returned.

calcFlowRateForPump

Uses a pump’s determined head value and supplied electricity to determine the starting flow rate for the network. This value is later compared in processing to determine if the pump is strong enough.

calcPressureReservoir

Calculates the pressure exerted on a reservoir’s free surface. Affected by gauge pressure, reservoir height, and whether or not a less-dense fluid is above the fluid entering the network. Needed for calculating a pump’s head value. Called when determining required head value. (A highly pressurized input reservoir helps the fluid move into the system!) The opposite is true for output reservoirs. (Pressurized output reservoirs will cause resistance to flow, which means the pump will have to work harder)

mapNewGenInputReservoir & mapNewGenOutputReservoir

Similar to the mapNewInfInputReservoir & mapNewUnboundedOutputReservoir functions but makes generalized reservoirs. The main difference is the generalized ones are pressurized and the density of the fluid applying pressure to the water has a user-defined density (parameters). mapNewGenOutputReservoir will have a maximum volume to cap off how much water can flow in.

addGenPump

Similar to addInfPump but adds a generalized one instead. The system flow rate will be dependent upon the flow rate of the pump, which will be affected by electricity, the pump’s head value, the density of the fluid, gravity, and efficiency (parameters).

fillGenOutputs + fillUnOuputs

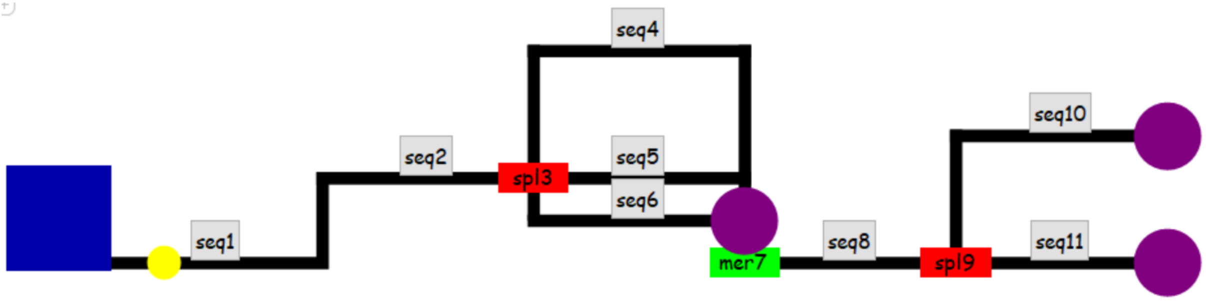
Fills generalized outputs or unbounded outputs, depending on if the network is an infinite or generalized setup. These are fluid transfer functions.

7. External Code

All of the code is original work.

8. Limitations

View:



Here is a special case that is technically an incorrect representation of the network. Because “layers” are used to determine the x-coordinates of where sequences are located, sometimes the representation is inaccurate. In a real network, the output reservoir from seq6 would be further back and not clipping into the pipes leading into mer7.

Exception Handling:
We implement a lot of exception handling that results in buttons doing nothing or telling you to change something if the setup isn’t correctly configured or if you gave it wrong input. However, because almost everything is customizable, it would take ages to implement proper exception handling for all the functions. Instead, when the user configures different components, they know what data to enter because we tell them in the pop up dialog. For example, when we tell the user to enter the height of a created sequence in meters, we do not expect them to enter in a string like “lsadkjflksdjf” or Null or {this, is, wrong}. We expect them to enter a number, because that’s what we tell them to enter. It is easy to break things if you went through the dialogs and entered strings and null values, but doing this doesn’t make sense if the user reads the text in the pop up.

Back-end Quantities:
In the back-end, it is critical that Quantity[number, “Units”] is used for anything when customizing components or setting up other system properties. One of the most commons bugs in the processing functions that had to be fixed is boolean expressions that compared a regular number to a quantity. The number 0 cannot be compared to Quantity[5, “Meters”], so the boolean expression wouldn’t evaluate, which interfered with processing at times. This means, if you use create or map commands in the back-end that install components, not using the quantity wrappings will result in many errors. In some demos (from Report 2 or the NewDemos file), if quantities aren’t used in the parameters related to physical properties for creating certain components, that’s because the focus of the demo isn’t processing. The demo is probably showcasing some new structural feature.

Difficulties:
Converting the model into an interface was an obstacle. After getting help, we decided to use buttons to select sequences, and arrays of buttons to carry out specified functions on them. The arrays of buttons would appear/disappear based on what phase of the build you’re on. For example, if you’re still designing the overall structure, you shouldn’t be able to calculate differential pressure.

9. References

Some textbook problems along with our notes from Dr. Chidurala are in the ResearchReferences folder.
The textbook we referred to is referenced below.

Dr. Chidurala, WKU Engineering. Email: manohar.chidurala@wku.edu

Textbook: “Fluid Mechanics Fundamentals and Applications”, Third Edition Cengel, Cimbala