Numerical methods

Dominic Skinner

June 28, 2016

Consider the govering equations

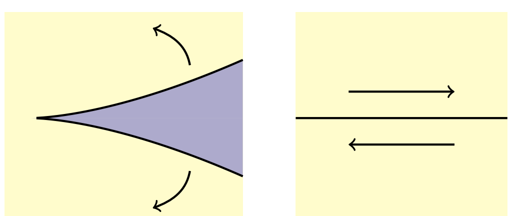
$$\begin{pmatrix} p(z) \\ 0 \end{pmatrix} = \begin{pmatrix} \sigma_y \\ \tau_{xy} \end{pmatrix} = \int_0^\infty \begin{pmatrix} K_{11}(x-z) & K_{12}(x-z) \\ K_{21}(x-z) & K_{22}(x-z) \end{pmatrix} \begin{pmatrix} g'(x) \\ h'(x) \end{pmatrix} dx \quad (1)$$

$$h^2 p' = \lambda \quad (2)$$

Have the "input" parameters as

- BC's P, M (or equivalently g', h'' at $x \to \infty$)
- λ , the speed

Want to solve for the toughness K_I and K_{II}. In this project, we have so far focused on K_I.



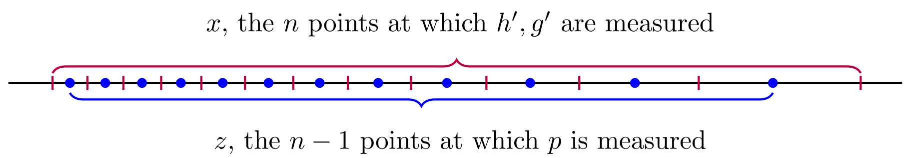
K_I, Bending moment K_{II}, Shear

<u>Goal:</u> Find λ such that $K_I(\lambda) = 0$, "Zero toughness solution". Given this we then want to investigate the behaviour for small $K_I \approx 0$. To do this, take some given value of λ and then solve equations 1, 2.

Discretization of problem

The method chosen to discretize the problem is to take a vector $(x_1, \ldots, x_n)$ of n points at which we measure g', h' and have a vector $(z_1, \ldots, z_{n-1})$ of $n-1$ intermediate points at which p is measured. (The spacing chosen is a $\tan^2$ spacing)

The "obvious" way to interpolate h' in between the x_i's is simple linear interpolation. But both h', g' become singular near 0. However, we expect a $x^{-1/2}$



singularity, which allows us to "remove" said singularity. The interpolation used is

$$g'(x) = \begin{cases} \frac{1}{\sqrt{x}}(a_i x + b_i) & i < t \\ a_i x + b_i & i \geq t \end{cases}$$

for x in the spline $x \in [x_i, x_{i+1}]$. Choose $1 < t < n$, typically $t = n/2$. Similarly

$$h'(x) = \begin{cases} \frac{1}{\sqrt{x}}(c_i x + d_i) & i < t \\ c_i x + d_i & i \geq t \end{cases}$$

With the same t used. We also define a_n, b_n, c_n, d_n for interpolation beyond x_n.

The values of g', h' are stored via

$$\boldsymbol{\theta} = \begin{pmatrix} a_1 x_1 + b_1 \\ \vdots \\ a_n x_n + b_n \\ c_1 x_1 + d_1 \\ \vdots \\ c_n x_n + d_n \end{pmatrix}$$

Once one has $\boldsymbol{\theta}$, it is trivial to recover, say $g'(x_i)$, since either $g'(x_i) = \boldsymbol{\theta}_i$ or $g'(x_i) = \boldsymbol{\theta}_i/\sqrt{x_i}$. Similarly, given $g'(x_i)$; $\boldsymbol{\theta}_i$ can be calculated.

Recovering the a$_i$'s

Suppose we know $\boldsymbol{\theta}$, (and always assume we know the x_i). Can we recover a_i, b_i, c_i, d_i? The answer is yes, once we add in the boundary conditions at ∞.

Further we have that

$$\boldsymbol{\gamma} = \begin{pmatrix} a_1 \\ \vdots \\ a_n \\ b_1 \\ \vdots \\ b_n \\ c_1 \\ \vdots \\ c_n \\ d_1 \\ \vdots \\ d_n \end{pmatrix} = T\boldsymbol{\theta}$$

Where T is a $4n \times n$ interpolation matrix. A quick check reveals we have $4n$ unknowns, in $\boldsymbol{\gamma}$. Knowing θ provides $2n$ equations. Demanding continuity of the interpolated g', h' provides another $2(n-1)$ equations, (match at $x_2, \ldots, x_n$). Finally boundary conditions on the spline at ∞ provide another 2 equations.

The continuity conditions are

$$\begin{aligned} a_1 x_2 + b_1 &= a_2 x_2 + b_2 \\ &\vdots \\ a_{t-2} x_{t-1} + b_{t-2} &= a_{t-1} x_{t-1} + b_{t-1} \\ (a_{t-1} x_t + b_{t-1})/\sqrt{x_t} &= a_t x_t + b_t \\ a_t x_{t+1} + b_t &= a_{t+1} x_{t+1} + b_{t+1} \\ &\vdots \\ a_{n-1} x_n + b_{n-1} &= a_n x_n + b_n \end{aligned}$$

and similar for c, d. This means that with the exceptions of $i = t-1, n$, have that

$$\frac{\theta_{i+1} - \theta_i}{x_{i+1} - x_i} = a_i$$

$$\frac{\theta_i\, x_{i+1} - \theta_{i+1} x_i}{x_{i+1} - x_i} = b_i$$

Same idea with x_{t-1}, just have to be a little careful about the switch in the continuity condition,

$$\frac{\sqrt{x_t}\theta_t - \theta_{t-1}}{x_t - x_{t-1}} = a_{t-1}$$

$$\frac{\theta_{t-1}\, x_t - \theta_t x_{t-1}\sqrt{x_t}}{x_t - x_{t-1}} = b_{t-1}$$

So we are almost done, just missing 4 rows in our matrix. Have the n^{th} row as all zeros, i.e. $a_n = 0$ due to boundary conditions, and so trivially the $2n^{th}$ row is 0 except $T_{2n,n} = 1$ Now, B.C. for h' implies $h''(x_n) \approx h''(x_{n-1})$ i.e. $c_n = c_{n-1}$ and thus from continuity $d_n = d_{n-1}$. This completes our interpolation matrix T. (and we still have some extra boundary conditions to impose, which we will do later.)

The discrete version of $\begin{pmatrix} p \\ 0 \end{pmatrix} = \int \begin{pmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{pmatrix} \begin{pmatrix} g' \\ h' \end{pmatrix}$ becomes

$$\begin{pmatrix} p(z_1) \\ \vdots \\ p(z_{n-1}) \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} B_{1,1} & \cdots & B_{1,2n} \\ \vdots & \ddots & \vdots \\ B_{2(n-1),1} & \cdots & B_{2(n-1),2n} \end{pmatrix} \begin{pmatrix} g'(x_1) \\ \vdots \\ g'(x_n) \\ h'(x_1) \\ \vdots \\ h'(x_n) \end{pmatrix}$$

Where the matrix B depends on the choice of spacings for x, z but does not depend on the values that g', h' take. We can go further and incorporate the boundary conditions into this equation. The discretized versions of the boundary conditions, become[1] $g'(x_n) = 1/2$ and $\frac{h'(x_n)-h'(x_{n-1})}{x_n - x_{n-1}} = 1$. These conditions are linear in terms of g', h', and so by adding another two rows onto the matrix B, get that

$$\begin{pmatrix} p(z_1) \\ \vdots \\ p(z_{n-1}) \\ 0 \\ \vdots \\ 0 \\ g'(\infty) \\ h''(\infty) \end{pmatrix} = \begin{pmatrix} A_{1,1} & \cdots & A_{1,2n} \\ \vdots & \ddots & \vdots \\ A_{2n,1} & \cdots & A_{2n,2n} \end{pmatrix} \begin{pmatrix} g'(x_1) \\ \vdots \\ g'(x_n) \\ h'(x_1) \\ \vdots \\ h'(x_n) \end{pmatrix}$$

Where $g'(\infty), h''(\infty)$ are some constants that are the boundary conditions.

Now we use the second equation for p, namely $p = \int_z^\infty \lambda/h^2 dx$. This depends on h' in a very much non linear way. It will however provide an expression for the $p(z_i)$ in terms of the $h'(x_j)$. Thus, switching to the notation $\boldsymbol{h}' = (g',\, h')$ where the first n coords of $\boldsymbol{h}'$ are the coords of g' and the second n are the

[1] Or something similar depending on the exact scalings

coords of h'. We have that

$$f(\boldsymbol{h}') = \begin{pmatrix} p(z_1) \\ \vdots \\ p(z_{n-1}) \\ 0 \\ \vdots \\ 0 \\ g'(\infty) \\ h''(\infty) \end{pmatrix} = A\boldsymbol{h}'$$

Where we now just need to solve for $\boldsymbol{h}'$.

Newton's method

Suppose $\boldsymbol{h}'$ is iterate 1. To get the next iterate you need to solve (to first order)

$$f(\boldsymbol{h}' + \delta\boldsymbol{h}') = A(\boldsymbol{h}' + \delta\boldsymbol{h}')$$

$$f(\boldsymbol{h}') + (Df|_{\boldsymbol{h}'})(\delta\boldsymbol{h}') = A\boldsymbol{h}' + A\delta\boldsymbol{h}'$$

Where $Df|_{\boldsymbol{h}'}$ is a matrix of partial derivatives. Therefore, get to first order that

$$\delta\boldsymbol{h}' = (A - Df|_{\boldsymbol{h}'})^{-1}(f(\boldsymbol{h}') - A\boldsymbol{h}')$$

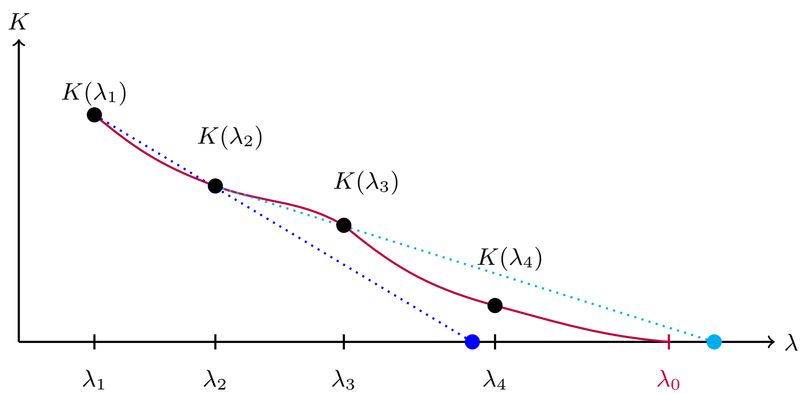
Ingredients:

- Matrix A itself (of which the $2(n-1) \times 2n$ part is the integral kernel)
- The function $f(\boldsymbol{h}')$. I.e. given $\boldsymbol{h}'$ you need to calculate $\int_z^\infty \lambda/h^2 dx$ (Key functions "hprime_to_h" and "hprime_to_p").
- Need to calculate Df which involves calculating $\dfrac{\partial}{\partial \boldsymbol{h}'} \displaystyle\int_z^\infty \frac{\lambda}{h(x)^2} dx$

So we have worked out numerically $K(\lambda)$, now we want to solve $K(\lambda_0) = 0$ for λ_0. We do a "march". Sublety in that $K < 0$ is unphysical, so a guess of $\lambda > \lambda_0$ where $K(\lambda_0) = 0$ does not make any physical sense (& will get bad numerical results). To get around this difficulty, take the next iterate of λ as smaller than predicted.

For example in Figure 1, the obvious choice for λ_4 (the light blue circle) is larger than the true value of λ_0, and therefore the naive extrapolation method won't quite work.

Figure 1: March to find λ_0



Guide to programs

K_of_c_march

First the program sets up the spacing as $\tan^2$. It also sets the initial $\boldsymbol{h}' = (\underbrace{1,\dots,1}_{g'}\underbrace{x_1+1,\dots,x_n+1}_{h'})$ (Not sure why this is a reasonable first guess, perhaps from the boundary conditions at ∞. Seems to have no problems converging though.)

Most of the work is then done by `fixed_lambda_M_iteration` which then solves for K_I and $\boldsymbol{h}'$.

N.B. $\boldsymbol{h}'$ is updated via $\boldsymbol{h}'_i = \dfrac{\boldsymbol{h}'_{i-1} - \boldsymbol{h}'_{i-2}}{\lambda_{i-1} - \lambda_{i-2}}\lambda_i + \dfrac{\lambda_{i-1}\boldsymbol{h}'_{i-2} - \lambda_{i-1}\boldsymbol{h}'_{i-1}}{\lambda_{i-1} - \lambda_{i-2}}$ which is just linear extrapolation. In the absence of any better ideas this is the sensible choice.

After iterating for a few values, get near λ_0. Hear we suspect that something like $K^3 \sim \lambda - \lambda_0$ near $\lambda = \lambda_0$, $K = 0$. So given two prior guesses, extrapolate via $\lambda_i = \dfrac{K^3_{i-1}\lambda_{i-2} - K^3_{i-2}\lambda_{i-1}}{K^3_{i-1} - K^3_{i-2}}$ But as noted earlier, must be careful to not extrapolate further than λ_0. So an idea is to take $(\lambda_i + \lambda_{i-1})/2$ as the next guess, i.e.

$$\lambda_i = \frac{\lambda_{i-1} - \lambda_{i-2}}{K^3_{i-1} - K^3_{i-2}}\frac{K^3_{i-1}}{2} + \frac{K^3_{i-1}\lambda_{i-2} - K^3_{i-2}\lambda_{i-1}}{K^3_{i-1} - K^3_{i-2}}$$

Then the program just iterates. If it doesn't converge, it simply tries a smaller value of λ.

fixed_lambda_M_iteration

Arguably the most important function. Takes a value of λ and returns the corresponding K value.

Hard coded into the program are the values of P and M set as

$$\begin{cases} M = 1 \\ P = 0 \end{cases}$$

Sets up spacing for x. $\tan^2$ spacing is used.

Somewhat concerningly, $\boldsymbol{h}'$ is assumed to already have this spacing, which could potentially cause issues. If you wanted to change the spacing you would have to do it in two different places.

Also a cause for concern, or note is that with this $\tan^2$ spacing is that the maximum value of x_{max} is not actually x_{max} but rather x^2_{max}. I.e. using $x_{max} = 20$ actually results in the maximum value of x used being 400.

Subroutines then return the kernel matrix & the interpolate matrix. The kernel matrix is in lieu of $\begin{pmatrix} p \\ 0 \end{pmatrix} = \int \underline{\underline{K}} \begin{pmatrix} g' \\ h' \end{pmatrix}$. I am not sure of what the interpolate matrix actually is, or what it's for. Finding this out is a big priority.

The matrix A is set up, which is part kernel, part interpolate matrix, same matrix as described earlier.

The rcond statement is testing how conditioned the matrix A is, or how ameniable it is to being numerically inverted.

Then the iteration loop begins. Follows Newton's method for the equation $f(\boldsymbol{h}') = A\boldsymbol{h}'$ and iterates via $\boldsymbol{h}'_{new} = \boldsymbol{h}'_{old} + (A - Df|_{\boldsymbol{h}'_{old}})^{-1}(f(\boldsymbol{h}'_{old}) - A\boldsymbol{h}'_{old})$ Where we already know A. f, Df are provided via `hprime_to_p` and f, Df are called p, dp respectively in the program.