Randall Woodall

September 21-27, 2020

Research Log

**September 21, 2020:**

I can implement a DC opf with linear programming in scipy for a general-purpose check. This is good, because I can implement the 3-bus system we’ve been using, then check it against the results of the distribution. My steps for tonight are to implement the centralized dcopf for the 3-bus and (if there’s time) then to implement the same system on the distributed style connections and observe the two sets of results.

Note: Need the suseptance (b), which is the imaginary portion of y

**September 22, 2020:**

Have the centralized DCOPF run, this is the set we’re expecting:

[[161.21141252+0.00000000e+00j]

[138.76322916+0.00000000e+00j]

[ 0. +0.00000000e+00j]

[ 0. +9.63042310e-18j]

[ 0. +1.24198515e-03j]

[ 0. -2.53583167e-02j]

[ 8.92724891+0.00000000e+00j]

[ 8.92680266+0.00000000e+00j]

[ 8.9262879 +0.00000000e+00j]

[ 0. +8.92628790e+00j]]Which is:

Pbus1

Pbus2

Pbus3

Theta1

Theta2

Theta3

Lambda1

Lambda2

Lambda3

Lambda4

This used an infinite value for cost of power at bus3 since it has no generator.

Progress!

A complete failure to converge on an initial distributed multi-bus system.

**September 23, 2020:**

So this is where I’m getting stuck. How do you decide on the export from a bus? I’ve given all the control to the lines, but should the busses have some control too? Should they be able to check all lines and keep a tally? That seems obvious, but a tally of what exactly?

We almost need a decision function inside the nodes. One that says: lambda of this line is lower, so I’m willing to buy more power and sell it to the higher lambda. Would that even work? Maybe, it would force a lambda across the system though. We would not have a lambda for each bus.

This is explained by bus a needing to settle the lambda between each line. If bus a needs to settle lines 1 and 3, and bus b is trying to settle lines 1 and 2, then lines 1, 2 and 3 all need to be the same lambda.

My concern, in bold: **Does this new lambda even mean anything. Is it some sort of blended mixture of the others, is it a system average lambda or is it so far away from the others that it isn’t useful at all?**

So every iteration, buses should check lambdas, and should try and sell power, but how much?

**September 24, 2020:**

This will not be any localized lambda. Will it be the system lambda? Like the ED lambda?

**September 29, 2020:**

Export from a bus should contain more than just the other lines? And how exactly does it fit into the generation model?

**September 30, 2020:**

Alright, spinning my wheels. It’s time for a change in direction. Let’s see if we can get the mathematics solid and prepped before trying to program it and make it work. Ignore how we calculate export, and assume we can simply see it at every step. Using Dr. Ranade’s base formulas.

Let’s go back to how Dr. Ranade explained the problem. Node’s want to minimize cost, while utility wants to maximize profit. Let’s write down the equations and see if we can come up with a distributed method to solve multi-bus.

Using PTDF will make this easier.

Minimize:

∑f(Pg)

Subject to:

Angle1 = 0

Pexport\_bus1 = Pg\_bus1 - Pload\_bus1

Pexport\_bus2 = Pg\_bus2 - Pload\_bus2

Pexport\_bus3 = Pg\_bus3 - Pload\_bus3

Where exported power is expressed as 100 \* Bj \* anglej where j signifies connected busses. This means we come down to only needing the angle and the suseptance to solve this problem!

Based on value of power to each line decide: how much power do I produce *for export*. Bus 1 is required to keep angle at 0, do not update angle on each iteration.

Buses and lines need to share the calculation, neither in control.

Lambda1 would normally be related to bus 1, but in our case will it be line related? They should match? Essentially, yes, for now.

**October 2, 2020:**

Essentially, we’re solving a power flow, and also minimizing costs. In fact, this is literally what we’re doing. We can do this through a lagrangian relaxation by giving each constraint a lambda. The bus angle lambda is not important to us, and the only catch to the others is that they (in this version) are balanced by the line, not by the buses. In this scenario, they must be equal, therefore this shouldn’t be a problem. What about when they are unequal and bus specific? Maybe I should take the control from the lines and give it to the buses? This makes the most sense, and would allow each to have an individual lambda. This requires a split from Dr. Ranade’s version with two buses. We shouldn’t need to actually check export and absorbed. How then, do we take care of the minimization problem? This is no longer a local minimization, where each node would only produce it’s load, but a global. This linkage must be where the line knowing lambda comes in. We essentially need to know if it’s cheaper to import power than to produce it at any time.

**The global minimum should result from the nodes minimizing their costs from both production and import.** 🡨 although unverified, this makes intuitive sense, and can be made more apparent after implementation.

So much needs to change with the file that I’ve already got that I’m going to simply start from scratch.

**October 5, 2020:**

So the conceptual side of the program came a long way yesterday. Where I’m stuck conceptually is in the angle calculations. My current idea: at each node, simply keep tabs on all the angle difference across lines. From this, once a solution settles, we can re-settle the angles to match the requirements. Essentially, each node will have a linear set of simple equations that say what its actual angle should be instead of just the relative angle.

To start: Let’s detail our process and then actually verify that it gets to the solution with a simple 2-bus.

Process:

1. Given current angles, how much does each node export?
2. Calculate the value of the export with our lambda
   1. Check import value with the other node’s lambda
3. Adjust each node’s lambda up if the price to produce > the value to import
4. Given new prices, estimate desired generation
5. Calculate the relative angles across each line

Now, let’s use this process on our buses and see if there are any major problems.

General info:

Bus A: Pload 0, a = 1, b = .0625

Bus B: Pload 100, a = 1, b = .0125

B-Matrix: [[-1, 1], [1, -1]

Iteration 1:

Bus A angle = 0, lambda = 0, Pgen = 0

Bus B angle = 0, lambda = 0, Pgen = 0

Line angle = 0

Bus A export: 100 \* (-1) \* 0 + 100 \* 1 \* 0 - 0 + 0 = 0

Bus B export: 100 \* 1 \* 0 + 100 \* (-1) \* 0 - 0 + 100 = 100

Bus A price to produce |0| for export: a + 2 \* b \* 0 = 1

Bus A price to buy |0| for loads/exports: lambdaB \* 0 = 0

Bus B price to produce |100| for export: a + 2 \* b \* |100| = 3.5

Bus B price to buy |100| for loads/exports: **lambdA \* |100| = 0**

Bus A lambda goes up: lambda = 1

Bus B lambda goes up: lambdaB = 3.5

PgenA = a / (lambdaA – 2\*b) = 1.142857143

PgenB = a / (lambdaB – 2\*b) = .2877697842

Line says: Node A generates 1.142857143 MW for export on line

Line says: Node B generates 0 for export on line]4]21’

Assign angle across line to node B, no angle to node A

Iteration 2:

Bus A angle = 0, lambda = 1, Pgen = 1.142857143

Bus B angle = -.0011428571, lambda = 3.5, Pgen = .2877697842

Bus A export: 100 \* (-1) \* 0 + 100 \* 1 \* (-.0011428571) - 1.142857143 + 0 = -1.257142853

Bus B export: 100 \* 1 \* 0 + 100 \* (-1) \* (-.0011428571) - .2877697842 + 100 = 99.59794451

Bus A price to produce |-1.257142853| for export: a + 2 \* b \* |-1.257142853| = .8428571434

Bus A price to buy |-1.257142853| for loads/exports: lambdaB \* |-1.257142853| = 4.4

Bus B price to produce |99.59794451| for export: a + 2 \* b \* |99.59794451| = 3.49

Bus B price to buy |99.59794451| for loads/exports: **lambdA \*** |99.59794451| **= 99.59794451**

Bus A lambda goes down: lambdaA = .8428571434

Bus B lambda goes down? lambdaB = 1

… Falls apart here. Lambda seems to be going in the wrong direction. Let’s write a simple script to fast-forward this process. We’ll put it together simply with clear comments so it can be modified to fix issues as we discover them.

**October 8, 2020:**

Implementing the previous pattern, saving to csv for easy iteration comparison. Using the specific 3 bus case instead of making a general program.

Base structure is put together, time to start fleshing it out.

Export values calculated.

Step two is where things get a little bit complicated. Perhaps we should keep the total exports in a matrix form, so we can measure transmission from 1->2 etc. Exports are separate, but what do these export numbers really mean? Clearly, the non-diagonal components are the amount shipped to the other bus. 100 \* B[1][2] \* angle2 is the power shipped 1->2, while 100\*B[2][1] \* angle1 is the power shipped 2->1 (should be equal, opposite signs). What does 100\*B[1][1]\*angle1 mean? With angle1 being reference, this would be 0, but in any other case:

In essence, it’s angle1 \* 100 \* -(B[1][2] + B[1][3])

I get it, this offsets the angles for when your angle is non-zero. Otherwise we’d get the value of reference->bus. The signs are exactly opposite, therefore, it cancels out the excess power due to non-zero self-angle. Essentially a correction factor.

This draws my method into concern, however. We can’t simply use the elements as exported power, we need the angle difference, every time, that we need node to node transfer. Specifically, our equation would look like: 100 \* B[1][2] \* (angle2 – angle1) or 100 \* B[2][1] \* (angle1 – angle2) for the other direction. This is simple when everything is 0, but it won’t always be that way.

**October 9, 2020:**

So what does this mean for the export calculation? It means that at the non-reference bus we need to normalize by our own angle for each export value. This can be done by the angle difference.

Either (since voltage magnitude is all assumed to be 1):

100 \* B[2][1] \* angle2 – 100 \* B[1][2] \* angle1

Or

100 \* B[2][1] \* (angle2 – angle1)

The second version exploits the symmetry of the matrix, while the first one is more mathematically proper. May as well implement the first, in case there’s ever a reason for needing it.

So we have total export, which is nice to calculate for the our cost to generate, and we have a method for calculating bus to bus power transfer.

This is going to be complex to program as general, so we’re just going to do specific for now to speed up the process.

Have bus to bus export, now to measure value.

I’m thinking a matrix is the best way to do this. Element [1][1] would be cost to produce our export, [1][2] will be gain to export 1->2, [2][1] will be cost to 2 of the export/import. Negative for gain for export, positive for cost to import (minimize costs). This will let us simply try to minimize our ‘row’ of the matrix. Is this a linear system? Let’s try to express the cost at each node simply as a single function…

We need to juggle import/export values to do this, can’t express as a single simple function.

Going back to the matrix, the off-diagonal will be value to first then second bus, the diagonal will be total value at that specific bus. The goal will be to minimize this plus generation cost. Our specific row will be affected by the angles. We can use an activation function based on sign to decide on which lambda in each case. Which would allow us to show it as a function of sorts. Multi-piecewise.

Row one would be:

[100 \* B[1][2] \* (angle2-angle1) \* (lambda2\*(angle2<angle1) – lambda1\*(angle2>angle1)) + 100 \* B[1][3] \* (angle3-angle1) \* (lambda3\*(angle3<angle1) – lambda1\*(angle3>angle1)), 100 \* B[1][2] \* (angle2-angle1) \* (lambda2\*(angle2<angle1) – lambda1\*(angle2>angle1)), 100 \* B[1][3] \* (angle3-angle1) \* (lambda3\*(angle3<angle1) – lambda1\*(angle3>angle1))]

That’s a heck of a mouthful.

The first element is the sum of the second two elements. The second two elements are the transfer between the bus 1 and the other two buses respectively.

**October 12, 2020:**

We have a matrix of transfers between buses such that:

[[one->two, two->one],

[one->three, three->one],

[two->three, three->two]]

We need to have each element multiplied by the lambda of the sending bus, half the elements will be positive, half will be negative. The positive elements will indicate cost to the receiving bus for the transfer. Do we need to calculate an estimated lambda first? Will this help in any way? Let’s assume no, and try to finish implementing this algorithm, despite the issues coming up.

We now have the values of the exports.

Moving on to step 3, we need decide for each bus if the lambda should go up or down. This will entail finding the positive element of the matrix corresponding to each line, and calculating each buses cost to produce it’s own import power. What we’re attempting to do is to find the balance point. We want to know how much it’s going to cost to produce the power we need to export. If we compare how much it’s going to cost us to produce the power we need to export, and it’s more than what we’re receiving to export it, then we need to up our lambda in order to balance. What is meant by balance is that we want to have a net cost of 0. If it costs 100 to produce the power, we need to charge a total of 100 for the exports.

It’s not only exports, we need to see how much it costs to produce what we need to export, as well as what we need to produce for loads, with respect to how much we paid to import.

How exactly will this work? Cost of producing for export + load – cost of the imports? Yeah, let’s try that.

What happens in the case of bus 3? We have a load, but infinite cost to generate?

**October 14, 2020:**

This approach is getting too complicated and bogged down. We need a better method, so back to the drawing board instead of continuing to beat my head against the keyboard. Is there some sort of iterative method starting from the node or from the lines that can ‘exhaustively’ (or a smarter method) look for the best solution for that point?

Let’s go back to the very basics and try to build up a solution from there. (keep it simple, stupid) Base DC OPF.

1. We want to minimize the total cost of generation.
2. We want each nodes lambda value
3. We want to find the amount of power on each line
4. We want to know the relative angle at each node
5. Meet the load

That’s the essential problem. So what are our constants?

1. The cost parameters at each generator
2. Admittance on each line
3. Loads

What is variable?

1. The lambda at each line
2. The power flowing on each line
3. The bus angles
4. Power generated at each bus

The output value of the function is:

1. Total cost to produce the power

What variables are linked?

1. Power flowing on each line
   1. Linked to bus angles
   2. Linked to power generated at each bus

How does this linking help?

If we know one of the three, we can calculate the others. For example, if we know the power generated and consumed at any bus, we can find out where the power is flowing, and also the bus angles. I’ve put it in terms of the power at each bus since this is the primary element to minimize in a global sense.

This reduces me to two variables. Power produced at each bus, and lambda at each bus. What’s the simplest method I can come up with to solve a 2 variable problem like this, without full knowledge of the system (distributed style of solution).

Here’s my plan. Can we make power generated depend on the lambda as well? If we can find or create such a link, then I’ve reduced to a single variable at each bus, which is great since that simplifies the solution.

In a base ED solution, we can, in fact, select power generated as a function of lambda. Does this mean every single one of our variables is linked? Can we assume this linking and sort of force the problem to behave this way for the beginning?

Let’s put together a 2 bus with a simple solution and check it out. Looking at Dr. Ranade’s example system.

If incremental cost functions are:

IC1 = 1 + .0625P1 = Lambda1

IC2 = 1 + .0125P2 = Lambda2

Let these start at:

Lambda1 = 0

Lambda2 = 0

We can express IC1 and IC2 as:

P1 = (Lambda1 – 1)/.0625

P2 = (Lambda2 – 1)/.0125

Now, as each lambda varies, what do we want to keep track of at each bus? We want to minimize C1 and C2, which are the integrals of the IC functions:

C1 = P1 + .0625/2\*P1^2 + const1

C2 = P2 + .0125/2\*P2^2 + const2

What will help us minimize this?

1. Producing less power
2. Increasing export

For example, the load at node 2 must be served by someone. Bus 2 can produce this load, but how much will it cost? Is it more cost effective to import the power? Instead of pure math, we can include a logic step that says, ‘import power as cheaply as possible’. This has the undesired effect of ignoring the transmission system, however. There is also currently nowhere that this load comes into our functions, which it certainly should. This is where the centralized lambda comes into consideration. We need someone to step in and say (essentially), ‘you need to offer more for power, you need to re-evaluate production based on the higher offer’. This goes back to the lambda at each line.

Essentially, for every line, find a lambda the node pair likes.

**October 16, 2020:**

Ignoring the node lambda for now.

For each pair of nodes across a line, we need to find a single lambda such that incremental costs are equal, and the loads are met. How do we handle multiple lines? Let’s look at our three bus, and see what needs to be solved.

With no line limits:

Find a lambda such that all loads are met, and cost is minimized. What’s the formulation for this?

Lambda such that:

P1 + P2 + P3 = Pload

Minimum: C1 + C2 + C3

For each node, this means:

Minimize C1 through a combination of lowering production and/or maximizing exported power

What does this really mean at each node?

Minimize production such that there’s no imbalance between nodes on the lines. How does the node decide which line to pull load from if there is no generation at the bus?

What if we stop looking at this from a node<->node perspective, and start looking as a all node perspective? First of all, it wont work as soon as there are line limits, so who cares if it works.

Went back to Jose’s thesis for clues. The change in the lambda is supposed to be an average of what each bus wants. This should drive them both towards a middle ground.

**October 19, 2020:**

I keep getting stuck asking who does the export go to, but the answer has been right in front of me. It’s part of everyone!

We don’t assign a portion to each line (we could using ptdf, but why). What we need to do is solve *the same* equation for every line. This means net in equals net out. Kirchhoff was sitting in front of me the whole time.

The equation for power out on any spot comes from:

0 = -Pin + Pout + load – Pgen

Or, more clearly, Pgen + Pin = load + Pout

This changes slightly because of out/in at every node to:

Pgen = load + Pout where Pout is the export on every line, if there is import, we end up with a lower Pout.

If we have two nodes connected to one, we have:

Pgen = load + Pn1 + Pn2 where Pn1 & Pn2 are the exports to those respective nodes.

The equation we need to solve at the node is Pn1 = Pgen – load – Pn2

Which can be expressed as:

Pnj = Pgen – load - ∑Pni with i != j

So how does this change the way our program should work at each node?

1. Calculate Pgen with lambda.
2. Calculate Pni using the load and the new Pgen
3. Lines add Pni from each node, if <0 increase lambda, if >0 decrease lambda

For now, we can ignore the angles, we don’t need them for the power calculations. Once we get the power balance correctly, we’ll worry about getting the angles. This is so much simpler than I was making it.

Have a system that works for the two nodes, but how can we minimize the error without adding too much time? We need alter lambda by small increments as we get closer to the result. We need to add zeros inversely proportionate to the size of the sum of the two lines power exports.

Have a functional system for 2-bus, how about our three bus?

Not working smoothly with 3-bus:

Pgen1: -1264.404609475032

Pgen2: -2023.1958762886595

Pgen3: 0

Lambda 1: 44480

Lambda 2: -44472

Lambda 3: -44480

Alright. I found the problem. In our iterations, we are changing the line export as we go. What this means for us is that at node 3, we have a load and no generation, we put our entire load onto the first line, then none on the second line. This causes the load to be completely satisfied off the first iteration using whatever the first line wants to charge.

Let’s try to prevent this by assigning a percentage of the load to each line based off lambda. Essentially a current divider to find lowest cost source.

The factor of each line will be:

Ld1/(ld1 + ld2) for the line 2, this weight should also come into the lambda selection for how much we generate. Actually it already should, so let’s apply this to the problem and see how it changes.

The problem with this is that node three can only consume what node 1 or 2 is willing to send it. How do we enforce this in a distributed manner. That’s what the lambda is for, we’re trying to find a line lambda such that both nodes are happy to send a certain amount of power.

Should we treat power produced uniquely for this? We should be using each lines lambda and sum the generation values after the fact? This is where we push past Dr. Ranade’s implementation and into new territory.

Unbelievably, that actually worked! The only corrective measure I can see that we need is that instead of using the average lambda to calculate power produced, we need to use individual line production summed to total generation.

This comes up with:

Pgen1: 299.8719590228673

Pgen2: 308.9175257699212

Pgen3: 0

Lambda 1: 8.251499999994108

Lambda 2: 8.58529999999333

Lambda 3: 8.647099999993186

The desired values were:

Pgen1: 161.21141252

Pgen2: 138.76322916

Pgen3: 0

Lambda 1: 8.92724891

Lambda 2: 8.92680266

Lambda 3: 8.9262879

As we can see, we doubled the load, so what happened? Each generator is producing double the power of the system balance lambda, this is because we split the export to each line. We’re seeing each node satisfy all the power required by the load. I’m guessing we have no power on line one, and each node is sending all the load down the other line.

What I’m seeing: all lines have strange mis-matches, nothing is actually settling properly. The algorithm is not behaving as expected. The algorithm isn’t implemented as expected, let’s go back through and see if we can find an error.

**October 20, 2020:**

Today was spent doing minor changes and trying to understand the flow of the problem to understand where something is going wrong.

**October 21, 2020:**

Went back to the 2-node system, going to try to build up slowly to see if we can find the error point.

**First modification:** remove the load from node 2, see what happens (expecting 0 power transfer).

Lambda settled to 1, generators chose to produce essentially 0 power, with 0 on the line.

*Node 1 power: 8.881784197001252e-15*

*Node 2 power: 4.440892098500626e-14*

*Power transfers on line: {<\_\_main\_\_.node object at 0x000001F3A2F5F470>: 8.881784197001252e-15, <\_\_main\_\_.node object at 0x000001F3A2F5F438>: 4.440892098500626e-14}*

*Line 1 lambda: 0.9999000000000011*

**Second modification:** Add a third node, with no generation, and also no load.

No lambda settle, even power on line 1 can’t settle, both generators want to buy power.

*Node 1 power: 232.0784000000001*

*Node 2 power: 1000.8080000000007*

*Node 3 power: 0*

*Power transfers on line 1: {<\_\_main\_\_.node object at 0x00000122A174F4A8>: 232.0784000000001, <\_\_main\_\_.node object at 0x00000122A174FE80>: 116254.44400002726}*

*Power transfers on line 2: {<\_\_main\_\_.node object at 0x00000122A174FE80>: -116414.02800002726, <\_\_main\_\_.node object at 0x00000122A174FEB8>: 0}*

*Line 1 lambda: 29.009800000000013*

*Line 2 lambda: -1.9895999999999958*

**Third modification:** Add a 100MW load to node3.

Still no settling, this is definitely where the problem is coming from.

**Fourth modification:** With the preceding modifications, we can see that there’s clearly a problem when we have a node that doesn’t generate any power. Allow node3 to have a similar generator.

We can see this is creating a problem for both nodes 1 and 3, node three is producing all its power, and node 1 still wants to buy power from node 3. The lines are still nowhere near balancing properly.

*Node 1 power: -55.92799999999998*

*Node 2 power: 0.7840000000000487*

*Node 3 power: 140.21199999999996*

*Power transfers on line 1: {<\_\_main\_\_.node object at 0x000001A6D55BF4A8>: -55.92799999999998, <\_\_main\_\_.node object at 0x000001A6D55BFE80>: 561.5040000011054}*

*Power transfers on line 2: {<\_\_main\_\_.node object at 0x000001A6D55BFE80>: -281.0800000011054, <\_\_main\_\_.node object at 0x000001A6D55BFEB8>: 40.21199999999996}*

*Line 1 lambda: -6.990999999999998*

*Line 2 lambda: 9.010599999999998*

Lambdas are rubber-banding excessively. They’re going from negative to positive and back and forth very quickly. What if each pair of nodes needs to settle before adding in another? That would take entirely too long to complete.

Assuming our lambda calculation is correct, we need to look at the bus power calculations a bit closer.

This is where the implementation falls apart. It works with one line, but falls apart quickly when there are more than one lines.

We almost want to ensure that the generators purchase cannot exceed it’s exports + load?

Right now, we’re ensuring that generation equals load and exports, reorganized to say that an export cannot be more than the sum of imports and generation minus. So why is our bus trying to import huge amounts of power and eat it with the generator? To compensate, duh. So let’s limit the generation to a minimum of 0.

This produces:

*Node 1 power: 0*

*Node 2 power: 0*

*Node 3 power: 99.99800000000046*

*Power transfers on line 1: {<\_\_main\_\_.node object at 0x000001CA5457F4A8>: 0, <\_\_main\_\_.node object at 0x000001CA5457FE80>: 0.0}*

*Power transfers on line 2: {<\_\_main\_\_.node object at 0x000001CA5457FE80>: 0.0, <\_\_main\_\_.node object at 0x000001CA5457FEB8>: -0.0019999999995405915}*

*Line 1 lambda: -4.00010000000002*

*Line 2 lambda: 6.000000000000023*

Which essentially says lines say, “Screw it, we aren’t exporting.” And gen3 produces all the power it needs.

What about if node 3 has 0 generation capacity?

Lambda went out of whack. Fast.

*Node 1 power: 0*

*Node 2 power: 20.398000000277534*

*Node 3 power: 0*

*Power transfers on line 1: {<\_\_main\_\_.node object at 0x00000231E0C1F4E0>: 0, <\_\_main\_\_.node object at 0x00000231E0C1F4A8>: 20.398000000277534}*

*Power transfers on line 2: {<\_\_main\_\_.node object at 0x00000231E0C1F4A8>: 0.0, <\_\_main\_\_.node object at 0x00000231E0C1FE80>: -100}*

*Line 1 lambda: -108330.9901*

*Line 2 lambda: 108334.01000000001*

This disagrees with Dr. Ranades slides anyway, the generators need to rubber band then find a normal.

Back to basics, keep it simple.

Power generatred = sum of the power we want to export on every line at any time = (ld1-a)/b^2 + (ld2-a)/b^2

This needs to include loads, we have the value if we generate power, but should we be looking at nodal injection? So a negative if we need to absorb power?

If we’re trying to minimize cost, we need to look at four things at each node. We need to minimize our cost by balancing production, imports, exports and loads. We need to balance our power in versus power out. We need to essentially balance costs of production and import with profits of export while also meeting our load.

Bottom line is: We must meet our loads, we do this via production or import whichever is cheapest. So let’s scrap our current method of deciding on power generation and try to design a method for this. With the final values needing to satisfy Pgen + imports = Pload + exports Essentially we’re lacking by holding imports and exports too loosely. We have three controllable variables: Pgen, imports, exports. We have a single static value which is load.

For every MW, we need to check: is it cheaper to produce each MW or to buy it at a line’s lambda? Since MW cost is incremental for the generator, but ‘constant’ for the lines, we should be able to find a balance point where the nodal lambda equals the lambda of the lines. **We need to split any load such that we equally divide the cost between generation + imports.**

**Pgen\*(A + Pgen\*2\*b) = Pimport1\*ld1 = Pimportn\*ldn Essentially, the amount spent on generation must be equal to the amount spent importing from each source.**

We need to fill the cheapest source first, if it’s cheaper to produce the first MW, then we produce instead of buy. How can we express this mathematically?

We want to produce enough power such that **A + Pgen\*2\*b = ldlines where ldlines is the average cost to import/export. We then need to assign excess of our load to these lines depending on the cost of each. Alternatively, if the Pgen is less than load, we need to split the import among these lines.**

***Pgen = (ldlines – a)/(2b)***

***If Pgen > Pload, export the power to the line that pays the most***

***If Pgen < Pload, import the power from the line that charges the least***

This doesn’t cover transfers. What about pass-through nodes that say(ld1 is < ld2) buy from l1 and sell to l2?There’s no penalty for selling infinite power, or buying infinite power, so how do we prevent this from happening?

What about when the lines have equal lambda? We split the import/export values across the best candidates?

Ldlines should be the *cheapest* line’s lambda for calculation. All power import comes from the cheapest line anyway.

**October 22, 2020:**

If a + infinity and a -inifinity case come into existance, it simply tells us that the current lambda is not the system lambda, it should rubber band back from extremes over time. On our three bus, we will always have this mismatch till we touch the right lambda. How do we program a node to do this type of buying and selling?

1. Calculate produced power using average cost of power from lines.
2. Decision: buy or export (load only node will have to buy)
3. Assign export to most expensive line
   1. Import to cheapest line
4. Evaluate for profits via purchase and sell due to lambda difference
   1. If we can make a profit, go ahead and assign +- infinity in those cases?

**October 26, 2020:**