Math 126 – 12W
March 9, 2012

Adaptive Collocation for Boundary Integrals in Two Dimensions using Smooth Reparameterization

Vipul Kakkad

Math 126: Numerical Analysis for PDEs and Waves

Boundary Integral Equations

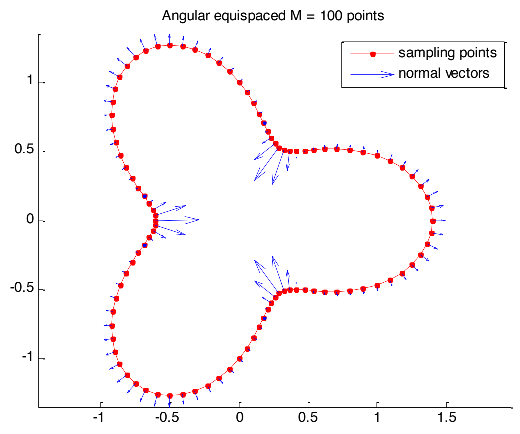
The boundary integral method is a fairly ubiquitous method of solving certain PDEs, whose solutions can be expressed in the form of linear combinations of fundamental solutions centered at different points. Notable examples include solving Boundary Value Problems for the Laplace equation, $\Delta^2 u = 0$, and also the Helmholtz equation, $(\Delta^2 + k^2)u = 0$, on domains in $\mathbb{R}^n$ usually for n = 2 or 3 (we will focus on the $n = 2$ case here).

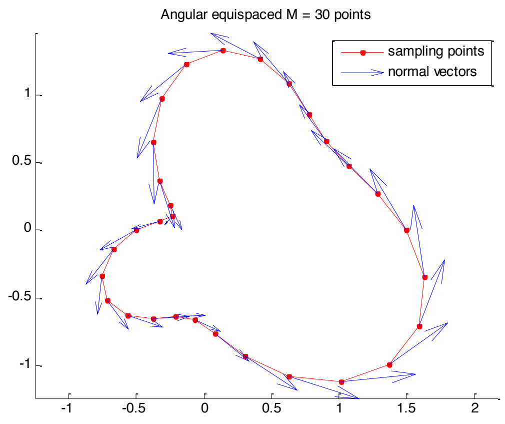
The boundary is usually parameterized by a function $\boldsymbol{z} : [0,2\pi] \rightarrow \partial\Omega$, that traverses the boundary once, and the integral

$$I = \int_{\partial\Omega} f(\boldsymbol{x})\, d\boldsymbol{x}$$

where f is the function that we are trying to integrate, and $\boldsymbol{x} \in \mathbb{R}^2$. And $\boldsymbol{z}$ is a continuous function that has a positive (and non-zero) 'speed' $|\boldsymbol{z}'(s)|$ at every point $s \in [0,2\pi]$, and $\boldsymbol{z}(0) = \boldsymbol{z}(2\pi)$ (requiring that $\boldsymbol{z}$ is 2π periodic, and smooth)

These equations involve numerically solving an integral equation created using Green's representation functions, along the boundary $\partial\Omega$ of the domain $\Omega \subset \mathbb{R}$, using the Nystrom method, that uses discretized collocation of $\partial\Omega$.





F1: Above, we have quadrature points for different two different shapes, with normal vectors scaled by curvature on the left, and unit tangent vectors on the right.

As reviewed in class, it is provable that the most efficient quadrature scheme for such an integral is the 'Periodic Quadrature' scheme, that uses evenly spaced points that are given by $\boldsymbol{z}(\frac{2\pi j}{N})$, where N is the number of nodes being used in the scheme, and $j \in [1,2,3, \dots N]$, all equally weighted by $\frac{2\pi}{N}$.

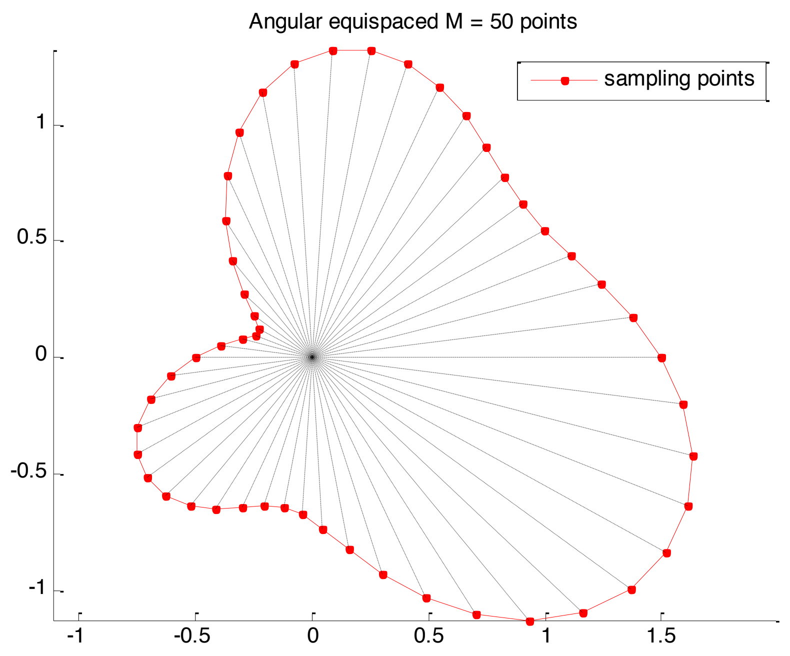
After using the periodic quadrature rule, our integral is now:

$$I \;=\; \int_0^{2\pi} f(\boldsymbol{z}(s))\,|\boldsymbol{z}'(s)|ds \;\approx\; \frac{2\pi}{N}\sum_{j=1}^{N} f\big(\boldsymbol{z}(s_j)\big)\,\big|\boldsymbol{z}'(s_j)\big|$$

where $s_j = \frac{2\pi j}{N}$.

Re-parameterizing the Boundary Curve

While the continuous integral may be independent of how we choose to parameterize the curve representing the boundary (our choice of the function $\boldsymbol{z}$), the approximation given by the periodic quadrature is clearly highly dependent on the collocation of points we use. The speed at which this parameterization travels through the boundary curve determines where the images $\boldsymbol{z}(\frac{2\pi j}{N})$ will lie. This is the degree of freedom (or rather infinitely many degrees of freedom) that we intend to exploit to give us a better approximation to the correct boundary integral than using the naïve approach of using periodic quadrature on the existing parameterization.



F2: Observe the clumping of data points in areas of the curve close to the 'origin' (which is arbitrary for a given parameterization of a curve). These nodes are the 'evenly spaced' periodic quadrature nodes for a radial parameterization of this curve

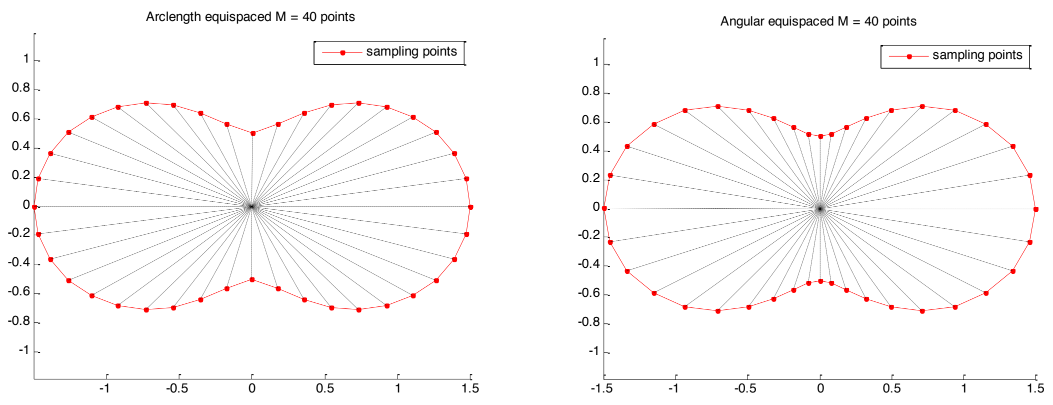
A correct solution of this problem would be an approach that doesn't depend on the initial parameterization $\boldsymbol{z}$, in which the curve is specified initially, and should depend only on the 'trace' of the function $\boldsymbol{z}$ in $\mathbb{R}^2$ (the set of all points that lie in the image $([0, 2\pi])$).

To modify the only the speed of traversal of the function $\boldsymbol{z}$ without changing its trace, we can send our parameter s through a reparameterization curve$\phi : [0, 2\pi] \to [0, 2\pi]$, such that the composition of the two functions $\boldsymbol{z}(\phi([0, 2\pi]))$ is the same as that of $\boldsymbol{z}([0, 2\pi])$.

This places a few restrictions on our re-speeding function ϕ:

a) $\phi(0) = 0$
b) $\phi(2\pi) = 2\pi$
c) $\phi'(0) = \phi'(2\pi)$
d) $\phi'(s) > 0$ for all $s \in [0,2\pi]$

These restrictions ensure that this re-speeding does not make $\boldsymbol{z} \circ \phi$ pass through the same point more than once, and that it is a smooth function (and by monotonicity, also ensure that ϕ is an invertible function). Therefore, $\boldsymbol{z} \circ \phi$ fulfills the same requirements of a parameterization as $\boldsymbol{z}$.



F3: Different parameterizations of the same curve yielding different 'equispaced points'

This modifies our integral to now be

$$I = \int_0^{2\pi} f(\boldsymbol{z}(s))\,|\boldsymbol{z}'(s)|\,ds = \int_0^{2\pi} f\left(\boldsymbol{z}(\phi(s))\right)\ \left|\boldsymbol{z}'(\phi(s))\right|\,\phi'(s)\,ds$$

Using the periodic quadrature scheme, this can now be approximated by

$$I \approx \frac{2\pi}{N}\sum_{j=1}^{N} f\left(\boldsymbol{z}(\phi(s_j))\right)\ \left|\boldsymbol{z}'(\phi(s_j))\right|\ \phi'(s_j)$$

which (if we choose our ϕ function carefully) should be a better approximation to the correct integral I than our previous approximation.

The Natural Parameterization

As discussed above, a correct solution to this problem would depend only on the shape of the curve that we are looking to integrate over, and be completely independent of the initial parameterization of the boundary that we are given. The first step in achieving this objective is to move to a parameterization that we are guaranteed to be only dependent on the shape – the 'natural parameterization', which traverses the boundary at a unit speed (measured in the metric of the target space $\mathbb{R}^2$)

This function can be defined by first finding the length function $\hat{y}(s)$ which defines the arc-length measured along the curve measured between a fixed point $\boldsymbol{z}(s_0)$ and $\boldsymbol{z}(s)$, which is given by:

$$\hat{y}(s) = \int_0^s |\boldsymbol{z}'(s)|\, ds$$

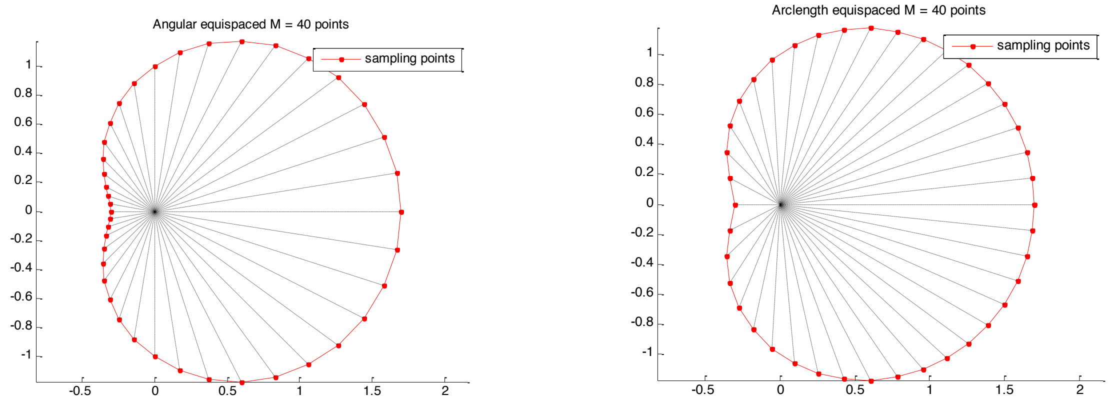
(s_0 is arbitrary, and we can just set it to 0)

Since $|\boldsymbol{z}'(s)| > 0$ for all $s \in [0,2\pi]$, we know that $\hat{y}(s)$ is a monotonically increasing function.

Let $L = \hat{y}(2\pi)$, and the function $y(s) = \frac{2\pi}{L}\,\hat{y}(s)$. We have now ensured that $y(2\pi) = 2\pi$, and this function fulfills all the criteria listed for a re-speeding, which means that it's inverse, which we will call ϕ_{nat} also fulfills all of these requirements.

The interpretation of the function $\boldsymbol{z}_{nat} = \boldsymbol{z} \circ \phi_{nat}$ is now: $\boldsymbol{z}_{nat}(s) =$ point on the curve such that the arc-length from $\boldsymbol{z}(0)$ till this point is the fraction $\frac{s}{2\pi}$ of the total length L. Therefore, this new parameterization $\boldsymbol{z}_{nat}$ has a constant value of $|\boldsymbol{z}'_{nat}| = \frac{L}{2\pi}$.

Using periodic quadrature on this parameterization involves using points that are evenly spaced along the curve, with distances measured along the curve, and therefore, the natural parameterization $\boldsymbol{z}_{nat}$ of a curve is a function only of the trace of the curve.



F3b

Implementation of the Natural Parameterization

In practice, the integral to find $y(s)$ is too difficult to perform analytically, and it is more practical to solve for y by just using a numerical solution to the ODE

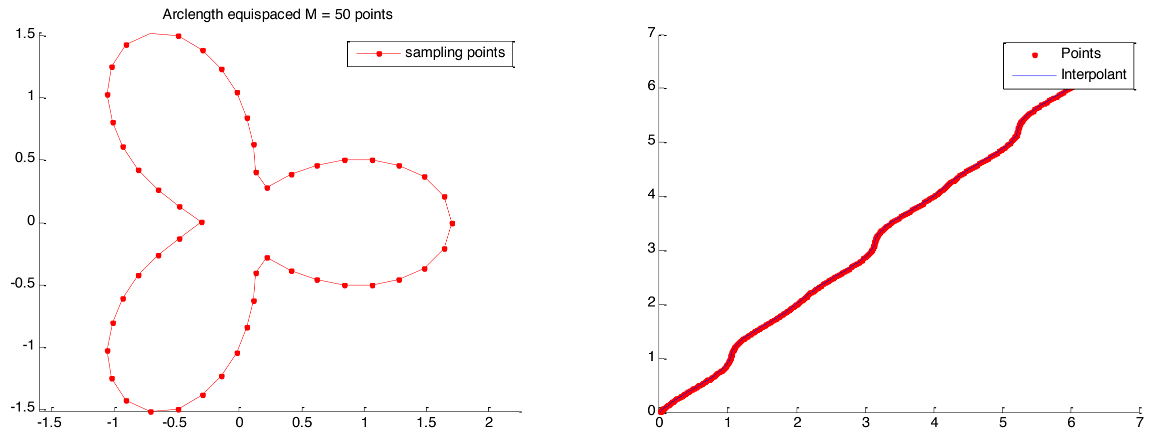
$$\frac{dy}{ds} = |\mathbf{z}'(s)|$$

This will yield, to reasonable accuracy a set of discrete points s_j, and the values $y(s_j)$. At this point, we can implement an interpolation scheme to be able to evaluate $y(s)$ for any point s.

Given that the number of points will be rather large, it doesn't make sense to high-order polynomials for Lagrange interpolation, or, in fact, any global interpolation scheme.

We opt for a local scheme, which means using piecewise polynomials in the form of clamped or natural cubic splines, or piecewise Hermitian cubic polynomials.

Using piecewise Hermitian cubic polynomials, or 'pchip's as Matlab calls them, we can actually guarantee that the interpolant will be monotonic for data points that are monotonically increasing (Fritsch, F. N. and R. E. Carlson, "Monotone Piecewise Cubic Interpolation," *SIAM J. Numerical Analysis*, Vol. 17, 1980, pp.238-246)



F4: The $\phi_{nat}(s)$ for the shape to it's left. We clearly see the 3 identical speed-ups and slow-downs right after them to represent the 3 leaves of the shape formed.

The pchip() function in matlab stores this as a 'PP structure', from which it is easy to evaluate the polynomial, and also quite easy to derive another PP structure for the derivative of the function, which is very useful when performing out integration and having to evaluate $\phi_{nat}{}'(s)$ at our collocated points.

Improving on the Natural Parameterization

The natural parameterization allows us to However, in most situations, the natural parameterization is not the parameterization that will yield the best approximation to our boundary integral. Intuitively speaking, the parameterization should slow down around the areas that have sharp turns, or approach other areas of the boundary very close, where our discrete approximations are likely to be more inaccurate.

In order to get any new parameterization that we want, we will now work with $\boldsymbol{z}_{nat}$ instead of $\boldsymbol{z}$, which effectively makes our solution unaffected by the initial parameterization.

Two possible approaches we could have are:

a) Generate a set of points s_j, and the values of $\phi(s_j)$, and from there, perform piecewise interpolation to obtain ϕ in the same way as we did for ϕ_{nat} in the previous section.
b) Somehow analytically derive a function ϕ that can be provided to the program directly to evaluate. (We will also have to provide the program an analytical derivative for this ϕ).

It is not clear whether or not an 'optimal' solution strategy for picking this parameterization is possible, so I tried a few different strategies that simulate the desired effect. They are listed below with some preliminary observations.

1) The first possible way to do this is, as suggested above, slow the curve down whenever there is a higher curvature. Since the curvature $\kappa(s)$ can assume both positive and negative values, but stays bounded for smooth shapes, we can use, in some form, the values of $e^{\alpha\kappa(s)}$, where $\alpha \in \mathbb{R}$ is some constant determined empirically by trial and error.

   Another possible approach would be to evaluate κ at a number of points evenly spaced in the natural parameterization, and then take a cumulative sum of those points. This cumulative sum can then be used as the discrete points s_j, and the values of $\phi(s_j)$.

   This approach was tried, without much success. It also has the problem of only taking into account the curvature, without seeing where our solution encounters problems with a different section of the boundary approaching too close to the section of the boundary in question. This issue is addressed in the approaches below.

2) The second approach is to try to apply an analogous approach to the Faraday Cage derivation of the Chebyshev nodes for the case of a linear interval.

   The Faraday Cage derivation is done by placing equal charges at random spots on the interval, and allowing them to redistribute themselves according to the forces acting on them until they equilibriate their positions to gain a full cancellation of field whenever there they are positions.

It can be shown that when limited to the interval $[-1, 1]$, these charges end up at the positions $\cos\left(\frac{2\pi j}{N}\right)$ for $j \in [1,2,3, \dots N]$, where N is the number of charges. These same nodes are also used as the standard nodes for the Clenshaw – Curtis quadrature scheme.

The analogous derivation in this case would be to limit the charges to $\partial\Omega$, and allow them to equilibrate their positions. Only forces tangential to the curve itself would act on the charges, because the charges are restricted to move within $\partial\Omega$.

Once the charges have settled (we know at what values of the parameter s they settle), we can use these values as $\phi(s_j)$, and create a ϕ that makes the new parameterization $\boldsymbol{z} \circ \phi$ have these locations as its periodic quadrature nodes.

This technique tends to also not work as well, because the charges tend to go as far away from the interior of the shape as possible, and attributes of the shape are not captured as intended.

The inverse of this parameterization was also tried (using $y(s)$ as ϕ instead of inverting as in the natural parameterization, to 'invert the density', but this was also ultimately not very successful.

3) The 3rd, most successful technique that was implemented was to first take N evenly spaced nodes on the natural parameterization, and then calculate the sum of the inverse squared distances to all other nodes for each given node.

   This has the advantage of capturing not only the corners and sharp turns in a shape, but also captures whenever there is a section of the boundary elsewhere approaching close to the node in question.

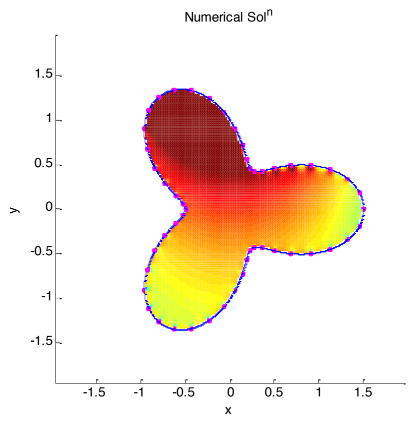
   The cumulative sum of this function, along with the nodes s_j, is used as set of data values to derive the interpolant ϕ. The results of this are shown in the figures.

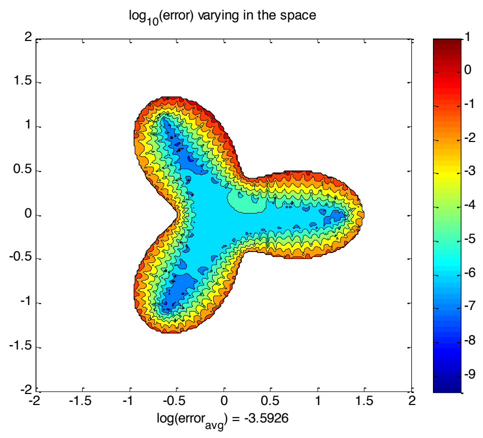
Some Implemented Results:

Something to be noted in some of the cases below: In figures parameterized by a constant added to a pure Fourier mode, the curvature is higher when nearing the origin, and therefore, the polar periodic quadrature (the naïve method mentioned earlier) performs exceptionally well. The 3rd successful strategy and the polar parameterization actually have very similar effects, in fact.

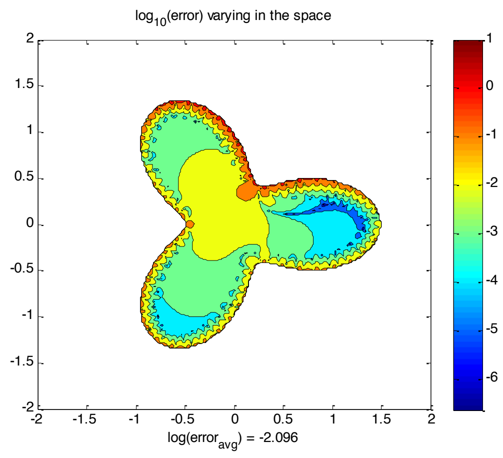
The natural parameterization is seen to clearly perform much worse than the others.

For example, this with shape:

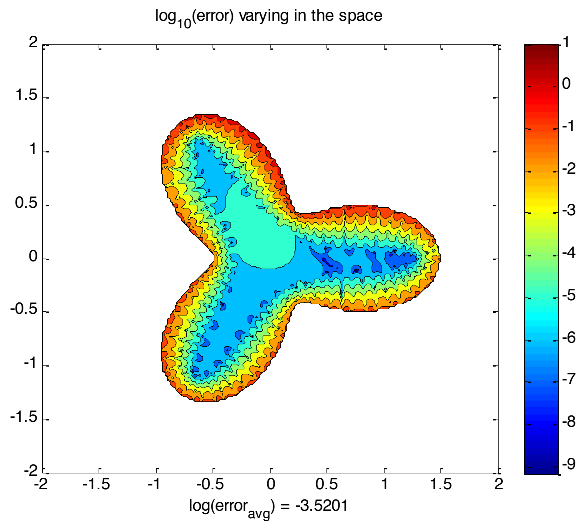




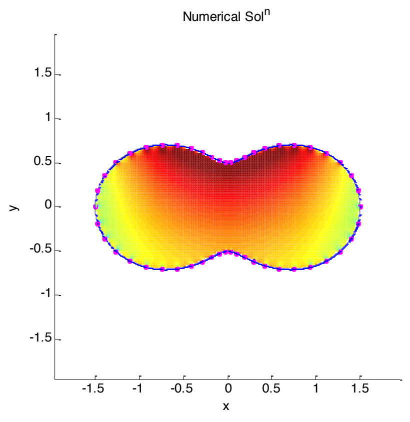
Shown above is the numerical solution and errors for polar periodic.

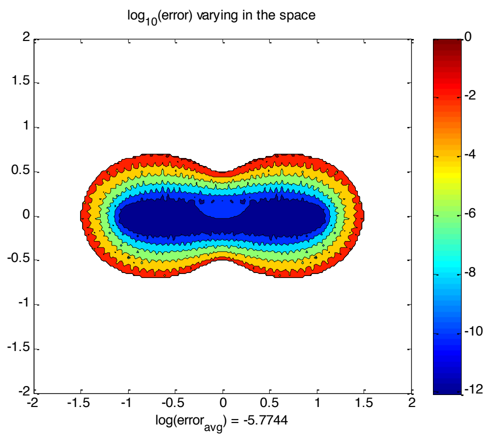


Accuracy is heavily diminished for natural parameterization, and almost matches polar if we use adaptive (which is the name used for option 3 described above)

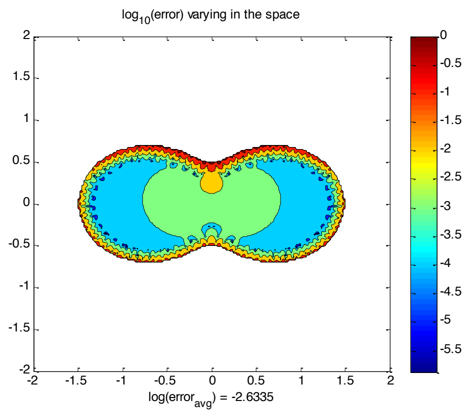


For example, this with shape:

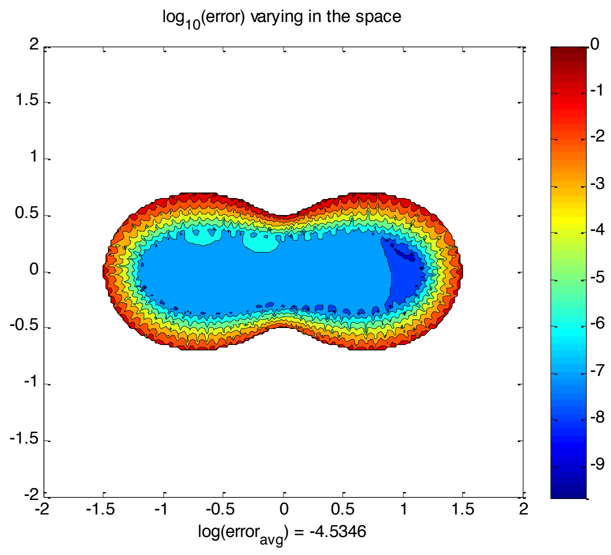


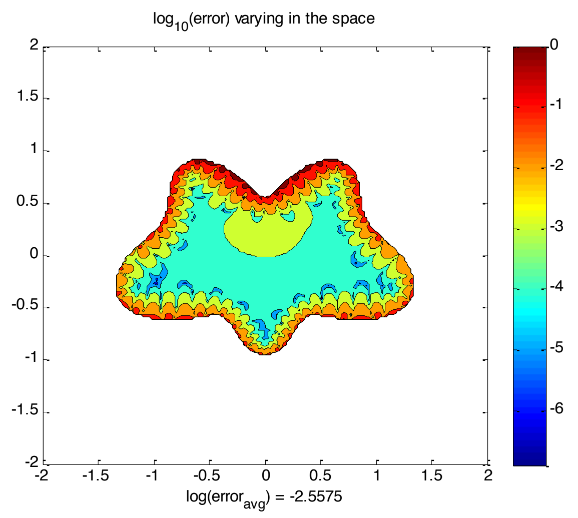


Shown above is the numerical solution and errors for polar periodic.

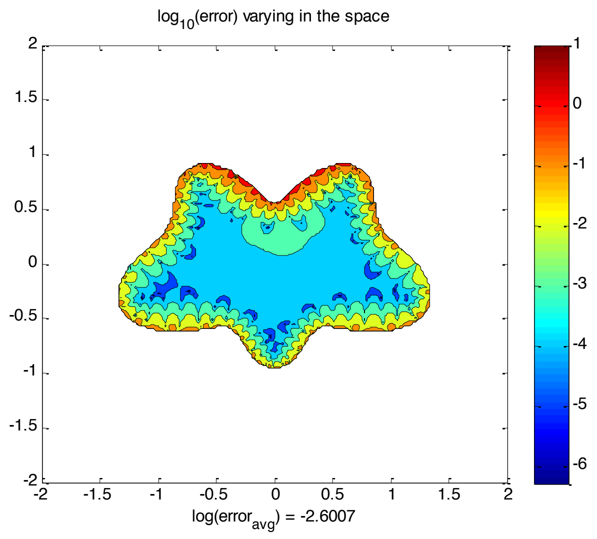


The exact same description as before applies to this set as well: Accuracy is heavily diminished for natural parameterization, and almost matches polar if we use adaptive

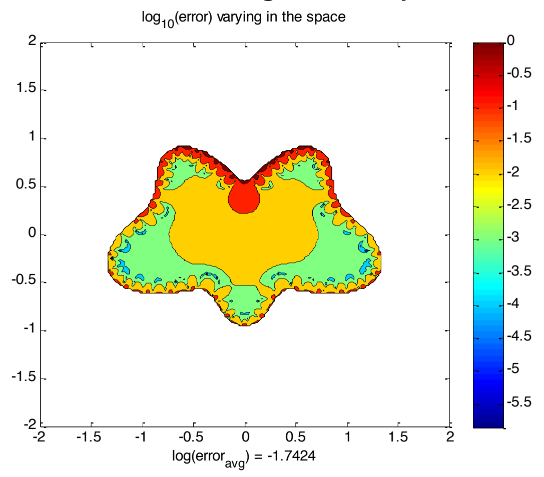




This inverse butterfly shape above, though, is one of the few shapes that I could construct for which the polar parameterization allowed for more error than the adaptive did.



The natural parameterization, though, is clearly nowhere near this good.


$\log_{10}$(error) varying in the space
$\log(\text{error}_{avg}) = -1.7424$