Introduction

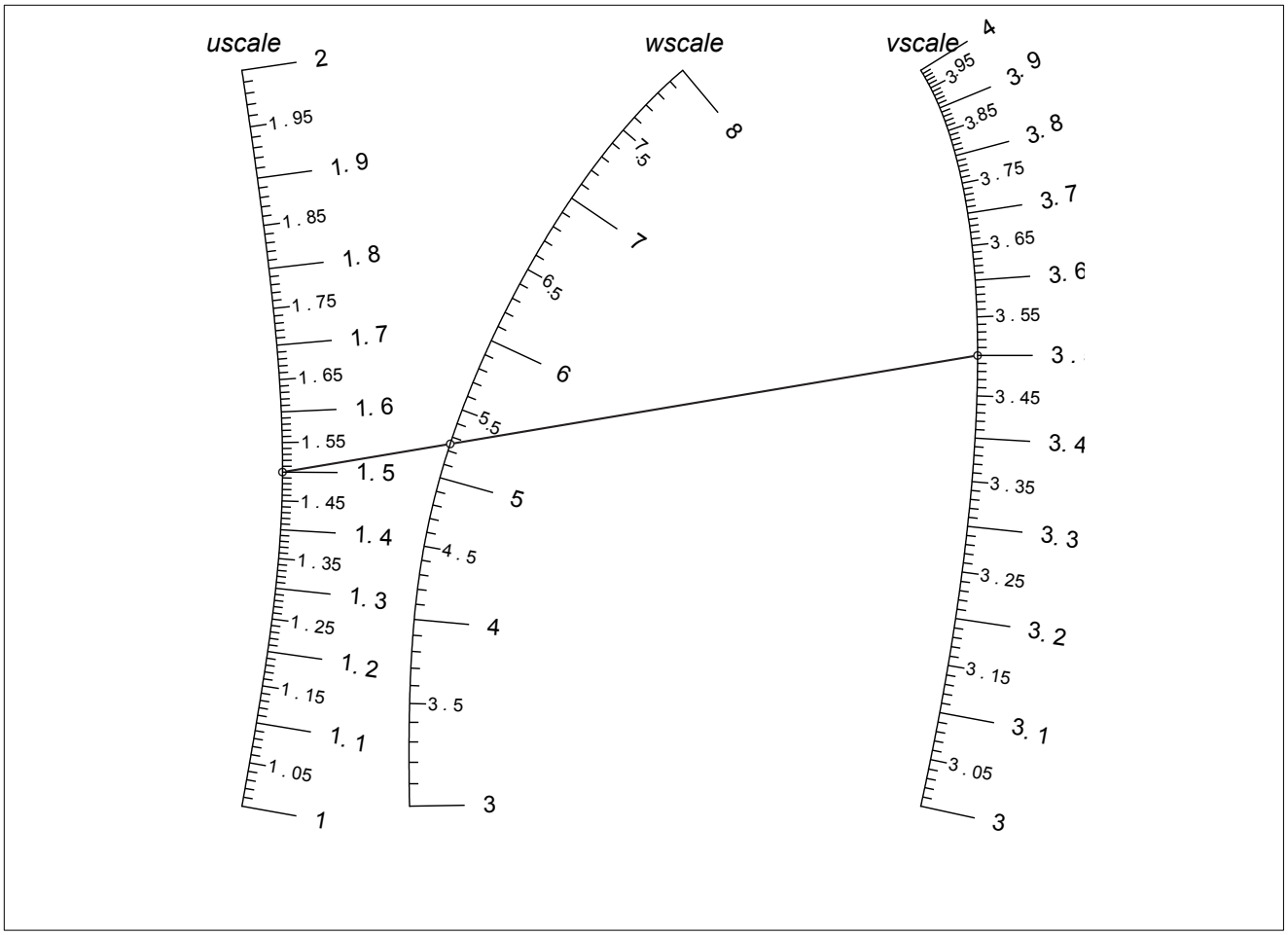
Nomgen makes a nomogram quick & easy to create by auto-generating the scale lines from a given function of 2 variables.

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

- use Python to calculate nomogra ,m 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.

Implicit function? Can solve numerically, e.g. see colebrookf.py

Set up 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 set up the determinant equations, so don't assign to these parameters.

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.

Smaller is faster, larger is 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. Use trial and error, starting with a degree of about 7.

When a nomogram is calculated, nomogen reports an estimate of the tolerance. If the tolerance is larger than 0.01mm, it is shown on the nomogram.

There's lots of examples to refer to 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.

Adding 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);
```

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$’. Adding text is not always obvious, 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 a handful of parameters.

Deriving a Nomogram

Use tried and tested **engineering principles**:

- use others' good work, just hook it all together
- use brute force. (given to us by Moore's law.)

The nomogram is calculated to **fit inside a unit square**. This is achieved by one of

1. tie the ends of the outer scales to the corners of the unit square *(current scheme)*
2. use a cost function that rewards the nomogram as it gets larger up to the boundary of the unit square, and then penalises it if it extends outside the unit square. *(possible future scheme)*

Method outline:

- Approximate each scale line with a polynomial,
- estimate the cost of each approximation. The cost is a function of the error in the scale curves and how well the curves fit into the allocated area
- use SciPy numerical optimisation library to minimise this cost,
- use pynomo libraries to scale and plot the nomogram.

The Polynomials

Each scale line is defined by a polynomial, not with a set of coefficients like $u_x = \sum_{k=0}^{N} \alpha_i T_i(u)$, but by interpolating its values at so-called Chebychev nodes.

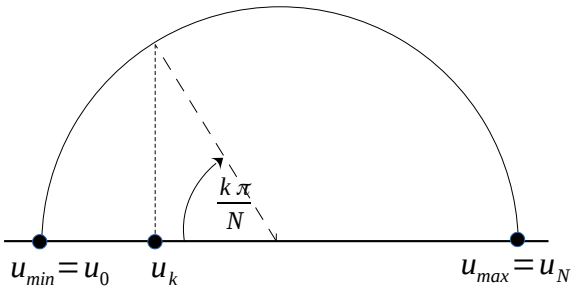
A major problem with using equally spaced nodes to approximate the scale lines (or any function in general) is that the approximation can deviate wildly near the end points.

Polynomials defined at the Chebyshev nodes counter this by (very nearly) minimising the maximum error.

If there are N+1 Chebyshev nodes for the u curve, they occur at the points

$$u_k = u_{min} + (u_{max} - u_{min})(1 - \cos(\frac{k * \pi}{N}))/2, \text{ where k = 0,1, .. N}$$

These nodes can be interpreted as the projection from a semi-circle onto the u axis of equally spaced angles, so node u_k is the projection from an angle $k\frac{\pi}{N}$:



This scheme bunches the nodes more densely near the ends which keeps the approximation close to its correct value.

If we have values of the function at these points, there is a unique N-degree polynomial that fits these points.

We determine the coordinates of the u, v & w curves at the Chebyshev nodes, and from there generate the curves of the nomogram.

Given x_k & y_k, the x and y coordinates of each of these nodes, we can efficiently interpolate the x coordinates of any point w as follows:

(This code assumes N is odd)

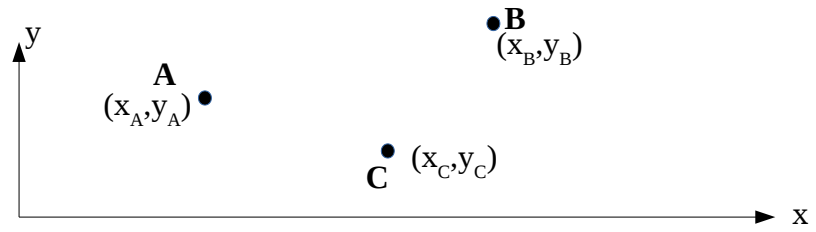
```
if w is one of wₖ then
    use xₖ & yₖ, the known values for node k
else
    sumA = (x₀/(w-w₀) + xₙ/(w-wₙ))/2;
    sumB = (1/(w-w₀) + 1/(w-wₙ))/2;
    for k in 0 .. N
        t = 1/(w-wₖ);
        sumA = t*xₖ - sumA;
        sumB = t - sumB;
x(w) = sumA / sumB
```

The y coordinate, y(w) is very similar.

See ref[4], which also describes how to calculate the first 2 derivatives of the function.

Basic Nomogram Calculations

Suppose we have 3 points, A, B & C in an x-y coordinate system:



The area inside the triangle ABC is

$$Area = \frac{1}{2}((x_A - x_B)(y_A - y_C) - (x_A - x_C)(y_A - y_B))$$

(This is the vector cross product of the 2 line segments. It is calculated like a determinant, but technically, it is not a determinant.)

- The points A,B,C lie on a straight line if

  Area = 0, or

  $$(x_A - x_B)(y_A - y_C) = (x_A - x_C)(y_A - y_B)$$

  $$x_A y_B + x_B y_C + x_C y_A - x_A y_C - x_B y_A - x_C y_B = 0$$

- The distance of the point C from the line AB is

  $$d = \frac{2 * Area(ABC)}{base\ line\ AB}$$

  $$d = \frac{(x_A - x_B) * (y_A - y_C) - (x_A - x_C) * (y_A - y_B)}{\sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}}$$

We wish to generate a nomogram for the formula

$$w = w(u, v), \quad \text{where } u_{min} <= u <= u_{max},\ v_{min} <= v <= v_{max}$$

The scale lines are approximated by parametric curves with coordinates given by polynomial functions of u, v and w:

u = (x_u, y_u), where $x_u = x_u(u)$ and $y_u = y_u(u)$

and so on for the v & w scale lines.

This gives us functions $x_u(u)$, $y_u(u)$, etc., and since for any u & v such that w = w(u,v), the nomogram points must lie on a straight line, we can write

$$\frac{(u_x - v_x)}{(u_y - v_y)} = \frac{(u_x - w_x)}{(u_y - w_y)} \quad \text{or}$$

$$(u_x - v_x)(u_y - w_y) = (u_x - w_x)(u_y - v_y) \tag{1}$$

multiply throughout:

$$u_x v_y - u_x w_y - v_x u_y + v_x w_y + w_x u_y - w_x v_y = 0$$

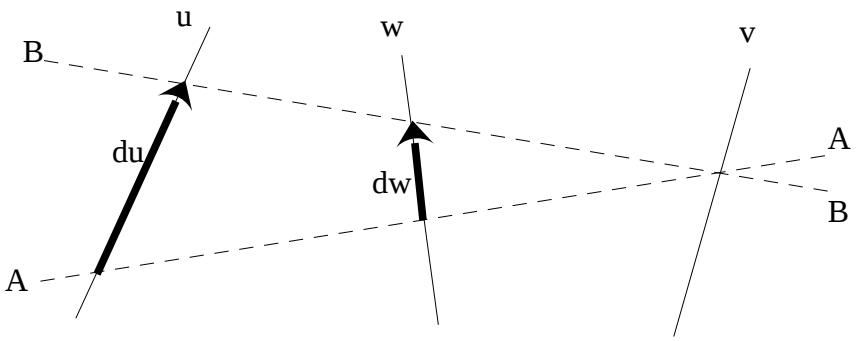
This is equivalent to the determinant:

$$\begin{vmatrix} u_x(u) & u_y(u) & 1 \\ v_x(v) & v_y(v) & 1 \\ w_x(w) & w_y(w) & 1 \end{vmatrix}$$

which can be used as input for a type 9 nomogram in pynomo (see refs [1] and [2]).

Derivatives

Consider a small segment of the u, v & w scale lines as follows:



The index line AA intersects the scale lines at (x_u,y_u), (x_v,y_v) & (x_w,y_w).

Now perturb AA by an amount du, leaving its intersection with the v scale line unchanged. This is index line BB.

The scale lines are defined by the equations $x_u = x_u(u)$, etc., and the function w is defined as w = w(u,v)

We can write the following since the intersection points on the lines are co-linear:

$$(x_u - x_v)(y_w - y_v) = (x_w - x_v)(y_w - y_v) \tag{2}$$

$$\left(x_u + \frac{dx}{du}du - x_v\right)\left(y_w + \frac{dy}{dw}dw - y_v\right) = \left(x_w + \frac{dx}{dw}dw - x_v\right)\left(y_u + \frac{dy}{du}du - y_v\right) \tag{3}$$

Expanding (3):

$$\left(x_u+\frac{dx}{du}du-x_v\right)y_w+\left(x_u+\frac{dx}{du}du-x_v\right)\frac{dy}{dw}dw-\left(x_u+\frac{dx}{du}¿du-x_v\right)y_v$$
$$=\left(x_w+\frac{dx}{dw}dw-x_v\right)y_u+\left(x_w+\frac{dx}{dw}dw-x_v\right)\frac{dy}{du}du-\left(x_w+\frac{dx}{dw}dw-x_v\right)y_v$$

Dropping second order terms & substituting (2):

$$y_w\frac{dx}{du}du+\left(x_u-x_v\right)\frac{dy}{dw}dw-y_v\frac{dx}{du}du=y_u\frac{dx}{dw}dw+\left(x_w-x_v\right)\frac{dy}{du}du-y_v\frac{dx}{dw}dw$$

$$\left(y_w-y_v\right)\frac{dx}{du}du+\left(x_u-x_v\right)\frac{dy}{dw}dw=\left(y_u-y_v\right)\frac{dx}{dw}dw+\left(x_w-x_v\right)\frac{dy}{du}du$$

We also know

$$dw=\frac{\partial w}{\partial u}du+\frac{\partial w}{\partial v}dv \quad ,$$

and since in this case dv = 0, we have:

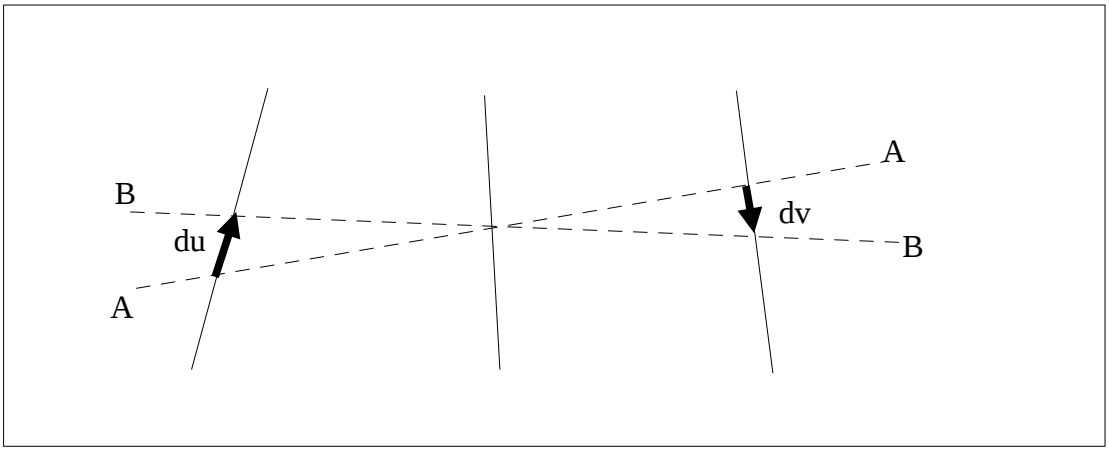
$$dw=\frac{\partial w}{\partial u}du \quad \text{densly}$$

$$\frac{\partial w}{\partial u}\left(\left(x_u-x_v\right)\frac{dy}{dw}-\left(y_u-y_v\right)\frac{dx}{dw}\right)=\left(x_w-x_v\right)\frac{dy}{du}-\left(y_w-y_v\right)\frac{dx}{du} \qquad (4)$$

There is a similar equation for an index line that shifts along the v scale line while keeping the u value constant:

$$\frac{\partial w}{\partial v}\left(\left(x_u-x_v\right)\frac{dy}{dw}-\left(y_u-y_v\right)\frac{dx}{dw}\right)=\left(x_u-x_w\right)\frac{dy}{dv}-\left(y_u-y_w\right)\frac{dx}{dv} \qquad (5)$$

Now move an index line by a small amount, but keep the w point unchanged:



As before, we can write:

$$\frac{x_u - x_w}{y_u - y_w} = \frac{x_w - x_v}{y_w - y_v}$$, and

$$\frac{x_u + \frac{dx}{du}du - x_w}{y_u + \frac{dy}{du}du - y_w} = \frac{x_v + \frac{dx}{dv}dv - x_w}{y_v + \frac{dy}{dv}dv - y_w}$$

$$\left(x_u + \frac{dx}{du}du - x_w\right)\left(y_v + \frac{dy}{dv}dv - y_w\right) = \left(x_v + \frac{dx}{dv}dv - x_w\right)\left(y_u + \frac{dy}{du}du - y_w\right)$$

$$\left(y_v - y_w\right)\frac{dx}{du}du + \left(x_u - x_w\right)\frac{dy}{dv}dv = \left(y_u - y_w\right)\frac{dx}{dv}dv + \left(x_v - x_w\right)\frac{dy}{du}du$$

Now, from $w = w(u, v)$, we get

$$dw = \frac{\partial w}{\partial u}du + \frac{\partial w}{\partial v}dv$$

In this case, $dw = 0$, so

$$-\frac{\frac{\partial w}{\partial u}}{\frac{\partial w}{\partial v}}du = dv$$

Substitute into the above, then tidy up:

$$\frac{\partial w}{\partial v}\left(\left(x_w - x_v\right)\frac{dy}{du} - \left(y_w - y_v\right)\frac{dx}{du}\right) = \frac{\partial w}{\partial u}\left(\left(x_u - x_w\right)\frac{dy}{dv} - \left(y_u - y_w\right)\frac{dx}{dv}\right) \quad (6)$$

NOTE: this is not independent of 4 & 5. This can be seen by dividing those 2 equations, then cancelling the common term and cross multiplying the denominators.

So, the coordinates of any index line must satisfy equations 2, 4 & 5. This information can be used to:

1. help determine an initial estimate
2. constrain the numerical solution for a nomogram

TODO: this is just the first term of a Taylor's series about the index line. Is it helpful to go further?

The error for a given approximation

Given a candidate set of coordinates for each of the u,v & w scales, determine an error function as

$E = \sum_{i,j=0}^{N} e_{ij}^2$, where

e_{ij} = distance of $w_{ij} = w(u_i, v_j)$ from line determined by u_i & v_j

TODO: add derivatives

- Create an error function by summing the squares of all errors over all nodes for u & v
- Use SciPy to minimise this error as a function of the node coordinates
- if the minimum error is acceptably small, then the solution is the nodes that define the nomogram, otherwise a solution cannot be found.

The nomogram is generated on a unit square, which PyNomo scales to the required size when it creates the output.

If there are N Chebychev nodes for each scale line, there are 6N unknown values and N(N+1)/2 equations. Fixing the ends of the outer scales to the 4 corners of the unit square leaves 6N-8 unknowns. The boundary between the system being over determined and under determined occurs when the number of equations equals the number of unknowns:

$$N(N+1)/2 = 6N - 8$$

$$N^2 - 11N + 16 = 0$$

N = 7.55

(TBC, consider derivatives?)

Initial Estimate

The optimisation algorithms need a good initial guess to converge on the optimal solution nomogram.

Which way are the scales oriented?

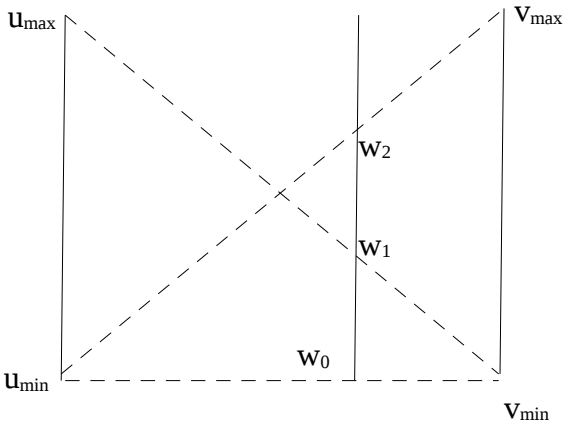
If we assume that the u scale increases (varies from min to max) as the y coordinate increases, and if we define

$$w_0 = w(u_{min}, v_{min})$$

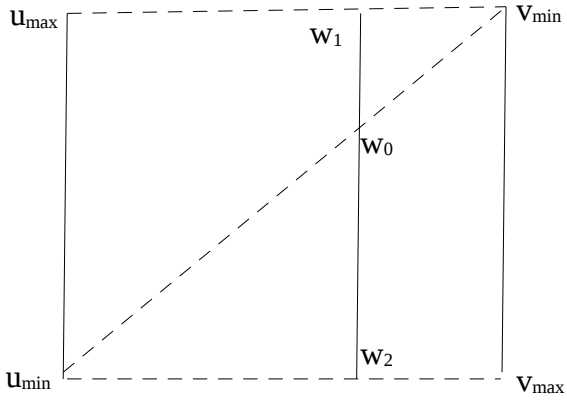
$$w_1 = w(u_{max}, v_{min})$$

$$w_2 = w(u_{min}, v_{max})$$

then the w scale increases upwards if $w_1 > w_0$ and the v scale increases upwards if $(w_1 > w_0) = (w_2 > w_0)$, i.e. w_1 and w_2 are both less than, or both greater than w_0, as can be seen in the diagrams below:



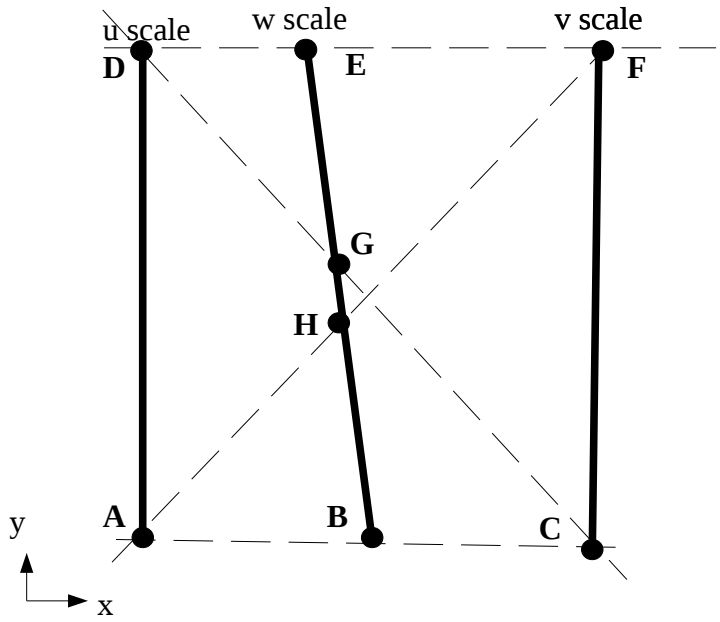
Here, the v scale increases upwards and both w_1 and w_2 must be greater than w_0 if the w scale increases upward, or they both must be less than w_0 if the w scale increases downwards.



Here, the v scale increases downwards and w_0, must lie between w_1 and w_2. As above, w_1 is greater than w_0 if the w scale is increasing upwards, otherwise w_1 is less than w_0 if the w scale increases downwards

A Simple Linear Estimate

Try to approximate the true nomogram with a simple linear one as follows:



The u, v & w scale lines are respectively **AD**, **CF** & **BE**.

Point **A** is at (0,0), **C** is at (1,0), **D** is at (0,1), and **F** is at (1,1), u varies from u_{min} to u_{max} & v from v_{min} to v_{max}. Four index lines are shown, and their points of intersection with the scale lines are marked **A** to **H**. The x coordinate of u is given by the function $x_u = x_u(u)$, with similar functions for the y coordinate and the variables v & w.

We have for the equations of each of the scale lines:

$$u_x = 0$$

$$u_y = \frac{u - u_{min}}{u_{max} - u_{min}}$$

$$v_x = 1$$

$$w_x = c + \alpha_{wx}(w - w_B)$$

$$w_y = \alpha_{wy}(w - w_B)$$

If the v scale has v_{max} at the top, then define

$$v_{top} = v_{max}, v_{bottom} = v_{min}$$

Otherwise, the v scale is reversed so that v_{min} is at the top, then define:

$$v_{top} = v_{min}, v_{bottom} = v_{max}$$

We can now write

- $$v_y = \frac{v - v_{bottom}}{v_{top} - v_{bottom}}$$

- **G** is where the scale line for w intersects the index line for u=u_{max}, v=v_{bottom}, so w_G=w(u_{max}, v_{bottom}), and similarly, w_H=w(u_{min},v_{top}), w_B=w(u_{min},v_{bottom}) & w_E=w(u_{max},v_{top}).

Note that point **B** is at (c,0), **E** is at ($c+\alpha_{wx}(w_E-w_B)$, 1), and **G** & **H** intersect the diagonal index lines, so

$$\alpha_{wy}=\frac{1}{w_E-w_B} \tag{7}$$

$$c+\alpha_{wx}(w_H-w_B) = \alpha_{wy}(w_H-w_B) = \frac{w_H-w_B}{w_E-w_B} \tag{8}$$

$$c+\alpha_{wx}(w_G-w_B) = 1-\alpha_{wy}(w_G-w_B) = 1 - \frac{w_G-w_B}{w_E-w_B} = \frac{w_E-w_G}{w_E-w_B} \tag{9}$$

Equation (7) gives α_{wy} and (8) & (9) can be solved to give:

$$\alpha_{wx}=\frac{w_E-w_G-w_H+w_B}{(w_E-w_B)(w_G-w_H)}$$

$$c = \frac{w_H-w_B}{w_E-w_B} - \alpha_{wx}(w_H-w_B)$$

Problems that might occur:

1. this approximation can leave w_B & w_E outside the unit square.
   A simple way to address this is to clip the values so that $0 \le w_B, w_E \le 1$
2. w_G might equal w_H, leading to a division by zero in the equation above.

Case 1: w scale outside unit square

(Work in progress ...)

Another way to address the problem of the line extending outside the unit square is to add a quadratic term to the formula for x_w:

$$x_w=c+\alpha_{wx}(w-w_B)+b(w-w_B)^2 \tag{10}$$

and choosing b, c and α_{wx} so that w_B & w_E lie inside the unit square.

Equations (8) & (9) above are modified to:

$$c+\alpha_{wx}(w_H-w_B) + b(w_H-w_B)^2 = \alpha_{wy}(w_H-w_B) = \frac{w_H-w_B}{w_E-w_B} \tag{11}$$

$$c+\alpha_{wx}(w_G-w_B) + b(w_G-w_B)^2 = 1-\alpha_{wy}(w_G-w_B) = 1 - \frac{w_G-w_B}{w_E-w_B} \tag{12}$$

Solve by eliminating c, then rearrange so that c and α_{wx} are functions of b ….

$$\alpha_{wx}(w_G - w_H) + b(w_G - w_H)(w_G + w_H - 2w_B) = 1 - \frac{w_G + w_H - 2w_B}{w_E - w_B} \quad \text{, or}$$

$$\alpha_{wx} = \frac{w_E - w_G - w_H + w_B}{(w_G - w_H)(w_E - w_B)} - b(w_G + w_H - 2w_B) \tag{13}$$

$$c = \frac{w_H - w_B}{w_E - w_B} - \alpha_{wx}(w_H - w_B) - b(w_H - w_B)^2 \tag{14}$$

$\alpha_{wx} = R + bS$ and $c = P + Qb$, where P, Q, R & S are the constants

$$R = \frac{w_E - w_G - w_H + w_B}{(w_G - w_H)(w_E - w_B)}$$

$$S = 2w_B - w_G - w_H$$

$$P = \frac{w_H - w_B}{w_E - w_B} - R(w_H - w_B) \quad \text{and}$$

$$Q = (w_H - w_B)S + (w_H - w_B)^2$$

Requiring that w = w_E and w = w_B lie on the top and bottom edges of the unit square gives these inequalities:

$0 \le c \le 1$, and $0 \le c + \alpha_{wx}(w_E - w_B) + b(w_E - w_B)^2 \le 1$

$0 <= P + Qb <= 1$, or

$$b >= \frac{-P}{Q}$$

$$b <= \frac{1 - P}{Q}$$

and

$$0 \le P + Qb + (R + Sb)(w_E - w_B) + b(w_E - w_B)^2 \le 1 \tag{15}$$

$$b >= -\frac{P + (w_E - w_B)R}{Q + (w_E - w_B)S + (w_E - w_B)^2}$$

$$b <= \frac{1 - P - R(w_E - w_B)}{Q + S(w_E - w_B) + (w_E - w_B)^2}$$

These inequalities give a feasible region for the value of b that makes the w scale line intersect the unit square at the top and bottom edges of the unit square.

The turning point of the parabola must not be in the shaded region shown:

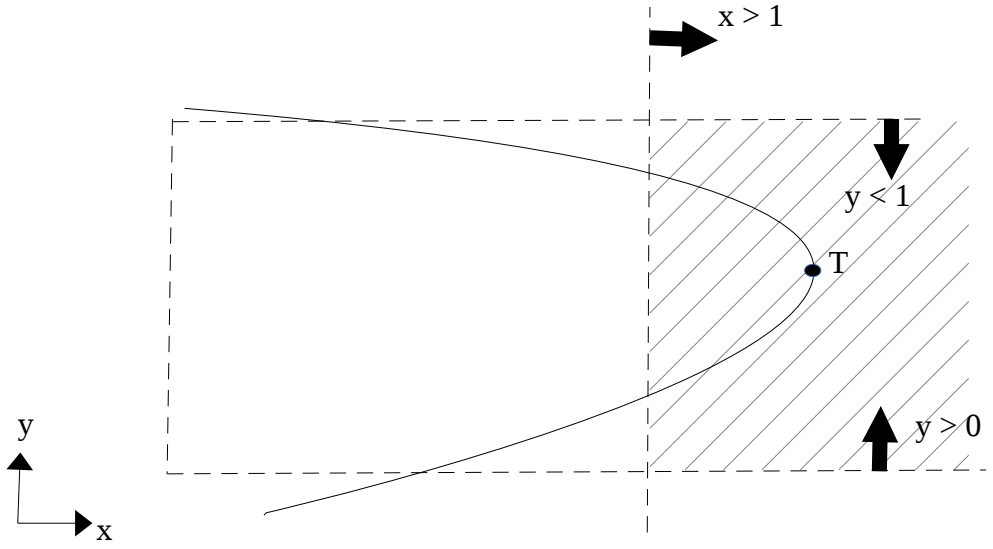


Diagram showing the w scale line with a vertex in a forbidden region

There is another forbidden region on the opposite side of the unit square.

The turning point, T, of the parabola occurs when

$$w'_x = \alpha_{wx} + 2b(w - w_B) = 0$$

giving

$$w_T = w_B - \frac{\alpha_{wx}}{2b}$$

From this, the x and y coordinates of T are:

$$x_T = c + \alpha_{wx}(w_T - w_B) + b(w_T - w_B)^2$$

$$y_T = \alpha_{wy}(w_T - w_B)$$

x_T and y_T can be expressed as functions of b:

… TBC …

The forbidden regions add more constraints on the forbidden values of b:

$$x_T > 1 \text{ and } 0 < y_T < 1 \quad \text{or} \quad x_T < 0 \text{ and } 0 < y_T < 1$$

Careful: combining **allowed** vs **forbidden** regions, **and** conditions vs **or** conditions

look at the extreme x pos of the w curve – it should be inside the unit square. This could add another constraint.

$$w_x' = \alpha_{wx} + 2b(w - w_B)$$

$$w_{xB}' = \alpha_{wx} \text{ at B, and } w_{xE}' = \alpha_{wx} + 2b(w_E - w_B) \text{ at E}$$

if these have opposite signs, then there is an extreme, (aka turning point), w_0, where

$\alpha_{wx} + 2b(w_0 - w_B) = 0$.

So may need to find some value of b in the allowed region st $0 <= w_x(w_0) <= 1$ or the derivatives have the same sign.

TBC

Case 2: Division by zero when $W_G = W_H$

Find quadratic and cubic approximations for the scale lines, using the derivative equations to find the required coefficients.

When $w_G = w_H$, we can assume an initial approximation as follows, and where a, b, c, d, e, f, g & s are parameters to be determined:

$$x_u(u) = a(u - u_{min})(u - u_{max})$$

$$y_u(u) = (u - u_{min})(b(u - u_{max}) + \frac{1}{u_{max} - u_{min}})$$

$$x_v(v) = c(v - v_{bottom})(v - v_{top}) + 1 \quad \text{newline}$$

$$y_v(v) = (v - v_{bottom})(d(v - v_{top}) + \frac{1}{v_{top} - v_{bottom}})$$

Note that the above equations make

- the u and v scales quadratic, and
- the scale lines go through their respective corners of the unit square

On the other hand, choose $x_w(w)$ to be a cubic with 3 parameters, e, f, & g:

$$x_w(w) = e(w - w_G)^3 + f(w - w_G)^2 + g(w - w_G) + 0.5$$

Note that this satisfies the requirement $x_w(w_G) = 0.5$

Since $y_w(w_{bottom}) = 0$, $y_w(w_{top}) = 1$ and $y_w(w_G) = 0.5$, $y_w(w)$ can be a cubic, which gives one degree of freedom, denoted by the parameter s:

$$y_w(w) = \alpha(w - w_{bottom})^3 + \beta(w - w_{bottom})^2 + s(w - w_{bottom}) \quad \text{, where}$$

$$\alpha = \frac{-\beta(w_{top} - w_{bottom})^2 - s(w_{top} - w_{bottom}) + 1}{(w_{top} - w_{bottom})^3} \quad \text{, and}$$

$$\beta = \frac{As + B}{(w_G - w_{top})(w_G - w_{bottom})^2(w_{top} - w_{bottom})^2} \quad \text{, where}$$

$$A = w_G^3 w_{bottom} - w_G^3 w_{top} + 3 w_G^2 w_{top} w_{bottom} - 3 w_G^2 w_{bottom}^2 + w_G w_{top}^3 - 3 w_G w_{top}^2 w_{bottom} + 2 w_G w_{bottom}^3 - w_{top}^3 w_{bottom} + 3 w_{top}^2 w_{bottom}^2 - 2 w_{top} w_{bottom}^3$$

$$B = w_G^3 - 3 w_G^2 w_{bottom} + 3 w_G w_{bottom}^2 - 0.5 w_{top}^3 + 1.5 w_{top}^2 w_{bottom} - 1.5 w_{top} w_{bottom}^2 - 0.5 w_{bottom}^3$$

Now use the derivative equations 4 & 5 on the 4 known index lines ABC, DEF, AGF & DGC to give 8 linear equations with the 8 unknowns a, b, c, d, e, f, g & s.

Cost Functions

If we have a formula

$$w = w(u, v), where\ u_{min} \le u \le u_{max}, v_{min} \le v \le v_{max}$$

we can determine the position of the u, v & w curves at the Chebyshev nodes, and from there generate the curves of the nomogram.

To get best accuracy, we want to maximise the area used by the nomogram, but we can't let it be larger than the unit square.

Use Area Between the Scale Lines

Using a line integral to determine the cost of a nomogram

See for example, ref[5]

The cost of a nomogram is given by the double integral

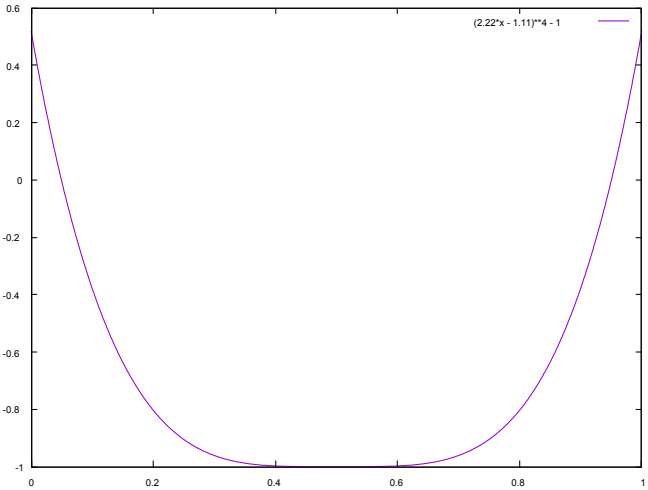
$$\iint_A c(x, y)\ \mathrm{dA} \quad ,$$

where c(x,y) is the cost function.

Consider this cost function:

$$c(x, y) = (20x/9 - 10/9)^4 + (20y/9 - 10/9)^4 - 1$$

It's value is about -1 in the central region of the unit square, but near the edges it quickly grows positive as it approaches the boundaries at 0 or 1:



If we define the cost of the nomogram as

$$cost = \iint_A c(x, y)\, dxdy$$

then the optimisation algorithm can minimise the cost by of the nomogram by using as much area as possible in the middle and up to the edges. Near the edges, the cost savings get smaller and the costs start to increase rapidly beyond the boundary of the unit square.

This can be turned into a line integral if we use Green's theorem with a vector function, **F** whose curl is c(x,y), i.e.

$$c(x,y) = \nabla \boldsymbol{x} \boldsymbol{F} = \frac{\partial F_2}{\partial x} - \frac{\partial F_1}{\partial y}$$

Such a function is:

$$\boldsymbol{F} = \left(y - y(20x/9 - 10/9)^4\right)\boldsymbol{i} + \left(x(20y/9 - 10/9)^4\right)\boldsymbol{j}$$

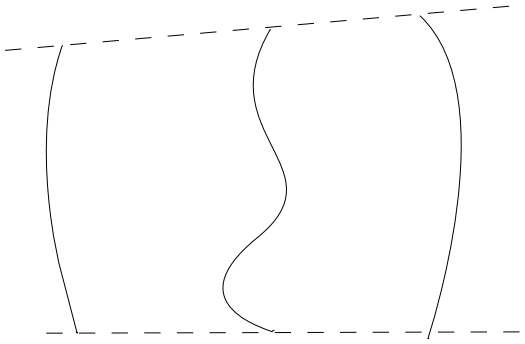
The line integral is

$$cost = \oint_C \left(y - y(20x/9 - 10/9)^4\right)dx + x(20y/9 - 10/9)^4\, dy$$

Along, say, the w scale line, we have $x_w = x_w(w)$, so $dx = x_w'(w)dw$ and similarly $dy = y_w'(w)dw$, and the cost function becomes

$$\oint_C \boldsymbol{F}.d\boldsymbol{s} = \int_a^b \boldsymbol{F}(\boldsymbol{c}(t)).\boldsymbol{c}'(t)dt$$

$$cost = \oint_C \left(y - y(20x/9 - 10/9)^4\right)x_w'(w) + \left(x(20y/9 - 10/9)^4\right)y_w'(w)dw$$



In the fictitious nomogram above, there are 3 areas –

1. s1 = the area between the 2 outer scales,
2. s2 = the area between the left and middle scales, and
3. s3 = the area between the right and middle scales

Ideally the nomogram cost should be least when the smaller 2 areas are roughly equal, i.e. $s2 \simeq s3$

If the cost is defined by combining the 3 areas as follows

$$\frac{1}{cost} = \frac{1}{s1} + \frac{1}{s2} + \frac{1}{s3}$$, or

$$cost = \frac{s1\ s2\ s3}{s1\ s2 + s2\ s3 + s3\ s1}$$

then the cost should me minimised when the smallest area is a large as possible.

Notes:

1. In the diagram above, s1 = s2 + s3. Could be slightly different if the min & max values of the middle scale do not lie on the dotted lines joining the 2 outer scales.
2. If the cost function is revised to
   $c(x,y) = (2x-1)^4 + (2y-1)^4 - 1$
   then the minimum cost of an area is -0.6, i.e. when an area exactly covers the unit square
3. By taking differentials, small changes in area affect the cost follows
   $$\frac{\Delta(cost)}{cost^2} = \frac{\Delta s1}{s1^2} + \frac{\Delta s2}{s2^2} + \frac{\Delta s3}{s3^2}$$

Cost function $\kappa(x^2 + y^2)$ can be implemented with vector field $\boldsymbol{F}(x,y) = (-\kappa x^2 y, \kappa x y^2)$

Boundary must be piecewise smooth, continuous & connected.

TBD
finding w' - see ref [4]
determine the circular path
detecting and handling crossovers

References

1. PyNomo documentation
2. http://www.myreckonings.com/pynomo/CreatingNomogramsWithPynomo.pdf
3. Allcock and Jones: The Theory and Practical Construction of Computation Charts
4. "Barycentric Lagrange Interpolation", Jean-Paul Berrut & Lloyd N. Trefethen, SIAM REVIEW Vol. 46, No. 3, pp. 501–517
5. https://mathinsight.org/greens_theorem_find_area