Goal

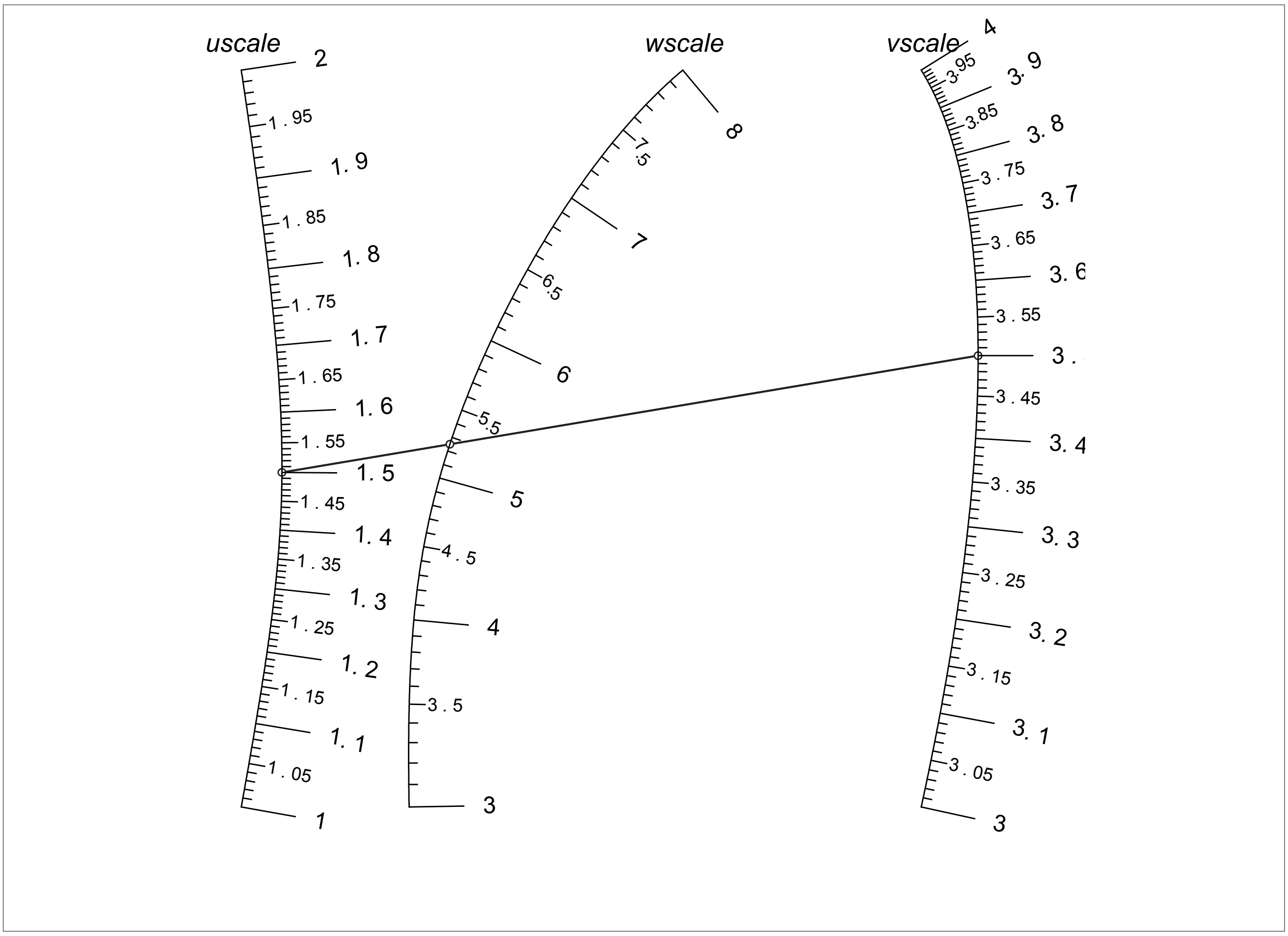
Generate a nomogram of up to 3 curved lines.

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.



Plan

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.)

Approximate the scale lines with polynomials, estimate an error using the distance of a point on its scale line from the corresponding index line, use SciPy numerical optimisation library to minimise this error.

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
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.

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 or w:

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

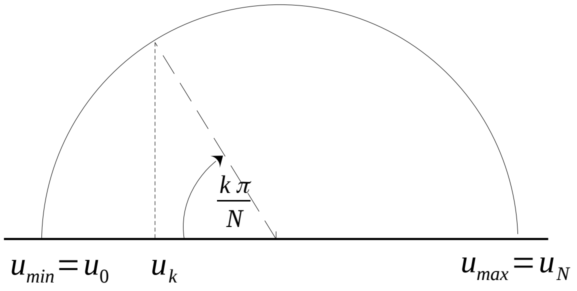
and so on for the v & w scale lines.

Each scale line is defined, not by 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.

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 line of equally spaced angles, so node u_k is the projection from an angle $k\frac{\pi}{N}$:



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

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

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

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

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.

- Approximate each scale line with a polynomial as described above.
- Frame the nomogram as a cost minimisation problem.
  The cost is a function of the error in the scale curves and how well the curves fit into the allocated area.
- use the SciPy library to minimise cost of nomogram.
  The cost is the square of the error and how well the nomogram fits the graph area. Cost and fit should be scaled to be of the same order of magnitude.
- Use PyNomo library to graph the nomogram

A major problem with using equally spaced nodes to approximate the scale lines 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.

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

(This code assumes N is even)

```
        if w is one of $w_k$ then
        use $x_k$ & $y_k$, the known values for node k
    else
        sumA = ($x_0$/(w-$w_0$) + $x_N$/(w-$w_N$))/2;
        sumB = (1/(w-$w_0$) + 1/(w-$w_N$))/2;
        for k in 0 .. N
            t = 1/(w-$w_k$);
            sumA = t*$x_k$ - sumA;
            sumB = t - sumB;
    x(w) = sumA / sumB
```

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

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

Now, 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

- 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

Initial Estimate

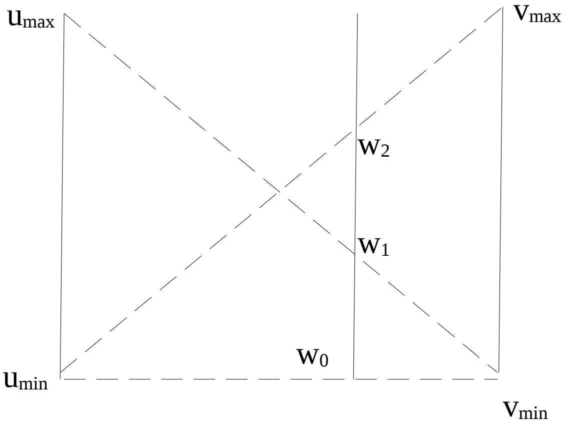
If we assume that the u scale increases (varies from min to max) as the y coordinate increases, and if we define

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

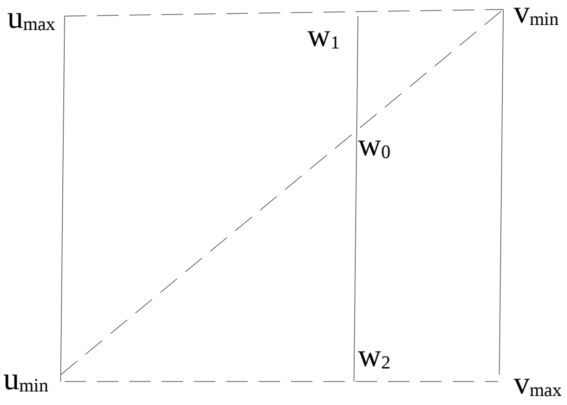
$$w_1 = w(u_{max}, v_{min})$$

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

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



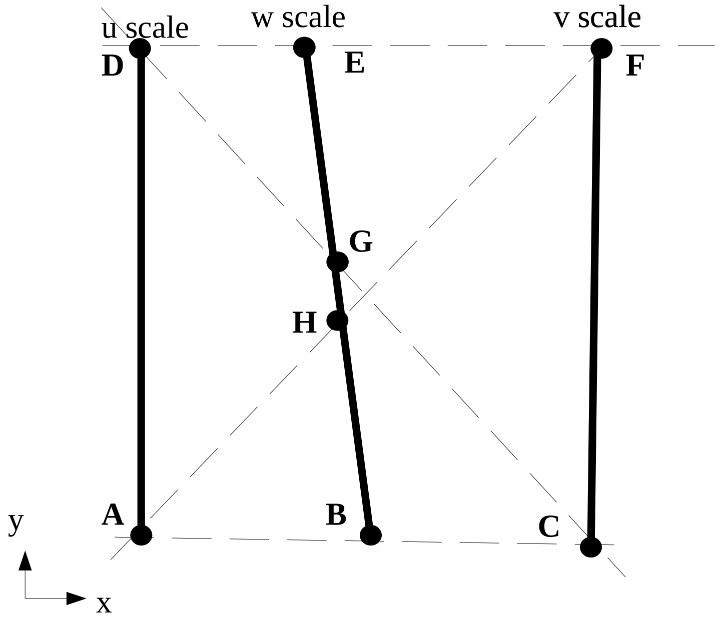
Here, the v scale increases upwards and both w_1 and w_2 must be greater than w_0 if the w scale increases upward, or they both must be less than w_0 if the w scale increases downwards.



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

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

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



The u, v & w scale lines are respectively **AB**, **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

- $v_y = \frac{v - v_{min}}{v_{max} - v_{min}}$

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

On the other hand, if the v scale is reversed so that v_{min} is at the top, then:

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

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

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

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

$$c+\alpha_{wx}(w_G-w_B)\ =\ 1-\alpha_{wy}(w_G-w_B)\ = 1\ -\ \frac{w_G-w_B}{w_E-w_B}\ =\ \frac{w_E-w_G}{w_E-w_B} \tag{3}$$

Equation (1) gives α_{wy} and (2) & (3) can be solved to give:

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

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

One problem that might occur is that this approximation can leave w_B & w_E outside the unit square. This can be corrected by adding a quadratic term to the formula for w_x:

$$w_x=c+\alpha_{wx}(w-w_B)+b(w-w_B)^2 \tag{4}$$

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

Equations (2) & (3) above are modified to:

$$c+\alpha_{wx}(w_H-w_B)\ +\ b(w_H-w_B)^2\ =\ \alpha_{wy}(w_H-w_B)\ =\ \frac{w_H-w_B}{w_E-w_B} \tag{5}$$

$$c+\alpha_{wx}(w_G-w_B)\ +\ b(w_G-w_B)^2\ =\ 1-\alpha_{wy}(w_G-w_B)\ = 1\ -\ \frac{w_G-w_B}{w_E-w_B} \tag{6}$$

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

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

$$\alpha_{wx} = \frac{w_E - w_G - w_H + w_B}{(w_G - w_H)(w_E - w_B)} - b(w_G + w_H - 2w_B) \qquad (7)$$

$$c = \frac{w_H - w_B}{w_E - w_B} - \alpha_{wx}(w_H - w_B) - b(w_H - w_B)^2 \qquad (8)$$

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

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

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

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

$$0 \le P + Qb + (R + Sb)(w_E - w_B) + b(w_E - w_B)^2 \le 1 \qquad (9)$$

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

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

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:

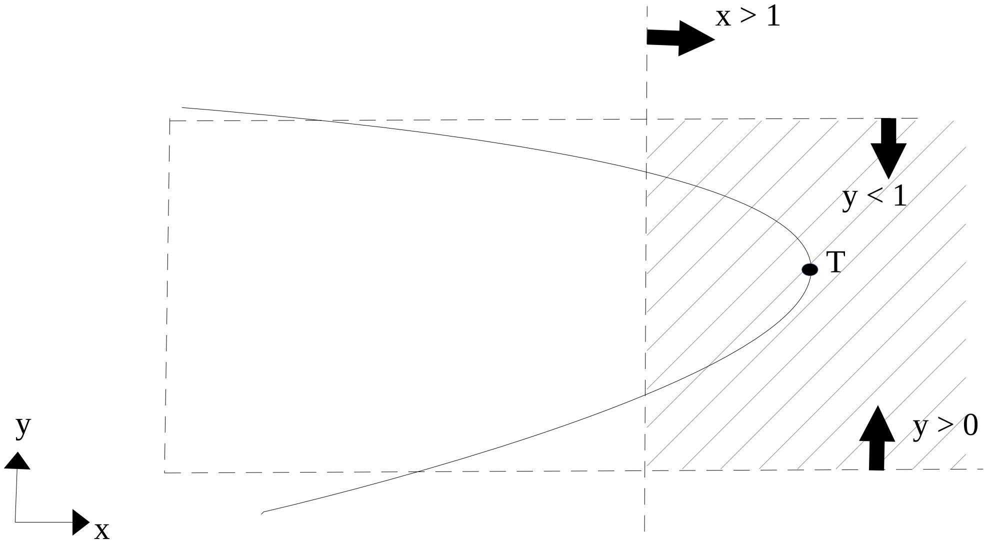


Diagram showing the w scale line with a vertex in a forbidden region

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.

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

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

The following is unused and not even checked for correctness ….

The points on each index line are co-linear , so we have

for index line **DEF**:

$$(D_x - E_x)(D_y - F_y) = (D_x - F_x)(D_y - E_y)$$

$$(0 - c) * (1 - \alpha_{vy}(v_{max} - v_{min})) = (0 - 1) * (1 - \alpha_{wy}(w_E - w_B))$$

$$c(1 - \alpha_{vy}(v_{max} - v_{min})) = 1 - \alpha_{wy}(w_E - w_B)$$

for index line **AHF**:

$$(A_x - H_x)(A_y - F_y) = (A_x - F_x)(A_y - H_y)$$

$$(0 - c) * (0 - \alpha_{vy}(v_{max} - v_{min})) = (0 - 1) * (0 - \alpha_{wy}(w_H - w_B))$$

$$c\,\alpha_{vy}(v_{max} - v_{min}) = \alpha_{wy}(w_H - w_B)$$

for index line **DGC**:

$$(D_x - G_x)(D_y - C_y) = (D_x - C_x)(D_y - G_y)$$

$$(0 - c) * (1 - 0) = (0 - 1) * (1 - \alpha_{wy}(w_G - w_B))$$

$$c = 1 - \alpha_{wy}(w_G - w_B)$$

This gives 3 equations for the 3 unknowns c, α_{vy} & α_{wy} with the solution:

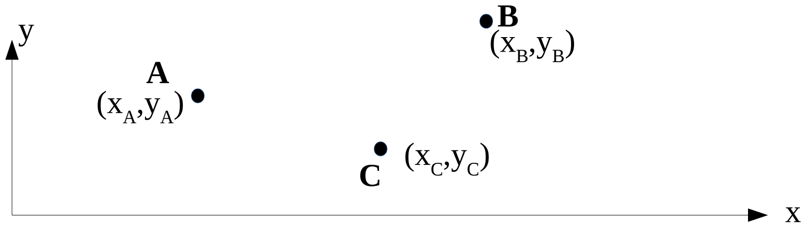
$$\alpha_{wy} = \frac{1}{w_E - w_B}$$

$$\alpha_{wx} = \frac{1 - \alpha_{wy}(w_G - 2w_B + w_H)}{w_G - w_H}$$

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

Notes

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

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

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

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.

- Assume each line can be approximated by a Chebyshev series, parametric curve,

$$u_x = \sum_{k=0}^{N} \alpha_i T_i(u)$$

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

$$u_k = x_{min} + (1 + \cos(\frac{k*\pi}{N}))*(u_{max} - u_{min})/2$$

Using a line integral to determine the cost of a nomogram

See for example, ref[3]

The cost of a nomogram is given by the double integral

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

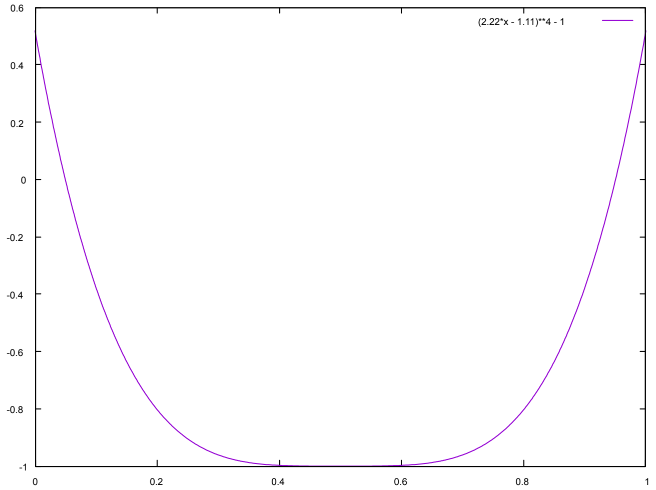
where c(x,y) is the cost function.

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.

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), ie

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

Such a function is:

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

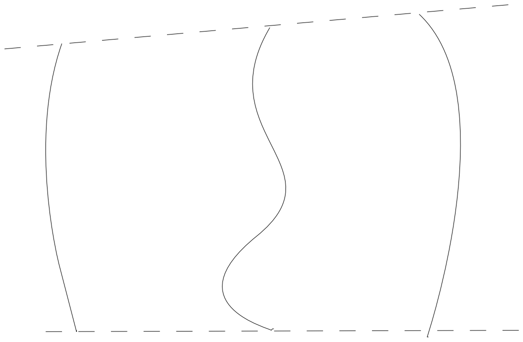
The line integral is

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

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, ie $s2 \simeq s3$
If the cost is defined by combining the 3 areas as follows

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

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

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, ie 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 [2]
determine the circular path
detecting and handling crossovers

References

1. Allcock and Jones: The Theory and Practical Construction of Computation Charts
2. “Barycentric Lagrange Interpolation”, Jean-Paul Berrut & Lloyd N. Trefethen, SIAM REVIEW Vol. 46, No. 3, pp. 501–517
3. https://mathinsight.org/greens_theorem_find_area