Progress Report Shooting for Bending Equation

Benjamin Schreyer
benontheplanet@gmail.com

1 Introduction

1.1 The bending equation

Consider a sheet-like flexure (one dimension of the flexure is much smaller than the other) suspending a weight. The weight experiences gravitational force $F_{w,0} = mg$ and the bending angle θ and moment M a distance s along the flexure depend on boundary conditions and material parameter E. The geometry of bending has the following governing equations.

$$\frac{dM}{ds} = F_{w,0}\sin(\theta(s)) \tag{1}$$

$$\frac{d\theta}{ds} = \frac{M(s)}{EI(s)} \tag{2}$$

The geometry of the unbent material is encoded in $I(s)$ the second moment area of the cross section. For the case we study:

$$I(s) = \frac{h(s)^3 b}{12}, \tag{3}$$

where $h(s)$ is the shape of the bending flexure and b is the thickness of the thin sheet flexure.

Bending can be solved using library solvers and one sided boundary conditions. Then to impose a boundary condition on the other end of the flexure, $s = L$, the shooting method can be used. This approach is effective but fails in extremely flexible cases due to magnitude limits of floating point representation.

1.2 Floating point representation

There are standards for floating point representation, for our purposes is it only important to know that a floating point number is represented roughly by two integers as $N = m \cdot 2^l$.

The base of two is unimportant but is chosen for convenience with digital devices. To make a classic floating point number, assign m and l a certain number of bits in representation. This makes hardware much more efficient due to standardization and the containment of m and l to say a single 64 bit word as is the case with float64 types. This inherently imposes a smallest and largest number possibly represented which is ok for most uses. Here for extreme bending we need to use a library mpmath for python which implements floating point calculations with m and l represented with as much memory as needed. This means that while the smallest number I can represent with float64 is something like $2^{-2^{10}} = 10^{-308}$. If we take seriously the arbitrary precision of mpmath and are using a computer with 1MB of memory for our number the smallest number representable is $2^{-(2^{1000000})}$. For representing physical quantities this large exponent is not needed, one can introduce a larger or smaller unit. For the calculation of bending however, the limitations of float64's exponent prevent calculations that appear in practical cases.

2 Analysis and solution

2.1 Extreme exponents in bending

The problems arising with exponent limitations in bending calculation are confusing because nothing physically relevant about the bending problem involves an exponent of say 10^{-308} or 10^{308}. If something is so small it goes unmeasured and extreme quantities would not be of interest.

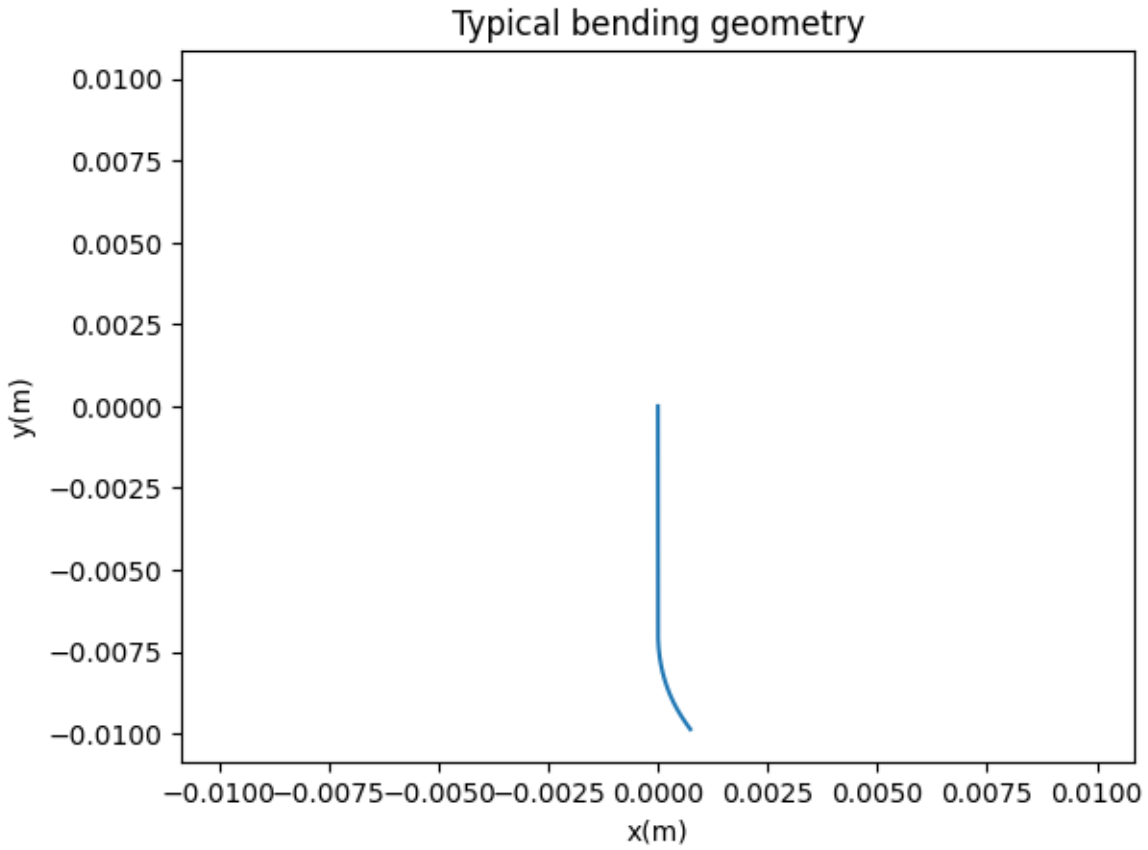

Figure 1: A solution to the bending problem taking the path $\theta(s)$ into Cartesian coordinates. The flexure is mounted at the origin.

When on tries to solve for the bending of a material in our application, the boundary conditions $M(0)$ and $\theta(0)$ are set. Then the shooting method is used on the final angle to reach a solution with $\theta(L) = \theta^\star$. When applied this algorithm works, except for when the geometry of the flexure arm becomes extreme. An extreme geometry is characterized by the flexure becoming extremely thin as is illustrated in the following figure.

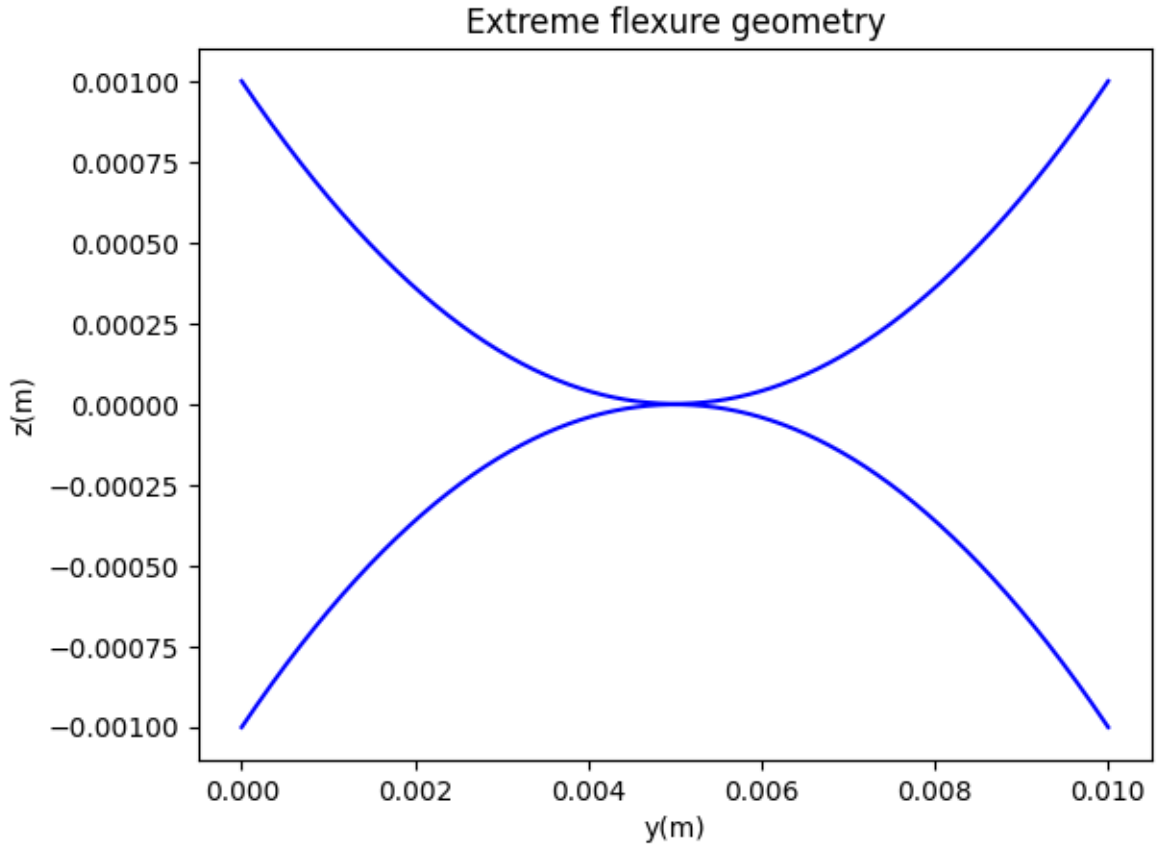

Figure 2: A geometry ($\frac{1}{2}h(s)$) for the flexure which is extremely small at its center. It shrinks from its base to be $10,000$ times thinner at its center and the sheet thickness $b = 0.1$mm

According to the equation for the second moment (3) and the variation of θ (2) when the geometry gets extremely thin the rate of change of θ will become extremely large. The result is that to get reasonable resulting $M(L)$ and $\theta(L)$ in a shooting approach the initial conditions must be extremely small. Then the final value for θ does not get evolve beyond π which is where oscillations in θ and M begin. For example a calculation can be made that gives $\theta(L) = 1$ for the extreme geometry pictured. To acheive this numerically, float64 or float128 will not allow a convenient solution. Convenient means without a headache of rescaling and dealing with the inherent scale of 2π from $\sin(\theta)$. Instead the mpmath library for python can be used when the geometry is extreme for implementing the shooting method applied to the bending equation.

2.2 Analytical guess for initial conditions when shooting

To illustrate beyond a qualitative understanding that there is an exponential increase present in bending, consider the region where θ is small which is true for most of the bending.

Additionally consider a straight geometry $h(s) = C$. Then the equations for bending simplify to

$$\frac{d^2\theta}{ds^2} = \frac{F_{w,0}}{EC}\theta. \tag{4}$$

The solution is trivially exponential. The scaling of θ from the initial $\theta(0)$ to $\theta(l)$ is easily quantified.

$$\theta(l) = \exp(\sqrt{\frac{F_{w,0}}{EC}}l)\theta(0) \tag{5}$$

Now to approximate the total scaling over the extreme geometry pictured above, approximate the geometry as piecewise constant. The scale factor will be

$$\exp(\sqrt{\frac{F_{w,0}}{E}}\int_0^L \frac{1}{\sqrt{I(s)}}ds). \tag{6}$$

Calculating this scale factor for the geometry above and $E = 7 \times 10^{10}$Pa and $F = 120$N the scale factor is 10^{19000}. In simulation the correct initial condition chosen to produce the bending plots given was order 10^{-1000} (to cancel the exponential growth). The single order scale error approximating the exponent was true in two cases. Clearly this is an approximation, the exponent predicted is one order larger than was required. This easy to calculate estimate provides a good starting guess when shooting for initial conditions.

2.3 Conclusion

In light of dealing with already unusual computing with large exponents, we implement shooting with binary search approach requiring no numerical differentiation. First the exponent is roughly determined such that $\theta(L)$ is less than one. Then a second search in linear space can be done to get the exact final bending angle. Implemented purely in python one bending calculation can be made per second. Implementing in a compiled language or using mpmath's built in ODE solver a bending calculation could be made roughly every 100ms with arbitrary geometry. Some example bending simulations are included for increasingly thin waisted geometries.

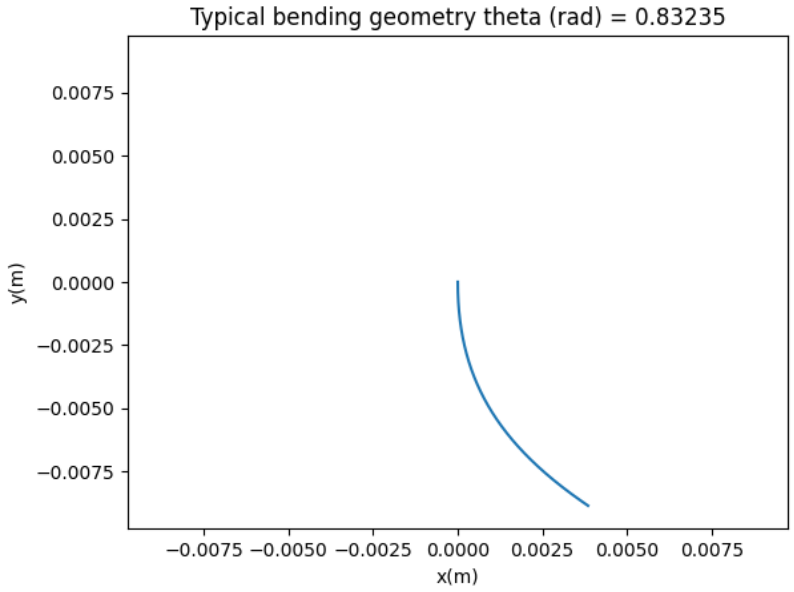

Figure 3: Bending of a nearly rectangular flexure.

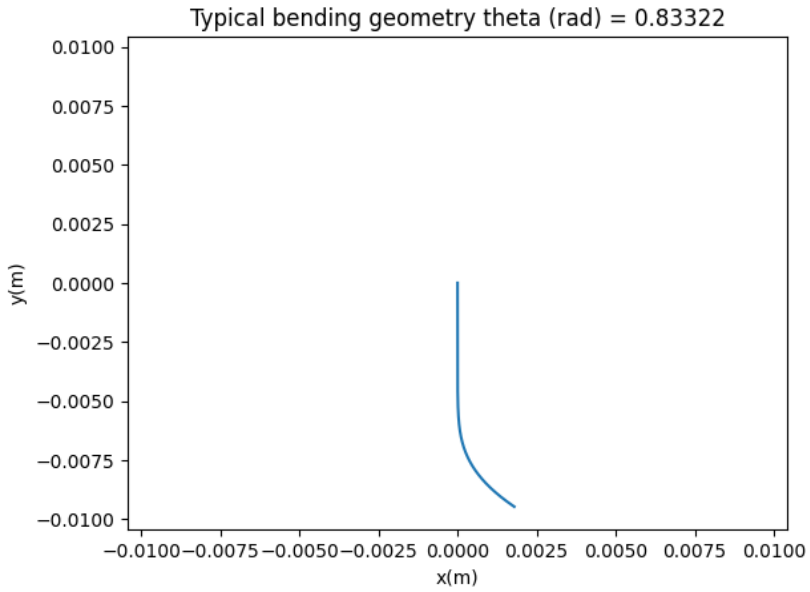

Figure 4: Bending of flexure that reaches a minimum half of its initial thickness.

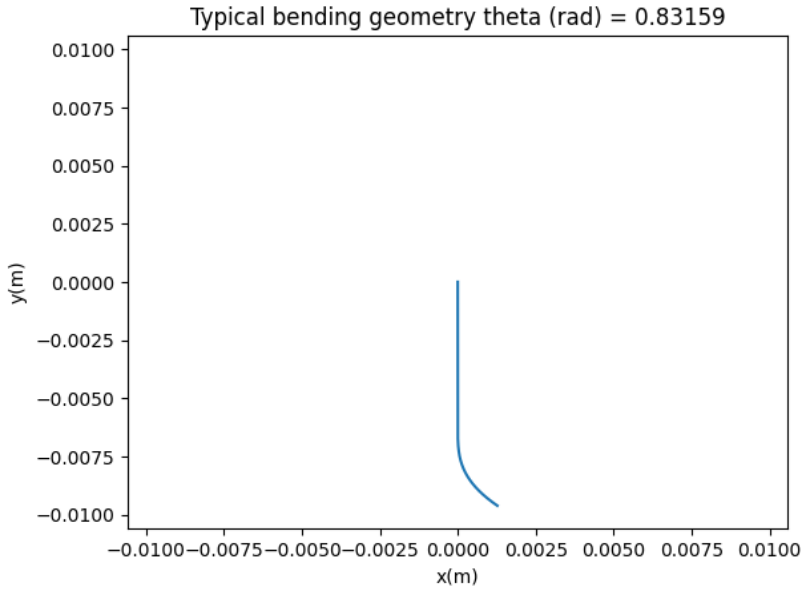

Figure 5: Bending of flexure that reaches a tenth of its initial thickness.

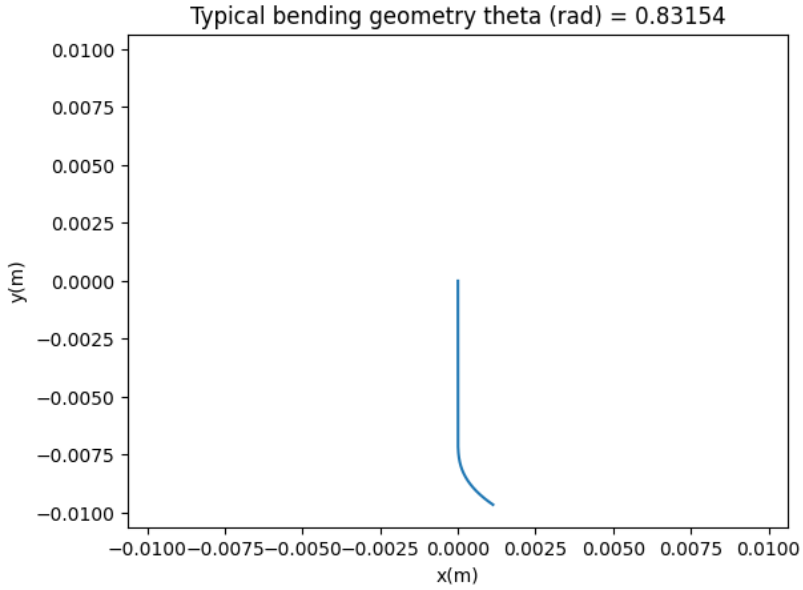

Figure 6: Bending of flexure that reaches a percent of its initial thickness.