1 Report 10/09/2020

2 Sparse to Fine

Running a problem on a sparse grid and interpolating onto a fine one. The sparse grid is $N = n = 10$, the fine one is $N = 20$, $n = 10$. The tolerances for the sparse grid are $0.1/10^{-4}$ and for the fine grid they are $10^{-3}/10^{-7}$. The sparse grid problem converges in 118 iterations, and takes 57 seconds (around 0.5 s per iteration). The fine grid however converges in 669 iterations and 6.2888×10^3 seconds (around 9 s per iteration). So there is no gain in doing this; at least not the way I have implemented this. I only tested one example though, so there may be a case for looking into this more.

3 In and outflow problem

Something weird is going on with this problem. I set up a target which has a small inflow, so that mass accumulates at the outflow, see Figure 1. Then I set the OCP so that the in and outflows are both 1 and ρ is also 1 initially. I would expect it wants to force mass to the outflow. Instead it accumulates mass at the inflow. This happens for other targets too. See Figure 2 and 3 for the first iteration solution. After two iterations, ρ gets pushed up that boundary to 200, which I think is what causes this to crash.

4 Interpolation Issue

The issue was that some points got lost in the interpolation matrix. Figure 4 shows where they got lost. This indicates that they are too close together and get deleted at a later point. This is fixed by adding a small value to the boundaries of the shape, see Figure 5. Is that allowed/ okay to do. It seems to fix the problem anyway.

5 Examples

5.1 Adaptive λ

We use an adaptive λ for the below examples to decrease computational time. We do this as follows:
Set $k = 0$ and k increases by one per iteration.

1. If $\mathcal{E}_n$ ¿ $\mathcal{E}_o$, and $k > 10$, set $k = k - 10$, and $w_{i+1} = w_i$.
2. Elseif $\mathcal{E}_n$ ¿ $\mathcal{E}_o$, and $k < 10$, set $k = k$, and $w_{i+1} = w_i$.

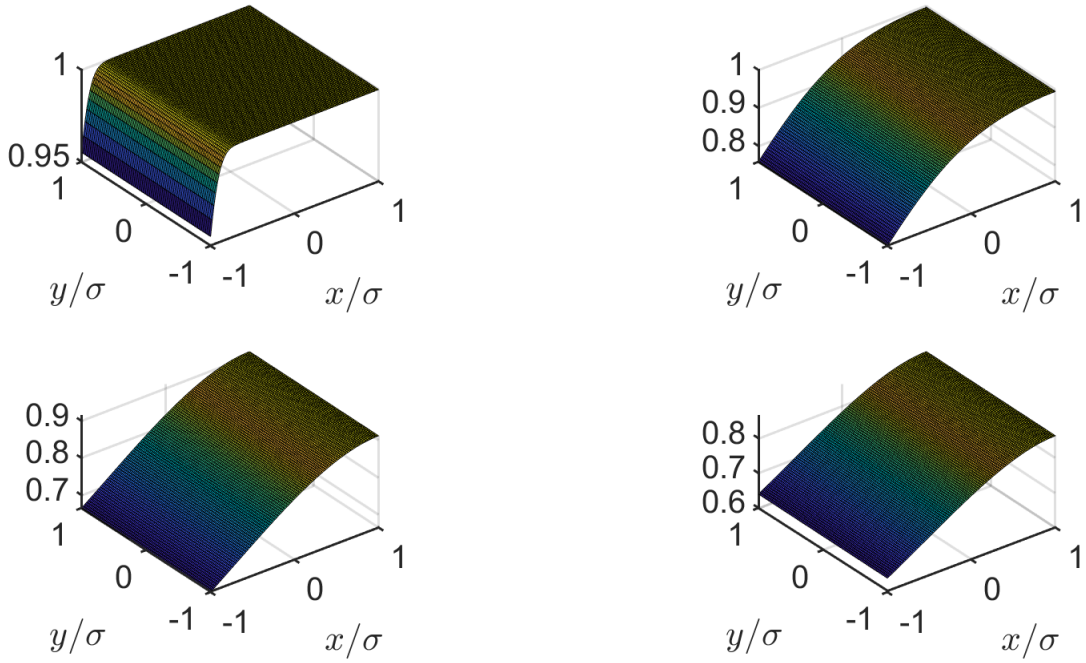

Figure 1: Target

3. If $\mathcal{E}_n = \mathcal{E}_o$, the $\lambda = 0.5\lambda$, and w_{i+1} is found by mixing scheme.

4. If $\mathcal{E}_n$ ¡ $\mathcal{E}_o$, them:

   - if $k < 10$, $\lambda = 0.01$.
   - if $k < 20$, $\lambda = 0.02$.
   - if $k < 30$, $\lambda = 0.03$.
   - if $k < 40$, $\lambda = 0.05$.
   - if $k < 50$, $\lambda = 0.06$.
   - if $k < 60$, $\lambda = 0.08$.
   - else, $\lambda = 0.1$.

After running the examples, I now changed step 2 to be the same as step 3.

5.2 Example1

Note: for $T = 1$, we had each iteration at 80 seconds. Now it is 35 seconds. Also. We had convergence in $500 - 600$ iterations. Now it is around 100.

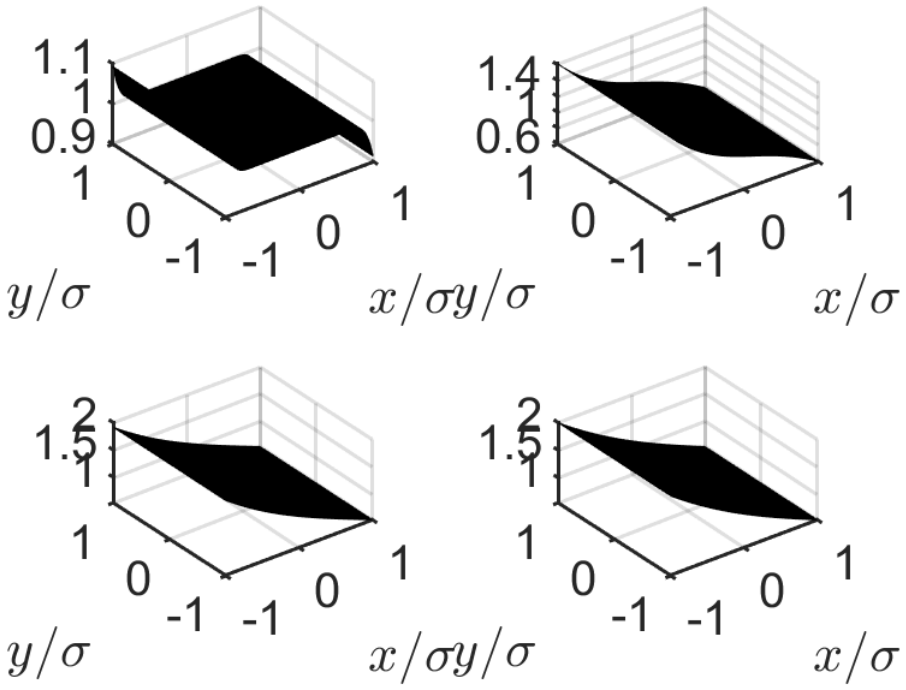

Figure 2: ρ after one iteration

This example runs with $T = 1.75$. We choose $N = n = 20$, tols $10^{-7}/10^{-3}$ and $\beta = 10^{-3}$. For $\kappa = -1$ we get convergence in 125 iterations, which takes 4.4314×10^3 seconds, which is around seconds per iteration. $J_{FW} = 0.3654$, $J_{Opt} = 0.0661$. See Figures 6 and 7. For $\kappa = 1$ we get convergence in 114 iterations, which takes 4.0090×10^3 seconds, which is around 35 seconds per iteration. $J_{FW} = 0.3090$, $J_{Opt} = 0.0667$.See Figures 8 and 9.
In both examples, in the OCP, the flow does not match the one from the target at the end. I think this is due to $\vec{w} = \vec{=}0$ at T.

5.3 Example 2

This is adapted from the original 'fancy channel'. Running the original takes a lot longer because there are more shapes involved.
In comparison: without the fix for time this would take 32 seconds per iteration (around 20 now). Without the Adaptive algorithm, it would take 467 iterations (around 100 now) to solve the problem with $\kappa = 0$. See Figures 10 and 11.
For $\kappa = -1$ we get convergence in 101 iterations, which takes 1.9365×10^3 seconds, which is around 20 seconds per iteration. $J_{FW} = 0.0184$, $J_{Opt} = 9.5323 \times 10^{-4}$. See Figures 12 and 13.
For $\kappa = 1$ we get convergence in 87 iterations, which takes 1.6845×10^3 seconds, which is around

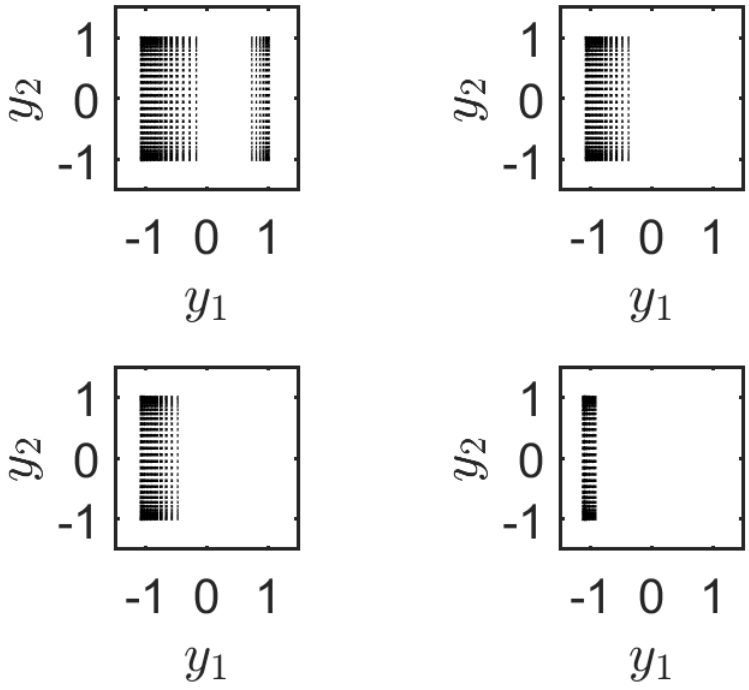

Figure 3: $\vec{w}$ after one iteration

20 seconds per iteration. $J_{FW} = 0.0150$, $J_{Opt} = 0.0010$. See Figures 14 and 15.

6 Other

- Memory issue with datastorage.

- Background section: Some DDFT background and some statistical mechanics stuff?

- Classes: As discussed last week, it makes most sense to take the MAC-MIGS course for credit this semester and the Python course for general skills credits. I am enrolled in Fundamentals of Optimization as class only student.
If possible, I would like to be enrolled in the following School of Informatics courses: 'Design and Analysis of Parallel Algorithms' and 'Numerical Algorithms for High Performance Computing'. I am not sure if these are what I think they are, but I would just like to have a look.

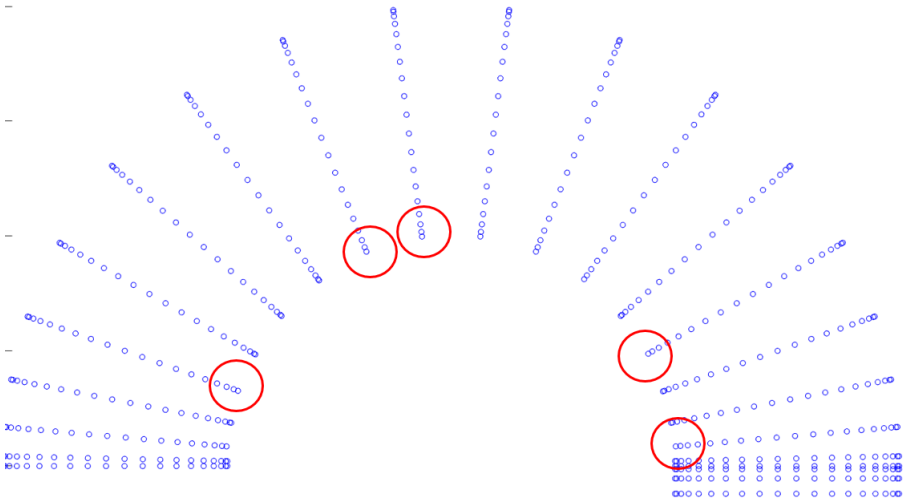

Figure 4: Missing points

```
289         function Interp = ComputeInterpolationMatricesPhys(this,Pts)
290
291             Y1 = Pts.y1_kv;
292             Y2 = Pts.y2_kv;
293
294             Y1_cut = [];
295             Y2_cut = [];
296             InterPol = [];
297
298             % loop through shapes and determine which points lie in the
299             % shape
300             for iShape = 1:this.nShapes
301
302                 InterPolShape = [];
303
304                 thisShape = this.Shapes(iShape).Shape;
305
306                 x1 = thisShape.Pts.x1;
307                 x2 = thisShape.Pts.x2;
308                 x1Min = min(x1)-10^-10;
309                 x1Max = max(x1)+10^-10;
310                 x2Min = min(x2)-10^-10;
311                 x2Max = max(x2)+10^-10;
312
313                 % transform all points from physical to computational space
314                 % and deal with polar coordinates
315                 if(strcmp(thisShape.polar,'polar'))
316                     Origin = thisShape.Origin;
317                     [Theta,R] = cart2pol(Y1-Origin(1),Y2-Origin(2));
318                     Theta = mod(Theta,2*pi);
319                     [X1,X2] = this.Shapes(iShape).Shape.CompSpace(R,Theta);
320                 else
321                     [X1,X2] = this.Shapes(iShape).Shape.CompSpace(Y1,Y2);
322                 end
323
324                 % only keep those which came from the shape
325                 keepMask = (X1>=x1Min) & (X1<=x1Max) & (X2>=x2Min) & (X2<=x2Max);
326                 X1_keep = X1(keepMask);
327                 X2_keep = X2(keepMask);
328                 Y1_keep = Y1(keepMask);
329                 Y2_keep = Y2(keepMask);
330
331                 % compute interpolation matrix using shape
332                 nPoints = length(X1_keep);
333                 for iPoint = 1:nPoints
334                     InterpShape = thisShape.ComputeInterpolationMatrix(X1_keep(iPoint),X2_keep(iPoint));
335                     InterPolShape = [InterPolShape;InterpShape.InterPol]; %#ok
336                 end
337
338                 % combine all together
339                 Y1_cut = [Y1_cut;Y1_keep]; %#ok
340                 Y2_cut = [Y2_cut;Y2_keep]; %#ok
341                 InterPol = blkdiag(InterPol,InterPolShape);
```

Figure 5: Interpolation Code

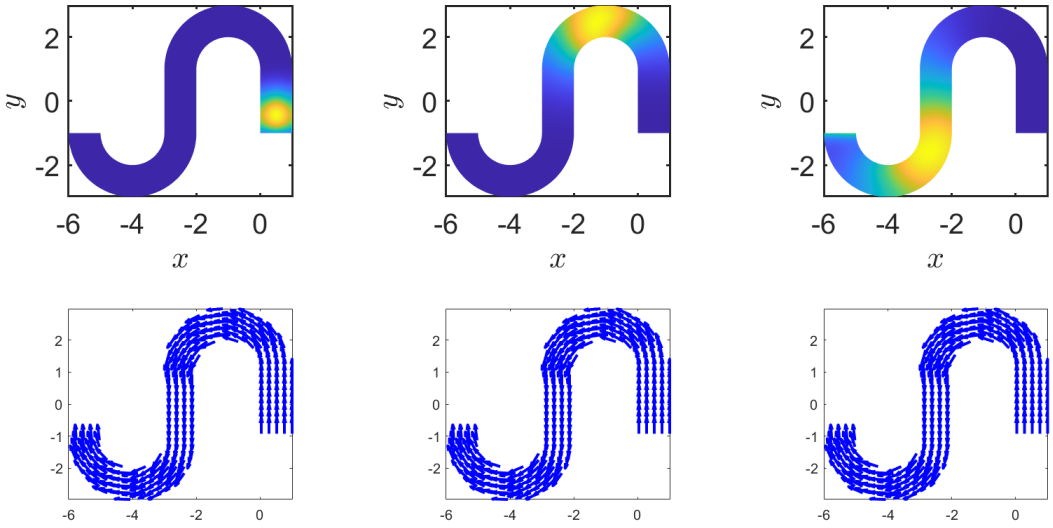

Figure 6: Target with $\kappa = -1$

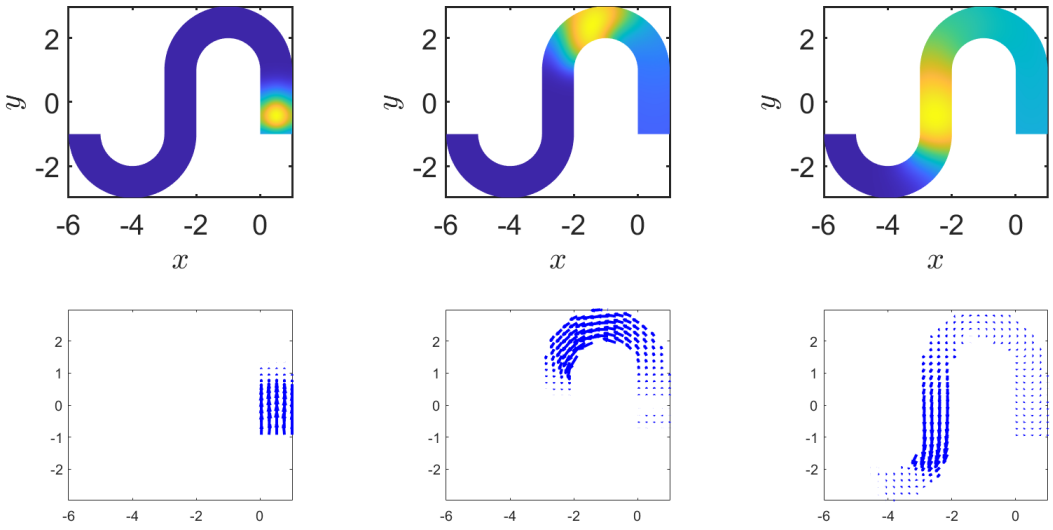

Figure 7: Optimal Result with $\kappa = -1$

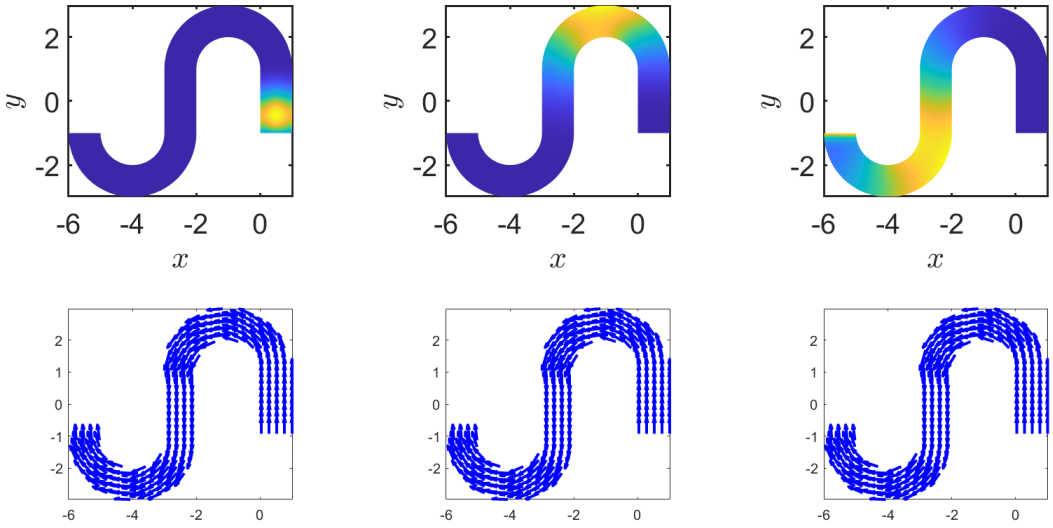

Figure 8: Target with $\kappa = 1$

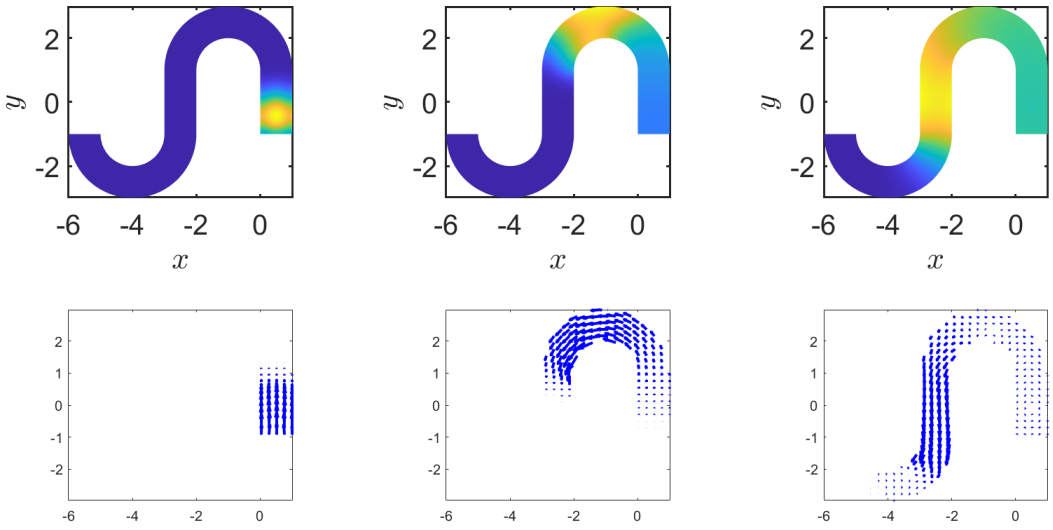

Figure 9: Optimal Result with $\kappa = 1$

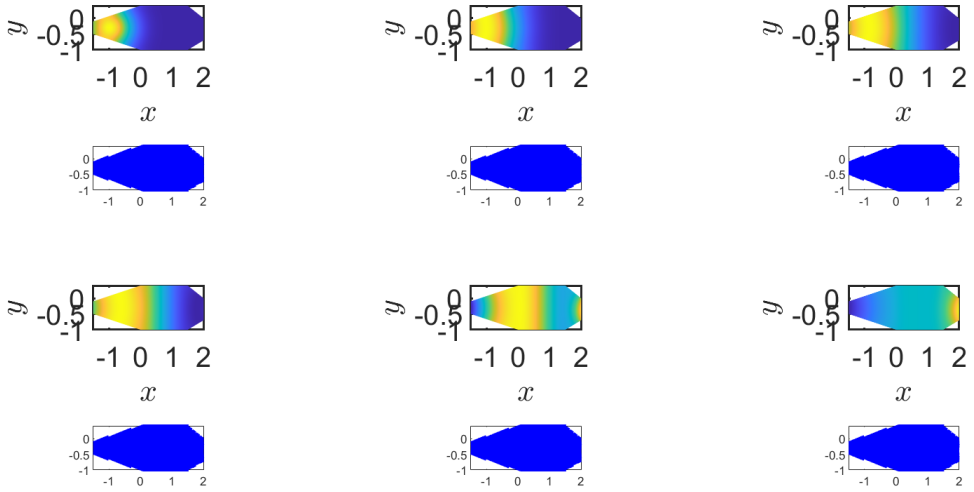

Figure 10: Forward Problem with $\kappa = 0$

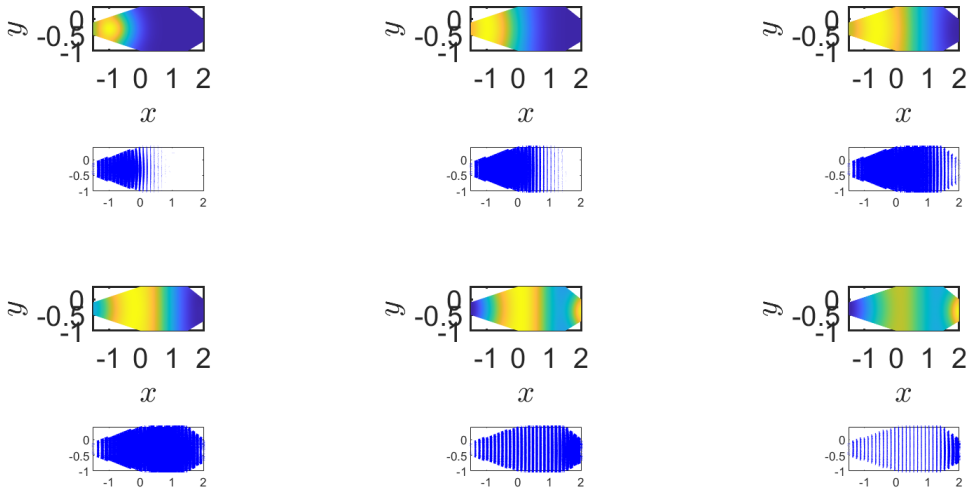

Figure 11: Optimal Result with $\kappa = 0$

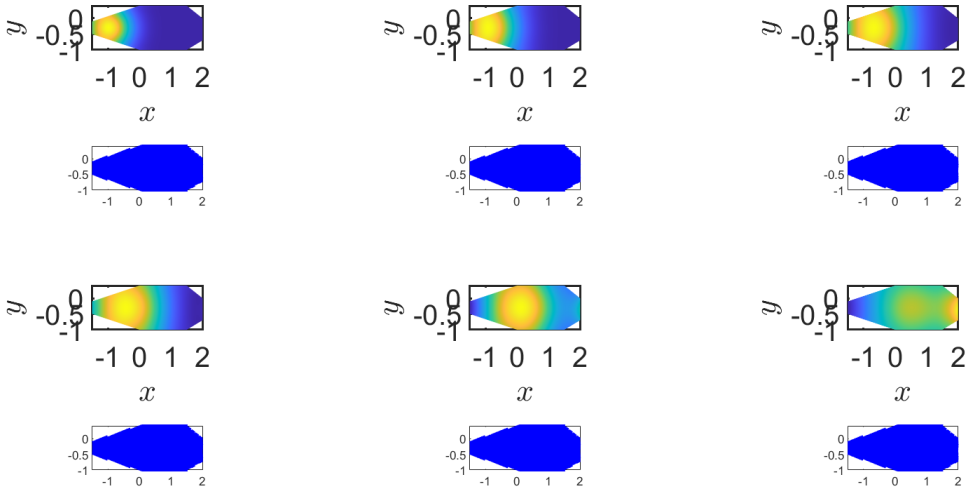

Figure 12: Forward Problem with $\kappa = -1$

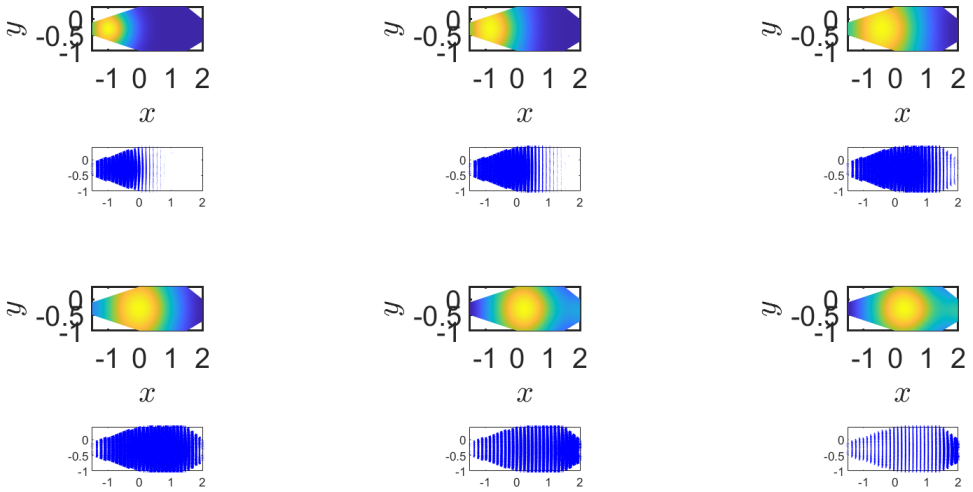

Figure 13: Optimal Result with $\kappa = -1$

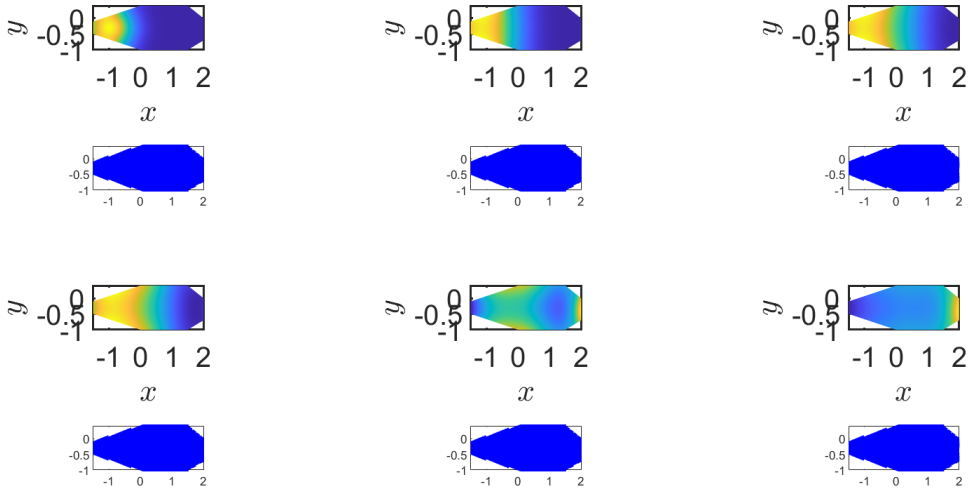

Figure 14: Forward Problem with $\kappa = 1$

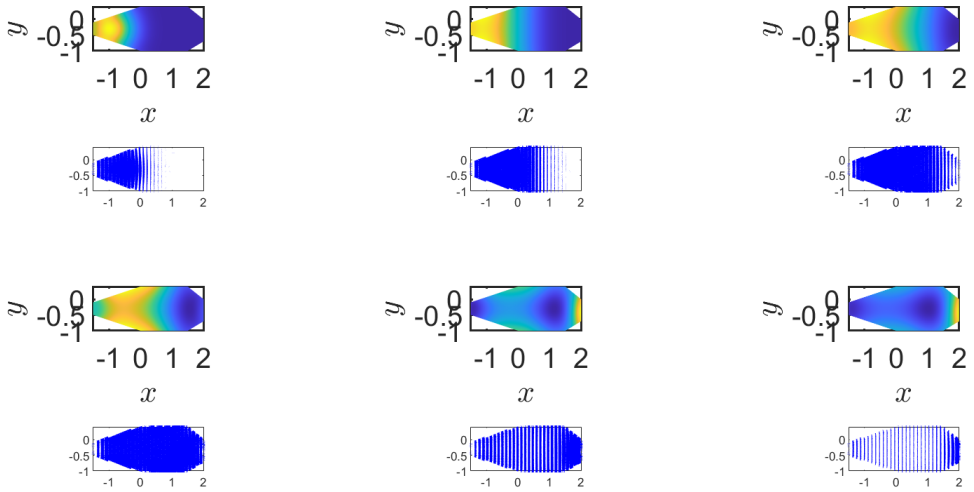

Figure 15: Optimal Result with $\kappa = 1$