Approximating contour integral solutions of the parabolic wave equation

January 23, 2019

1 Problem statement

In [1], it was show that there exist families of solutions to the parabolic wave equation

$$2\mathrm{i}\frac{\partial u(x,y)}{\partial x}+\frac{\partial^2 u(x,y)}{\partial y^2}=0,\quad (x,y)\in\mathbb{R}^2.$$

These solutions were of the form:

$$u(x,y)=\int_\gamma \mathrm{e}^{\mathrm{i}g(t;x,y)}\mathrm{d}t, \tag{1}$$

where γ is an unbounded contour in $\mathbb{C}$ and $g(\cdot;x,y)$ is a polynomial with coefficients depending on x and y. Given that the integrand of (1) is entire, it follows by Cauchy's integral theorem that γ can be deformed onto any contour with the same (infinite) endpoints in $\mathbb{C}$. For stable numerical evaluation of (1), this contour should be chosen to be on, or close to, the path of steepest descent. One cannot expect a brute force quadrature approach to work if this path is not known, as it is very possible that the integrand will grow exponentially along a given contour, even if this integrand is zero at its endpoints.

2 Outline of code

PathFinder was developed with the intention of evaluating highly oscillatory integrals over a bounded subset of $\mathbb{R}$ using the method of numerical steepest descent. Although (1) is not highly oscillatory, we were able to use some components of PathFinder to determine a suitable contour γ, which is sufficiently close to the path of steepest descent.

Currently, for each $(x, y) \in \mathbb{R}^2$ a separate integral (1) must be evaluated, which means that evaluation on an $N \times N$ grid in $\mathbb{R}^2$ runs in $O(N^2)$ time. To increase convergence in N, we use a tensor product of two Chebyshev grids, and approximate $u(x, y)$ using Chebfun2. To gain back some additional CPU time, the code has been written to run in parallel, so for P cores the code runs in $O(N^2/P)$ time.

3 Sample code

Before the code can be used, you must ensure that Chebfun and PathFinder are added to the Matlab search path (including all subfolders).

Once all of the necessary directories have been added, we can define the key components of our integral (1). These are the endpoints of the contour γ and the coefficients of the polynomial g, which in this example are:

```
aValley = exp(9i*pi/10); bValley = exp(1i*pi/10);

Pcoeffs = @(X,Y) [2/5 -X/2 0 -Y 0 0];
```

Notice that the polynomial coefficients vector `Pcoeffs` is defined as an anonymous function, with dependency on X and Y. Next we choose the number of quadrature points per integral, which here we take to be a very prudent

```
Npts = 50;
```

and the DOFs in each direction on the Chebyshev grid, again a very prudent vector of the form $[x \text{ DOFs}, y \text{ DOFs}]$

```
degs = [1000 1000];
```

and the range over which the Chebfun will be defined, as a vector the form $[x_-, x_+, y_-, y_+]$,

```
range = [-10 10 -10 10];
```

Now we have defined the necessary parameters, we can call the main function

```
A31 = Aij2(aValley, bValley, Pcoeffs, 1, Npts, degs, range);
```

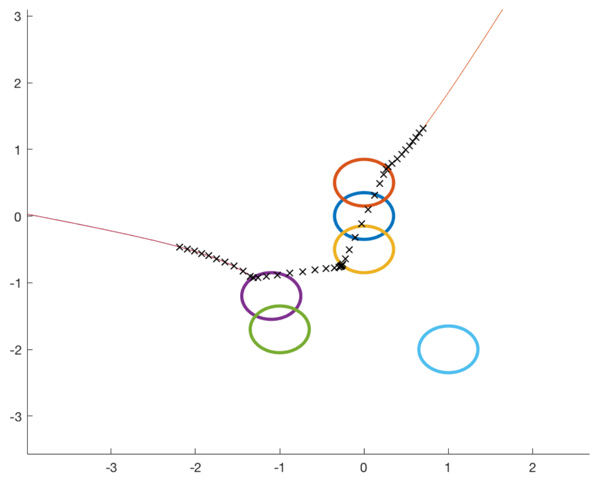

Figure 1: SD paths and nodes for (2) in $\mathbb{C}$.

typically `degs = [250 250];` runs in around 45 minutes on my laptop (using two cores), but due to the $O(N^2)$ CPU time, the above example would need to be left overnight (as it would take around 12 hours). `A31` will be a Chebfun, and can therefore be evaluated anywhere in $[-10, 10]^2$.

In the PathFinder2 folder, `Challenge1.m` and `Challenge2.m` are similar to the above code, approximating Dave's challenges. Another great way to try and break the code is using `coalesceTest.m`, which allows you to specify stationary points manually, the code then constructs a polynomial phase with these stationary points, and attempts to determine the a suitable contour. An example is depicted in Figure 1 for a phase with

$$g'(z) = (z + .5\mathrm{i})(z)(z - .5\mathrm{i})(z + 1 + 1.7\mathrm{i})(z - 1 + 2\mathrm{i})(z + 1.1 + 1.2\mathrm{i}), \quad (2)$$

and a contour with endpoints at $\exp(2i * pi * (1/4 + 3)/7) + \exp(2i * pi * (1/4 + 1)/7)$. From this figure the outline of the algorithm is clear - a ball is constructed around each stationary point. Outside of these balls the steepest descent path is constructed, and inside of these balls, standard quadrature is used.

4 Some results

Figure 2 shows plots of Challenge 1, for 250 × 250 Chebyshev nodes, which appear sufficient in the eyeball norm.

Figure 3 shows some plots of Challenge 2, for 250 × 250 and 1000 × 1000 Chebyshev nodes respectively. In the eyeball norm it appears to be converging, although still not in total agreement with Dave's plot. There appear to be oscillations on a much finer scale than for Challenge 1. It should also be noted that the right-hand plot took around 12 hours to produce, so either more cores, more time, or a more sophisticated approach is necessary.

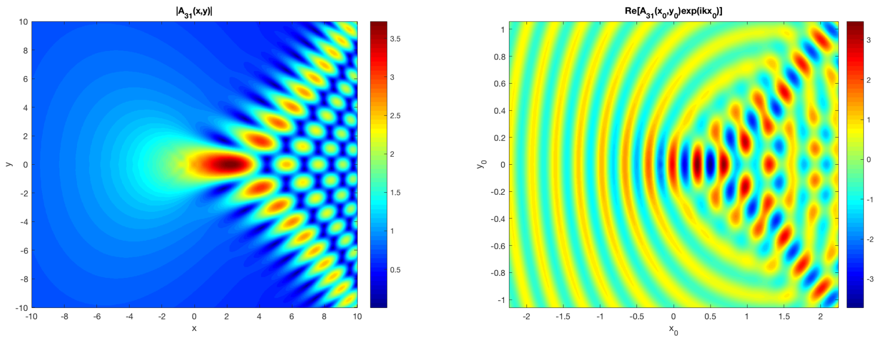

Figure 2: Approximations of Challenge 1

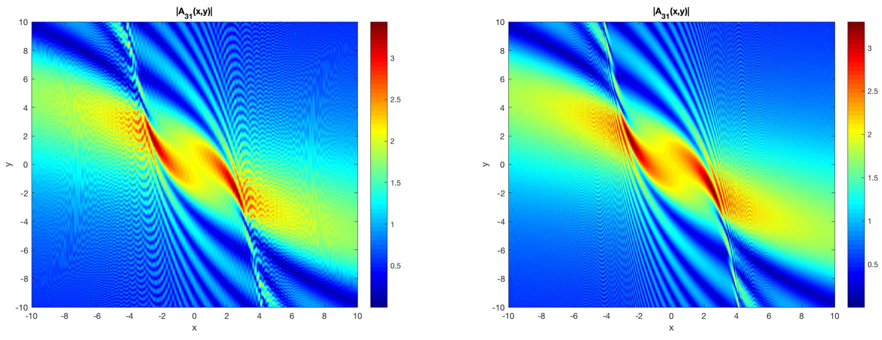

Figure 3: Approximations of Challenge 2

5 A more sophisticated approach

Each integral takes around 0.05s to approximate. This adds up when there are tens of thousands of integrals. A more sophisticated approach would involve partitioning the domain of interest (e.g. $[-10, 10]^2$) into small subdomains, and a quadrature rule for the point at the centre of the domain is obtained. From here,

1. This quadrature rule can be used for all other points in the domain, as the domain width and the perturbations in the phase g are sufficiently small that we are still nearly on the path of steepest descent.
2. This quadrature rule can be used as a starting point, which can then be iterated towards the path of steepest descent.

References

[1] David P. Hewett, John R. Ockendon, Valery P. Smyshlyaev. *Contour integral solutions of the parabolic wave equation.* arXiv:1806.02294v2, 2018.