Contents

Part I

Problem statement

Pipe networks can be analogous to Electrical Circuits. Therefore, it can be tempting to use electrical flow solvers for pipe networks.

1 Unit Comparison

Voltage is energy per unit charge unit wise. $\frac{J}{C}$ Joules per coulomb.

Whereas Pressure can be thought of as Energy per unit volume. $\frac{J}{m^3}$ Joules per meter cubed. That would be the unit of pascals.

$$(workDone)\ Joules = (pressure)\ Pa * (volume)\ m^3$$
$$(pressure)\ Pa = \frac{(workDone)\ Joules}{(volume)\ m^3}$$

Compare this to voltage:

$$(voltage)\ Volts = \frac{(workDone)\ Joules}{(Charge)\ Coulomb}$$

Likewise for volumetric flowrate, this is

$$(flowrate)\ m^3/s = \frac{(vol)\ m^3}{(time)\ s}$$

And for current,

$$(current)\ C/s = \frac{(Charge)\ Coulomb}{(time)\ s}$$

2 nonlinearities in flow resistance

However, the components exhibiting flow resistance often do not obey Ohm's law.

$$electrical\ resistance\ (\Omega) = \frac{V\ (Volts)}{I\ (Ampere)}$$

$$flow\ resistance = \frac{\Delta P\ (Pa)}{\dot{V}\ (m^3)}$$

For ohm's law, the ratio V/I reduces to a constant, but flow resistance often reduces to some expression.

If we were to use Fanning friction factor for pipe (cite perry's handbook)

$$f = \frac{\Delta P}{(\frac{4L}{D})\ \frac{1}{2}\rho u^2}$$

$$\Delta P = f(\frac{4L}{D})\ \frac{1}{2}\rho u^2$$

In Perry's chemical engineering handbook, the formula used for fanning's friction factor by Churchill is:

$$f = 2\left[\left(\frac{8}{Re}\right)^{12} + \left(\frac{1}{A+B}\right)^{3/2}\right]^{1/12}$$

Where:

$$A = \left[2.457 \ln \frac{1}{\left((\frac{7}{Re})^{0.9} + 0.27 \frac{\varepsilon}{D} \right)} \right]^{16} \quad ; \ B = \left(\frac{37530}{Re} \right)^{16}$$

$$Re = \frac{ux}{\nu} = \frac{\dot{V}x}{A_{XS}\nu}$$

Where A_{XS} represents cross sectional area. We can see that this is strongly non linear with respect to volumetric flowrate. abcde

Part II

Newton Raphson method for Nonlinear Equations

3 one equation example of Newton Raphson

We want to solve if we have some equation y=f(x)

$$f(x) = 0$$

We are going to make use of derivatives to help us get there:

We do a first order Taylor series approximation of derivatives

$$\frac{df(x)}{dx} = \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

we want to make it such that:

$$f(x + \Delta x) \approx 0$$

So we have this:

$$f(x) >> f(x + \Delta x)$$

$$\frac{df(x)}{dx} = \frac{-f(x)}{\Delta x}$$

$$\frac{df(x)}{dx} = \frac{-f(x)}{\Delta x}$$

Solve for Δx:

$$\Delta x = \frac{-f(x)}{\frac{df(x)}{dx}}$$

Once we solve for Δx, we can update:

$$f(x_{i+1}) = f(x + \Delta x)$$

and repeat the method after we solve for $f(x_{i+1})$

4 Newton Raphson for multiple equations

$$f_1(x_1, x_2, x_3) = 0$$
$$f_2(x_1, x_2, x_3) = 0$$
$$f_3(x_1, x_2, x_3) = 0$$

We need derivatives as well if we want to solve above system of eqns:

$$df_1 = f_1(x_1 + \Delta x_1, x_2 + \Delta x_2, x_3 + \Delta x_3) - f_1(x_1, x_2, x_3)$$

Let's expand the LHS:

$$df_1 = \frac{\partial f_1}{\partial x_1}\Delta x_1 + \frac{\partial f_1}{\partial x_2}\Delta x_2 + \frac{\partial f_1}{\partial x_3}\Delta x_3$$

$$\frac{\partial f_1}{\partial x_1}\Delta x_1+\frac{\partial f_1}{\partial x_2}\Delta x_2+\frac{\partial f_1}{\partial x_3}\Delta x_3 = f_1(x_1+\Delta x_1, x_2+\Delta x_2, x_3+\Delta x_3)-f_1(x_1,x_2,x_3)$$

We need to do this for f_2 and f_3 as well:

$$\frac{\partial f_2}{\partial x_1}\Delta x_1+\frac{\partial f_2}{\partial x_2}\Delta x_2+\frac{\partial f_2}{\partial x_3}\Delta x_3 = f_2(x_1+\Delta x_1, x_2+\Delta x_2, x_3+\Delta x_3)-f_2(x_1,x_2,x_3)$$

$$\frac{\partial f_3}{\partial x_1}\Delta x_1+\frac{\partial f_3}{\partial x_2}\Delta x_2+\frac{\partial f_3}{\partial x_3}\Delta x_3 = f_3(x_1+\Delta x_1, x_2+\Delta x_2, x_3+\Delta x_3)-f_3(x_1,x_2,x_3)$$

Let's write this in matrix form

$$\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta x_3 \end{bmatrix} = \begin{bmatrix} f_1(x_1+\Delta x_1, x_2+\Delta x_2, x_3+\Delta x_3) \\ f_2(x_1+\Delta x_1, x_2+\Delta x_2, x_3+\Delta x_3) \\ f_3(x_1+\Delta x_1, x_2+\Delta x_2, x_3+\Delta x_3) \end{bmatrix} - \begin{bmatrix} f_1(x_1,x_2,x_3) \\ f_2(x_1,x_2,x_3) \\ f_3(x_1,x_2,x_3) \end{bmatrix}$$

Again, we need to approx:

$$\begin{bmatrix} f_1(x_1+\Delta x_1, x_2+\Delta x_2, x_3+\Delta x_3) \\ f_2(x_1+\Delta x_1, x_2+\Delta x_2, x_3+\Delta x_3) \\ f_3(x_1+\Delta x_1, x_2+\Delta x_2, x_3+\Delta x_3) \end{bmatrix} \approx \vec{0}$$

And we just solve:

$$\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta x_3 \end{bmatrix} = - \begin{bmatrix} f_1(x_1,x_2,x_3) \\ f_2(x_1,x_2,x_3) \\ f_3(x_1,x_2,x_3) \end{bmatrix}$$

We shall say that the Jacobian matrix is:

$$\vec{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} \end{bmatrix}$$

We need to find the inverse in order to solve the above equation

$$\vec{J}^{-1}$$

So that we can find:

$$\begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta x_3 \end{bmatrix} = -\vec{J}^{-1} \begin{bmatrix} f_1(x_1, x_2, x_3) \\ f_2(x_1, x_2, x_3) \\ f_3(x_1, x_2, x_3) \end{bmatrix}$$

Once we find $\vec{\Delta x}$, we can update the values:

$$\vec{\Delta x} = \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta x_3 \end{bmatrix}$$

This is the part where we find the next iteration

$$\overrightarrow{x_{i+1}} = \overrightarrow{x_i} + \overrightarrow{\Delta x}$$

$$\overrightarrow{x_i} = \begin{bmatrix} x_{1(i)} \\ x_{2(i)} \\ x_{3(i)} \end{bmatrix}$$

$$\overrightarrow{x_{i+1}} = \begin{bmatrix} x_{1(i+1)} \\ x_{2(i+1)} \\ x_{3(i+1)} \end{bmatrix}$$

Keep substituting until the solution is found within tolerance.

Part III

How are Matrices Constructed

https://spicesharp.github.io/SpiceSharp/articles/structure/flow.html

The Y-matrix (Jacobian) and the RHS matrix: $-F(x^{(k)}) + J(x^{(k)}) \bullet x^{(k)}$

https://spicesharp.github.io/SpiceSharp/articles/custom_components/modified_nodal_analysis.html

This is done in the IBiasingBehaviour: "contains methods that are called each iteration in order to build up the Y-matrix and right-hand side vector."

https://spicesharp.github.io/SpiceSharp/articles/structure/flow.html

IBehaviour is found here:

https://github.com/SpiceSharp/SpiceSharp/tree/master/SpiceSharp/Simulations/Behaviors

IBiasingBehaviour is found here:

https://github.com/SpiceSharp/SpiceSharp/tree/dace7bddaadf7ac22bc32abd2dc61b4d28afb13d/SpiceSharp/Simulations/Base/Biasing

4.1 Example of How matrices are constructed for ISRC (current Source)

We can see how matrices are constructed in current sources, under the biasing class:

https://github.com/SpiceSharp/SpiceSharp/blob/ea977448c74aaa1cd515678585529e707b29f655/SpiceSharp/Components/Currentsources/ISRC/Biasing.cs

Dependency injection for Biasing is done using IComponentBiasingContext. Context classes contain objects used for Dependency injection.

https://github.com/SpiceSharp/SpiceSharp/blob/4d118c3458a1f4921ebab4c8ed9e9ec3f08dd5f9/SpiceSharp/Components/IComponentBindingContext.cs#L11

And one class that implements this is componentBindingContext:

https://github.com/SpiceSharp/SpiceSharp/blob/4d118c3458a1f4921ebab4c8ed9e9ec3f08dd5f9/SpiceSharp/Components/ComponentBindingContext.cs

It inherits from bindingContext.

https://github.com/SpiceSharp/SpiceSharp/blob/4d118c3458a1f4921ebab4c8ed9e9ec3f08dd5f9/SpiceSharp/Entities/BindingContext.cs#L19

4.1.1 matrices and RHS vectors are contained in elementSet type objects under the Biasing class for each component

For matrix construction, we see that in the constructor,

```
 _elements = new ElementSet<double>(_biasing.Solver, null,
  _variables.GetRhsIndices(_biasing.Map));
```

This is what constructs the system of equations.

In the load method, we actually manipulate this elementSet object to add a value.

```
_elements.Add(-value, value);
```

So elementSet is important. It contains the solver object, which in turn contains the YMatrix and RHS vector directly.

https://github.com/SpiceSharp/SpiceSharp/blob/ea977448c74aaa1cd515678585529e707b29f655/SpiceSharp/Algebra/ElementSet.cs

Elementsets also contain arrays of matrix locations (ie rows and columns) and rhsPins.

These are arrays of locations to access variables within the Ymatrix and rhs vector respectively.

4.1.1.1 How constructors for the ElementSet works

The elementSet constructor takes in a few arguments. The solver object, the matrixPins and the rhsPins.

As said before, the matrix pins help to index certain variables and elements within the Ymatrix or jacobian. And the rhs pins help to index the rhs vector.

the solver object contains the Ymatrix, rhsVector and methods to solve the system of equations.

Other than that, the elementSet doesn't seem to invoke anything else. It's just a class that contains the data needed.

It seems to build its own internal array of elements here.

So it extracts specific values of elements from the solver matrices and rhs vectors based on the pins (indices) you give.

This will give it the initial set of elements to work with.

4.1.1.2 How the addition function works

To manipulate the matrices, for example for the current source,

we extract the elements in the matrix pertaining to this specific current source using the rhsPins.

then we use the ElementSet.Add() method to add and subtract values to it.

Int the currentSource example, two values of current are added and subtracted from the RHS matrix. Possibly to the node before and after the current source.

The node before and after the current source is accessed using the getRHSIndices using in the OnePort method. Which then takes in the biasing variable and context variable.

4.1.2 How are variables mapped to circuit components?

Our clue in

https://github.com/SpiceSharp/SpiceSharp/blob/ea977448c74aaa1cd515678585529e707b29f655/SpiceSharp/Components/Currentsources/ISRC/Biasing.cs

Is that the variables object has a getRhsIndices method to return the correct rhsVector Index.

The input to this method is a biasing map object, which implemnts the IBiasingSimulationState interface.

So the oneport object has a positive and negative connection. Much like an inlet and outlet.

4.1.2.1 onePort Source code

https://github.com/SpiceSharp/SpiceSharp/blob/ea977448c74aaa1cd515678585529e707b29f655/SpiceSharp/Components/Common/OnePort.cs

The OnePort class is located under common behaviours.

Note again that behaviours deal with how matrices are created and solved.

As we can see, oneport refers to a behaviour of a device with two pins with current in one pin being equal to current going out of one pin.

The positive and negative nodes are represented by objects implementing IVariable Interfaces.

Matrix locations and rhsVector locations are both mapped using IVariableMap type objects.

This is line 59-83 of the OnePort code.

One thing though is that for the Ymatrix, there are four elements.

How does that work? we can take a look at the example below to find out

4.1.2.1.1 Modified Nodal Analysis Example https://spicesharp.github.io/SpiceSharp/articles/custom_components/example_resistor.html

We can see from here a single resistor has two contributinos to the RHS vector and four to the Ymatrix.

The individual equations are constructing taking current balances over nodes.

On the inlet node, the current balance equation is f_A, which means the sum of currents coming out of node A is zero.

$$f_A = i_{out} - i_{in} = 0$$

For f_A, we only have the current flowing into resistor being the current output for this node,

$$f_A = \frac{v_A - v_B}{R} - i_{in}$$

Likewise for node B, we have current flowing in and out of node B

$$f_B = i_{out} - \frac{v_A - v_B}{R}$$

Let a resistor with an inlet voltage v_A and outlet voltage v_B have a resistance r and current i_R

Now how does this contribute to our matrix and RHS vector?

For starters, take a look at the example for this simple case, a 3x3 vector, which represents our newton raphson method.

$$\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta x_3 \end{bmatrix} = - \begin{bmatrix} f_1(x_1, x_2, x_3) \\ f_2(x_1, x_2, x_3) \\ f_3(x_1, x_2, x_3) \end{bmatrix}$$

We rewrite this as:

$$\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} \end{bmatrix} \begin{bmatrix} x_1^{(k+1)} - x_1^{(k)} \\ x_2^{(k+1)} - x_2^{(k)} \\ x_3^{(k+1)} - x_3^{(k)} \end{bmatrix} = - \begin{bmatrix} f_1(x_1, x_2, x_3) \\ f_2(x_1, x_2, x_3) \\ f_3(x_1, x_2, x_3) \end{bmatrix}$$

$$\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} \end{bmatrix} \begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ x_3^{(k+1)} \end{bmatrix} = - \begin{bmatrix} f_1(x_1, x_2, x_3) \\ f_2(x_1, x_2, x_3) \\ f_3(x_1, x_2, x_3) \end{bmatrix} + \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} \end{bmatrix} \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \\ x_3^{(k)} \end{bmatrix}$$

In spicesharp, the RHS vector isn't simply a result of the current balance equations. But rather that plus contributions from the jacobian.

For our case, we will need to compute the Jacobian first in order to find out contributions to both RHS vector and Y matrix.

For contributions to the jacobian, four partial derivatives are taken.

$$\frac{\partial f_A}{\partial v_A}, \frac{\partial f_A}{\partial v_B}, \frac{\partial f_B}{\partial v_A}, \frac{\partial f_B}{\partial v_B},$$

This is why there are four contributions to the matrix.

Take for example we have a vector

$$\begin{bmatrix} v_A \\ \ldots \\ v_B \\ \ldots \end{bmatrix}$$

Let's say v_A is at index 1.

From this we know A correponds to index one.

And to locate $\frac{\partial f_A}{\partial v_A}$ within the jacobian. the numerator ∂f_A corresponds to the row number. The denominator ∂v_B corresponds to the column number.

So $\frac{\partial f_A}{\partial v_A}$ corresponds to row 1 col 1. If v_B is at row 7, then B corresponds to index no. 7.

$\frac{\partial f_B}{\partial v_A}$ then belongs to row 7 (see B in numerator) and col 1 (see A in denominator).

Under the GetMatrixMethod in oneport,

https://github.com/SpiceSharp/SpiceSharp/blob/ea977448c74aaa1cd515678585529e707b29f655/SpiceSharp/Components/Common/OnePort.cs

We see that the (pos,pos) which is the positive row index and positive column index, this will correspond to $\frac{\partial f_B}{\partial v_B}$, while the negative row index and negative column index (neg,neg) will correspond to $\frac{\partial f_A}{\partial v_A}$.

whereas, $\frac{\partial f_A}{\partial v_B}$, we note that equation indices correspond to the row index of the jacobian, while the variable with which we perform partial derivatives corrsponds to the column. So this will then correspond to (neg,pos) coordinates within the MatrixLocation method.

Likewise $\frac{\partial f_B}{\partial v_A}$ will correspond to the (pos,neg) location. (I think) since node B is the positive end of the resistor and node A is the negative end.

In summary:

(pos, pos) will correspond to

$$\frac{\partial f_{pos}}{\partial v_{pos}}$$

(pos, neg) will correspond to:

$$\frac{\partial f_{pos}}{\partial v_{neg}}$$

(neg, pos) will correspond to:

$$\frac{\partial f_{neg}}{\partial v_{pos}}$$

For resistors, the biasing code is here:

https://github.com/SpiceSharp/SpiceSharp/blob/ea977448c74aaa1cd515678585529e707b29f655/SpiceSharp/Components/RLC/Resistors/Biasing.cs

We can see a Conductance value given to (pos,pos), (neg,neg) index. And a negative conductance value given to the (neg,pos) and (pos,neg) index. Can't really tell which is which here.

For diodes, i also cannot tell which is which

https://spicesharp.github.io/SpiceSharp/articles/custom_components/example_diode.html

4.1.2.1.2 which terminal is positive and negative?

https://spicesharp.github.io/SpiceSharp/articles/tutorials/gettingstarted.html

We can see that from the constructor for resistor, the second argument is the name of the +ve terminal and the first argument is the name of the -ve terminal.

https://github.com/SpiceSharp/SpiceSharp/blob/master/SpiceSharp/Components/RLC/Resistors/Resistor.cs

```
public Resistor(string name, string pos, string neg, double res): this(name)
```

This code proves the above statement.

Then we can look at the diagram shown here:

https://spicesharp.github.io/SpiceSharp/articles/tutorials/gettingstarted.html

We see for resistors and voltage sources, the second argument is the positive node, and the third argument is the negative node.

For resistors, current flows from positive to negative within the resistor.

For voltage sources, current flows from positive to negative AROUND the circuit rather than within the voltage source.

Conventions are reversed for resistor and voltage sources.

https://github.com/SpiceSharp/SpiceSharp/blob/master/SpiceSharp/Components/Voltagesources/VSRC/VoltageSource.cs

4.1.2.1.3 How nodes are indexed (continued...)

in the Biasing.cs code under resistor

https://github.com/SpiceSharp/SpiceSharp/blob/master/SpiceSharp/Components/RLC/Resistors/Biasing.cs

The oneport type object has a method called getmatrixlocations. Getmatrixlocations takes in an IVariableMap type object and returns matrix indices.

(if we go down the rabit hole of IBiasingSimulationState which contains the VariableMap, we likely will get lost in code)

So let's just go straight to see IVariableMap.

What's this IVariableMap?

https://github.com/SpiceSharp/SpiceSharp/tree/ea977448c74aaa1cd515678585529e707b29f655/SpiceSharp/Simulations/Variables

VariableMap is here:

https://github.com/SpiceSharp/SpiceSharp/blob/ea977448c74aaa1cd515678585529e707b29f655/SpiceSharp/Simulations/Variables/VariableMap.cs

VariableMap contains a dictionary storing pairs of IVariable vs their index (int).

The constructor of the variableMap will add a "ground" variable to the unknown vector with index 0.

```
public VariableMap(IVariable ground)
        {
            _map.Add(ground, 0);
        }
```

VariableMaps store the name of the node with an index number (integer).

4.1.2.2 Question is: how are Variables Indexed in variableMaps?

The circuit class doesn't store the data of variableMaps, rather it is essentially a list of entities (or components if that makes more sense)

https://github.com/SpiceSharp/SpiceSharp/blob/4d118c3458a1f4921ebab4c8ed9e9ec3f08dd5f9/SpiceSharp/Entities/Circuit.cs

So the code that deals with it is the resistor Class. Which uses the Connect(pos,neg) method.

The connect method is inherited from Component line 42-60 as of may 2022.

https://github.com/SpiceSharp/SpiceSharp/blob/4d118c3458a1f4921ebab4c8ed9e9ec3f08dd5f9/SpiceSharp/Components/Component.cs

The connect method just populates the connections private variable (line 21 in Component.cs as of May 2022).

If you want to access the connections private variable, you call the Nodes method within the entity (component). It will return a readonly list of strings (names of the nodes).

Next place to look for where variableMaps is constructed is DC simulation (i'm looking at the getting started example. https://spicesharp.github.io/SpiceSharp/articles/tutorials/gettingstarted.html

The DC simulation code is found here:

https://github.com/SpiceSharp/SpiceSharp/tree/4d118c3458a1f4921ebab4c8ed9e9ec3f08dd5f9/SpiceSharp/Simulations/Implementations/DC

So we looked at the constructor of the DC simulation, no information of nodes is supplied here.

The only place we supply information to the DC simulation object is in the line:

```
dc.Run(ckt)
```

So where is the Run method?

we want to look at the parent(base) classes where DC inherits from.

DC inherits from Biasing Simulation

Biasing Simulation is split into three files (partial classes), and is not meant to be instantiated on its own (abstract class).

BiasingSimulation Partial Classes

- https://github.com/SpiceSharp/SpiceSharp/blob/4d118c3458a1f4921ebab4c8ed9e9ec3f08dd5
  SpiceSharp/Simulations/Base/Biasing/BiasingSimulation.IterationState.

cs#L3

- https://github.com/SpiceSharp/SpiceSharp/blob/4d118c3458a1f4921ebab4c8ed9e9ec3f08dd5 SpiceSharp/Simulations/Base/Biasing/BiasingSimulation.SimulationState. cs#L8
- https://github.com/SpiceSharp/SpiceSharp/blob/4d118c3458a1f4921ebab4c8ed9e9ec3f08dd5 SpiceSharp/Simulations/Base/Biasing/BiasingSimulation.cs#L23

Under simulation state, we see VariableMap appear for the first time in our code exploration. Under the SimulationState nested class.

The Map method accesses the variable map using a lambda function.

In simulationState constructor, a new solver variable called gnd (ground) is constructed and loaded into a newly instantiated variable map.

```
var gnd = new SolverVariable<double>(this, Constants.Ground, 0, Units.Volt);
_map = new VariableMap(gnd);
Add(Constants.Ground, gnd);
```

So besides adding the ground variable to the new variable map, we also use the CreatePrivateVariable method within the simulationstate to construct the variable map.

```
public IVariable<double> CreatePrivateVariable(string name, IUnit unit)
{
var index = _map.Count;
var result = new SolverVariable<double>(this, name, index, unit);
_map.Add(result, index);
return result;
}
```

The first variable to be put in here is the ground variable. Followed by whatever variables we make using createPrivateVariable.

And everytime we make a shared variable, we also call the createPrivateVariable method. (look at later)

SimulationState plays an important role in matrix construction by constructing the variableMap.

https://github.com/SpiceSharp/SpiceSharp/blob/4d118c3458a1f4921ebab4c8ed9e9ec3f08dd5f9/SpiceSharp/Simulations/Base/Biasing/BiasingSimulation.SimulationState.cs#L45

Note: the Run method is within the Simulation Class https://github.com/SpiceSharp/SpiceSharp/blob/4d118c3458a1f4921ebab4c8ed9e9ec3f08dd5f9/SpiceSharp/Simulations/Simulation.cs#L18

The run method takes in an entityCollection, which is a circuit object essentially.

it first sets up the entities like so:

```
try
{
Status = SimulationStatus.Setup;
Setup(entities);
}
```

Then it validates the entities

```
try
{
Status = SimulationStatus.Validation;
Validate(entities);
}
```

once the simulation is setup and we start to run

```
do
{
// Before execution
OnBeforeExecute(beforeArgs);

// Execute simulation
Statistics.ExecutionTime.Start();
try
{
Execute();
}
finally
{
Statistics.ExecutionTime.Stop();
}

// Reset
afterArgs.Repeat = false;
OnAfterExecute(afterArgs);

// We're going to repeat the simulation, change the event arguments
if (afterArgs.Repeat)
beforeArgs = new BeforeExecuteEventArgs(true);
} while (afterArgs.Repeat);
```

the Execute() method is used. So three important functions here here:

(1) firstly, the setup method, then (2) secondly the Validate method (3) thirdly the Execute() method

Here, the Setup method has both the CreateBehaviours method and CreateStates Method. https://github.com/SpiceSharp/SpiceSharp/blob/4d118c3458a1f4921ebab4c8ed9e9e SpiceSharp/Simulations/Simulation.cs#L240

CreateStates is found under BiasingSimulation, a child class

https://github.com/SpiceSharp/SpiceSharp/blob/4d118c3458a1f4921ebab4c8ed9e9ec3f08dd5f9/ SpiceSharp/Simulations/Base/Biasing/BiasingSimulation.cs#L141

this feels like another setup class.

4.1.2.2.1 CreateBehaviors But most of the matrix creation would be found under CreateBehaviours

https://github.com/SpiceSharp/SpiceSharp/blob/4d118c3458a1f4921ebab4c8ed9e9ec3f08dd5f9/ SpiceSharp/Simulations/Simulation.cs#L267

this CreateBehaviours method simply cycles through all the CreateBehaviours methods stored within each entity

https://github.com/SpiceSharp/SpiceSharp/blob/4d118c3458a1f4921ebab4c8ed9e9ec3f08dd5f9/ SpiceSharp/Simulations/Simulation.cs#L288

CreateBehaviours is found as an abstract method under entities (essentially like an interface)

https://github.com/SpiceSharp/SpiceSharp/blob/4d118c3458a1f4921ebab4c8ed9e9ec3f08dd5f9/SpiceSharp/Entities/Entity.cs#L39

This is exactly what is shown in the documentation here:

https://spicesharp.github.io/SpiceSharp/articles/structure/flow.html

Now, not all entities have special behaviours to create. For example, a linear (ohm's law obeying) resistor doesn't have a CreateBehaviours method explicitly defined. It doesn't do anything.

However, the createBehaviours doesn't end there. (It does for simulation). But for a DC simulation, which inherits from BiasingSimulation, createbehaviours is also defined there.

https://github.com/SpiceSharp/SpiceSharp/blob/4d118c3458a1f4921ebab4c8ed9e9ec3f08dd5f9/SpiceSharp/Simulations/Base/Biasing/BiasingSimulation.cs#L153

We see that the EntityBehavior objects are used extensively here to load behaviours (change the system of eqns to solve).

```
protected override void CreateBehaviors(IEntityCollection entities)
{
base.CreateBehaviors(entities);
_temperatureBehaviors = EntityBehaviors.GetBehaviorList<ITemperatureBehavior>();
_loadBehaviors = EntityBehaviors.GetBehaviorList<IBiasingBehavior>();
_convergenceBehaviors = EntityBehaviors.GetBehaviorList<IConvergenceBehavior>();
_updateBehaviors = EntityBehaviors.GetBehaviorList<IBiasingUpdateBehavior>();
```

EntityBehaviors is the class which is instantiated in CreateBehaviors.

```
EntityBehaviors = new BehaviorContainerCollection(entities.Comparer);
```

this is of the type BehaviorContainerCollection

https://github.com/SpiceSharp/SpiceSharp/blob/4d118c3458a1f4921ebab4c8ed9e9ec3f08dd5f9/SpiceSharp/Simulations/Behaviors/BehaviorContainerCollection.cs#L12

The BehaviorContainerCollection class takes in a IEqualityComparer Class from the entities.Comparer attribute or object it uses it to initiate a new dictionary of String and IBehaviourContainer https://github.com/SpiceSharp/SpiceSharp/blob/4d118c3458a1f4921ebab4c8ed9e9ec3f08dd5f9/SpiceSharp/Simulations/Behaviors/BehaviorContainerCollection.cs#L12

```
public virtual BehaviorList<T> GetBehaviorList<T>() where T : IBehavior
{
_lock.EnterReadLock();
try
{
var list = new List<T>(_values.Count);
foreach (var elt in _values)
{
if (elt.TryGetValue(out T value))
list.Add(value);
}
return new BehaviorList<T>(list.ToArray());
}
finally
{
_lock.ExitReadLock();
}
}
```

Basically what it looks like the code is doing is that we are looking at the _values attribute. The _values attribute and trying to extract behaviours from this list according to the type supplied.

The next question is where is this list generated in the first place?

One clue is that the Add method in line 90 https://github.com/SpiceSharp/SpiceSharp/blob/4d118c3458a1f4921ebab4c8ed9e9ec3f08dd5f9/SpiceSharp/Simulations/Behaviors/BehaviorContainerCollection.cs#L90

This method actually adds to both the dictionary and the _values variable inside the BehaviorContainerCollection object.

The _value variable is declared in the constructor, but the add method will populate this with values.

The Add method also populates an internal dictionary with value key pairs so that they can be referenced.

Only Problem is, i don't see clearly where the add method is invoked.

One other curious thing, EntityBehaviours has a BehaviorsNotFound Atrribute or property.

BehaviorsNotFound is an event handler.

4.1.2.2.2 How to make void functions "return" values And it is here i found the out keyword, this is what actually makes it possible for a void method to modify a value outside itself.

https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/out-parameter-modifier

4.1.2.2.3 More on Behaviors and Events in C# So to understand this, i will likely have to understand events because they are very prevalent in the Simulation.cs file.

So anyhow since i don't see any add method for BehaviorContainerCollection being used. I assume the construction is called in the constructor. By this i mean the _values is made in the constructor or when the TryGetValue method is invoked,

The Class accesses the dictionary and adds it to the values.

But i cannot really see how the logic flows. Looks like everything is really heavily dependent on events and their flow in order to see what really happens and how the _values list is generated for us to see how GetBehaviorList actually works.

One way to move forward is to use tests, modify the DC simulation code, introduce print statements to see where the values are. And then try to debug like so.

Probably better to pry things apart, see how they work step by step rather than try to figure it all out in theory and reading source code.

4.1.2.2.4 So i attempted to try prying apart the simulation code..

So i tried out the idea of attempting to make my own version of the Simulation class. As well as the biasing Simulation class, so as to put in print statements.

However, the moment i started creating my own classes in a new namespace, there were a lot of contexts missing, as well as the Properties object being missing.

Looking into the code for hours, I find that it's not worth it for now. Just make a pipe and pump as if it were a nonlinear resistor.

Part IV

Matrix Solvers

Note we are solving for (in a 3x3 matrix case):

$$\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} \end{bmatrix} \begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ x_3^{(k+1)} \end{bmatrix} = - \begin{bmatrix} f_1(x_1, x_2, x_3) \\ f_2(x_1, x_2, x_3) \\ f_3(x_1, x_2, x_3) \end{bmatrix} + \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} \end{bmatrix} \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \\ x_3^{(k)} \end{bmatrix}$$

Note, that the Ymatrix or Jacobian will be evaulated at $\overrightarrow{x^{(k)}}$. We substitute all the partial derivatives in the Jacobian with the x values at the current iteration. And these all become constants for that iteration.

These are found under: https://github.com/SpiceSharp/SpiceSharp/tree/master/SpiceSharp/Algebra

So once we have our Jacobians, the matrix solver takes care of the rest. So the inversion of the Jacobian (Y matrix), and matrix multiplication are taken care of by these solvers.

Now let's say we want to solve a matrix, where we have a Y matrix and a RHS vector.

How do we put them into a class?

nstructor

5 Where the Y matrix is defined

Each solver class, eg. SparseRealSolver

https://github.com/SpiceSharp/SpiceSharp/blob/dace7bddaadf7ac22bc32abd2dc61b4d28afb13d/SpiceSharp/Algebra/Solve/LU/SparseSolver/SparseRealSolver.cs

https://spicesharp.github.io/SpiceSharp/api/SpiceSharp.Algebra.SparseRealSolver.html

will have a way of setting the matrix.

For example,

under the sparseSolverTests

https://github.com/SpiceSharp/SpiceSharp/blob/dace7bddaadf7ac22bc32abd2dc61b4d28afb13d/SpiceSharpTest/Algebra/SparseSolverTests.cs

from line 43-72,

5.1 setting matrices and RHS vector values within the solver

we can clearly see a matrix and rhs being defined for us.

the matrix there is just a array of array of doubles. Whereas the rhs vector is simply an array of doubles.

The Solver.GetElement method will modify the matrix within the solver object if a row and column is supplied.

And the Solver.GetElement method will modify the RHS vector within the solver object if a row value is supplied.

GetElement methods are defined in SparseLUSolver.cs, and two overloads are given. One gets the element of the matrix within the solver, one gets the vector.

Both of these return Element<T>, and this class represents the properties of elements within matrices and vectors.

5.2 getting no. of equations within the solver

The solver.size represents the number of equations represented by the matrix

https://spicesharp.github.io/SpiceSharp/api/SpiceSharp.Algebra.SparseRealSolver.html

The sparseRealSolver inherits from sparseLUSolver which in turn inherits from pivotingSolver.

https://github.com/SpiceSharp/SpiceSharp/blob/dace7bddaadf7ac22bc32abd2dc61b4d28afb13d/SpiceSharp/Algebra/Solve/PivotingSolver.cs

the pivoting solver in turn defines the size property. Which returns the maximum value of the size of the matrix and size of the vector (which is the RHS vector).

The size attribute is defined in the constructor

https://github.com/SpiceSharp/SpiceSharp/blob/dace7bddaadf7ac22bc32abd2dc61b4d28afb13d/SpiceSharp/Algebra/Matrix/SparseMatrix/SparseMatrix.cs

At least for the sparseMatrix.cs, we can see that these matrices are necessarily square matrices from line 75-94 of SpiceSharp/Algebra/Matrix/SparseMatrix/SparseMatrix.cs

The same size attribute is used to defined **both** rows and columns of the matrices. This defines the matrix size.

A similar process can be said for the vector:

/SpiceSharp/Algebra/Vector/SparseVector/SparseVector.cs

in lines 54-59.

We can see that getting the maximum value of the size and vectors will get us the number of equations.

What if they weren't the same?

It should be apparent that an error should occur.

But that is kind of an exception than the norm. Shouldn't happen if we define our matrices and vectors properly.

5.3 Sparse vs Dense Vectors or Matrices

Sparse vectors or Matrices have most of their elements set to zero.

Why do we have different classes for sparse vectors and dense vectors? Likely their solving strategies may be different.

5.4 Solving the system of equations with Ymatrix and RHS vector

see line 174-234 of

https://github.com/SpiceSharp/SpiceSharp/blob/dace7bddaadf7ac22bc32abd2dc61b4d28afb13d/SpiceSharpTest/Algebra/SparseSolverTests.cs

- (1) define RHS and Y vector using for loops
- (2) OrderAndFactor
- (3) create a new solution vector
- (4) feed solution vector into solver.Solve() method
- (5) check the solution vector

Part V

Simulation Algorithm

This is taken care of by ISimulation

https://spicesharp.github.io/SpiceSharp/articles/structure/flow.html https://github.com/SpiceSharp/SpiceSharp/tree/master/SpiceSharp/Simulations

Part VI

Nonlinear Resistors

Now I tried prying apart the algorithm to no avail.

It seems like the most efficient course of action is to just use the nonlinear resistor code and just modify it to fit a pipe.

We take the conversion here to be I (A)= flowrate in m^3/s and ; Voltage (V) = Pressure in Pa

6 nonlinear resistor source code under spicesharpTests

The source code for nonlinear resistors is in

https://github.com/SpiceSharp/SpiceSharp/tree/master/SpiceSharpTest/Examples/CustomResistor

And we pretty much add a new component in the namespace called a nonlinear resistor.

https://github.com/SpiceSharp/SpiceSharp/blob/master/SpiceSharpTest/Examples/CustomResistor/NonlinearResistor.cs

7 New Biasing Behaviors

Once the component is created, it will look for biasing behaviours by deafult.

This is done using the automatic Dependency injection mechanism.

https://spicesharp.github.io/SpiceSharp/articles/structure/di.html

For the nonlinear resistor, the biasingBehavior should be in the namespace:

```
SpiceSharp.Components.NonlinearResistorBehaviors
```

So in essence, use the name of the component, and add the word behaviors and that will be the name of the namespace.

https://github.com/SpiceSharp/SpiceSharp/blob/master/SpiceSharpTest/Examples/CustomResistor/BiasingBehavior.cs

And for this, we just need to alter the equations in the matrices and perform partial derivates accordingly.

Everything else can be copied wholesale.

8 Adding Properties

The last part is the BaseParameters file, where we define some parameters which will define how our pipe behaves. Eg pipe length, elevation, etc.

For the nonlinear resistor, we just have A and B.

Part VII

Matrices

$$\begin{bmatrix} 1 & 2 & 3 \\ a & b & c \end{bmatrix}$$