Introduction

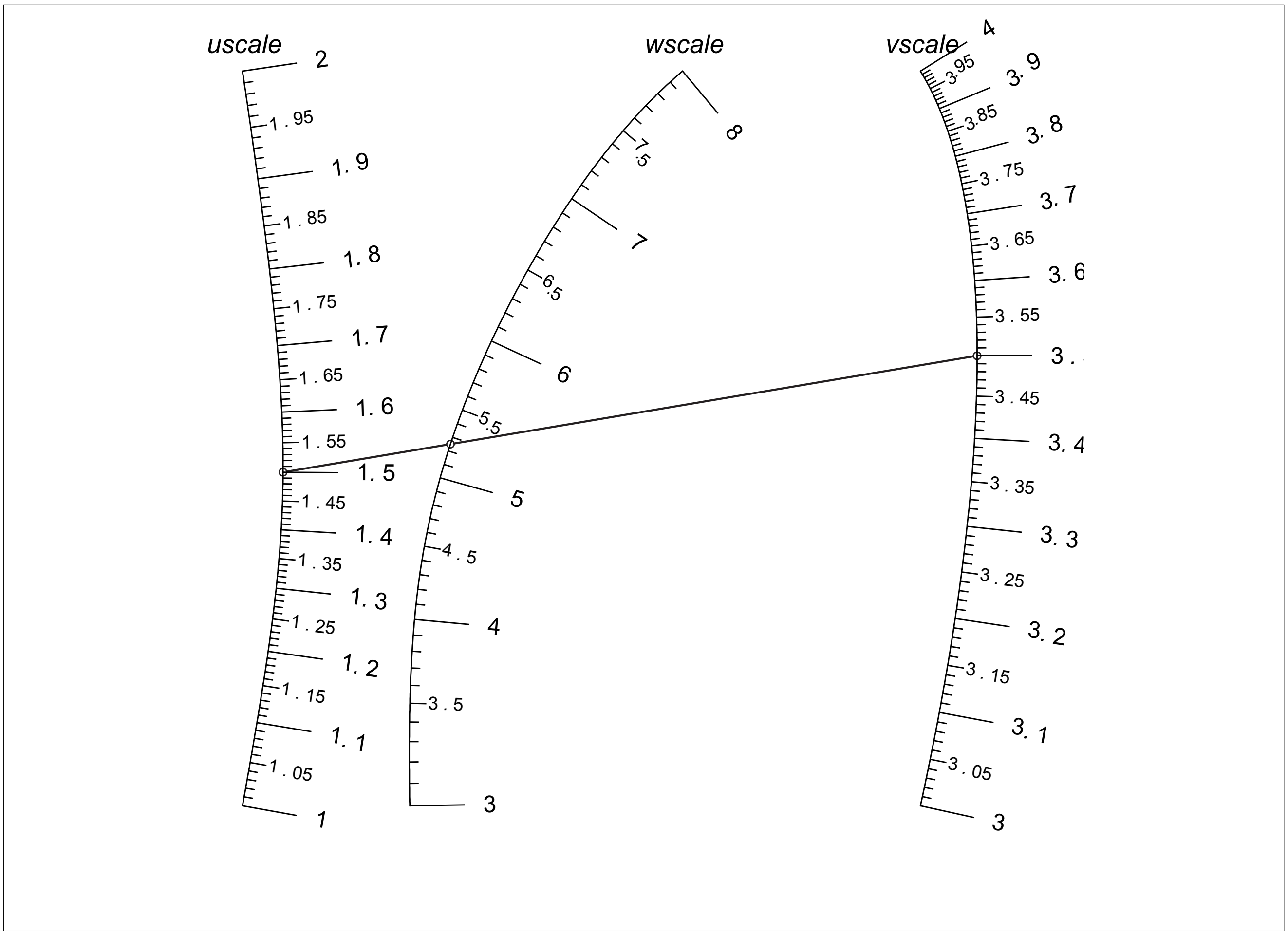
Nomgen makes a nomogram quick & easy to create by auto-generating the scale lines from a given function of 2 variables.

Enter formula, max & min values, generate a nomogram, possibly (probably) non linear.

- use Python to calculate nomogram numerically using SciPy libraries
- output using pynomo libraries

Note some formulas cannot have a nomogram, so need to be able to recognise this.

A nomogram with M lines, N of them curved is known as Class M, Genus N. Thus this project attempts to auto-generate class 3, genus 3 nomograms.



Using nomogen

Assume you have a python3 setup with pynomo, etc., as described in reference [2.].

Write a python function of 2 arguments, corresponding to the left and right hand scales, with the return value corresponding to the middle scale.

You may need to solve your equation numerically if it is an Implicit function. See colebrookf.py for example.

nomogen determines the equations of the scale line for you, but you still need to arrange the nomogram layout for things like the nomogram paper size, title, etc., as described in the pynomo documentation. Also, Ron Doerfler's excellent essay [2.] is definitely worth checking out for instructions on setting up the nomogram.

Nomogen uses your equation to derive and set up the determinant equations, so there's no need to do any of that.

The scale lines are represented by polynomials, so nomogen needs the polynomial degree to use. The more straight the scale lines, and the more evenly spaced the intervals, the lower the polynomial degree needed. Use trial and error, starting with a degree of about 7.

Smaller is faster, larger is more accurate.

A scale line can be configured for 'linear smart' if the scale is approximately evenly spaced. If the high values on the scale are bunched together tighter, the scale type might be better set to 'log' or 'log smart'. Nomogen plots the log of these scale variables, so the program might be faster and more accurate.

Note that some functions are not accurately representable as a nomogram, so reducing the range of one or more scales should improve accuracy.

When a nomogram is calculated, nomogen reports an estimate of the tolerance. If the tolerance is larger than 0.1mm, it is shown on the nomogram.

There's lots of reference examples included with nomogen.

Step by Step Example

Let's take example (xv) from Allcock & Jones, ref [3.], the relative load on a retaining wall.

Details are here:

https://babel.hathitrust.org/cgi/pt?id=mdp.39015000960115&view=1up&seq=128

The formula is:

$$(1+L)h^2 - Lh(1+p) - \frac{1}{3}(1-L)(1+2p) = 0$$

with

$$0.5 \le L \le 1$$

$$0.5 \le p \le 1$$

thus

$$0.75 \le h \le 1 \quad .$$

The nomogram is to fit inside a 10 cm x 15 cm rectangle.

Use example file test1.py as a template, copy it to, say, myproj.py and edit that.

Write a new function **AJh(L, p)** replacing (or perhaps ignoring) the existing w(u,v). A beginners tutorial is given in ref [2.].

The function above cannot be rearranged to give a value for h (it's a so-called implicit function of h), but it is a function of the form

$Ah^2 + Bh + C = 0.$, from which h can be found using the quadratic formula.

Assigning values for the maximum and minimum of each of the variables gives a code fragment something like this:

```
def AJh(L,p):
    a = 1+L
    b = -L*(1+p)
    c = -(1-L)*(1+2*p)/3
    h = ( -b + math.sqrt(b**2 - 4*a*c) ) / (2*a)
    return h

Lmin = 0.5; Lmax = 1.0;
pmin = 0.5; pmax = 1.0;
hmin = AJh(Lmax, pmin);
hmax = AJh(Lmin, pmax);
NN = 3
```

The variable NN is the degree of the polynomials that define the scale lines. It can be tricky to get right, but 3 works well for this example.

Now define the scales, with **L** and **p** on the outer scales and **h** (the function value) in the middle. For each scale, set the minimum & maximum values, and (optionally) a title. Leave the other definitions as they are:

```
left_scale = {
    'u_min': Lmin,
    'u_max': Lmax,
    'title': r'$L$',
    'scale_type': 'linear smart',
    'tick_levels': 3,
    'tick_text_levels': 2
}

right_scale = {
    'u_min': pmin,
    'u_max': pmax,
    'title': r'$p$',
    'scale_type': 'linear smart',
    'tick_levels': 3,
    'tick_text_levels': 2 ,
}

middle_scale = {
    'u_min': hmin,
    'u_max': hmax,
    'title': r'$h$',
    'scale_type': 'linear smart',
    'tick_levels': 3,
    'tick_text_levels': 2
}
```

Note that the title text needs to go between the ‘$’ characters, as in r’$myTitle$’. It is not always obvious how to add text, so it’s worth checking ref [2.] beforehand.

In the main_params section, set the nomogram height and width, in cm, and add a title:

```
main_params = {
    'filename': __file__.endswith(".py") and __file__.replace(".py", ".pdf")
or "nomogen.pdf",
    'paper_height': 15, # units are cm
    'paper_width': 10,
    'title_x': 4.5,
    'title_y': 1.5,
    'title_str':r'$example$',
    'block_params': [block_params0],
    'transformations': [('scale paper',)],
    'pdegree': NN
}
```

The parameters **‘title_x’** & **‘title_y’** are the x and y positions of the title, measured in cm from the bottom left of the nomogram.

Finally, give nomogen the function defined above:

```
Nomogen(AJh, main_params);  # generate nomogram for AJh function
```

That’s it! Now run the program, and a neatly generated nomogram will be in myproj.pdf.

The pynomo documentation and ref[2.] describe lots of other options for setting up and fine tuning the nomogram. See also the example Ajh.py.

Compare this with the calculations shown in the example in ref [3.], which needs several pages of algebra and geometric theory to derive the equations and fill a determinant to calculate the curves for the nomogram. There’s no need for that with nomogen, just write the function and set the parameters.

References

1. PyNomo documentation
2. http://www.myreckonings.com/pynomo/CreatingNomogramsWithPynomo.pdf
3. Allcock and Jones: The Theory and Practical Construction of Computation Charts