# Report 27/08/2020

Delete Fluke example from paper code!!

Implemented the dot product in shape and mulitshape. For multishape the code can be seen in Figure 1. The implementation of the adjoint integral term is in Figure 2. The term we are

```
%%%% The dot product function for MultiShape
    function dprod = dotProduct(this, u, v)

        [u1,u2] = this.SplitVector(u);
        [v1,v2] = this.SplitVector(v);
        dprod = zeros(length(this.Pts.y1_kv), 1);
        for iShape = 1:this.nShapes

            if(strcmp(this.Shapes(iShape).Shape.polar,'polar'))
                mask = this.Shapes(iShape).PtsMask;
                lower = (iShape-1)*length(u(mask)) + 1;
                upper = iShape*length(u(mask));
                u1p = u1(mask);
                u2p = u2(mask);
                v1p = v1(mask);
                v2p = v2(mask);
                radius = (u1p.*v1p);
                angle = cos(u2p - v2p);
                dprod(lower:upper) = radius.*angle;
            elseif (strcmp(this.Shapes(iShape).Shape.polar,'cart'))
                mask = this.Shapes(iShape).PtsMask;
                lower = (iShape-1)*length(u(mask)) + 1;
                upper = iShape*length(u(mask));
                u1c = u1(mask);
                u2c = u2(mask);
                v1c = v1(mask);
                v2c = v2(mask);
                dprod(lower:upper) = (u1c.*v1c) + (u2c.*v2c);
            end

        end

    end
```

Figure 1: Implementation of the dot product

trying to implement there is:

$$\nabla_r \int V_2(|r - r'|)\nabla_{r'} p(r')dr'.$$

## 1 Tests

Tested the forward problem and optimal control problem in different settings. It all seems to be working very well. Forward Tests (Figure 3):

- Test 1: Compare computations on a box, with ADInf solution.
- Test 2: Compare on a box with AD Flow Neumann Exact solution.

```
%%% Computing the second interaction term for the adjoint equation:
%%% Grad(Conv*((Grad*p)*rho))

% The only difference would be how to get Grad1/ Grad2:

    if isa(this.IDC, 'MultiShape')
        y1Mask = this.IDC.y1Mask;
        y2Mask = this.IDC.y2Mask;
        Grad1 = Grad(y1Mask,:);
        Grad2 = Grad(y2Mask,:);
    elseif isa(this.IDC, 'Box')
        Grad1=Grad(1:end/2,:);
        Grad2=Grad(end/2+1:end,:);
    end
        Conv1 = Conv*((Grad1*p).*rho);
        Conv2 = Conv*((Grad2*p).*rho);
        l= Grad1*Conv1 + Grad2*Conv2;
```

Figure 2: implementation of adjoint integral term

- Test 3: Split box in MS code and compare to box in Box code using ADInf solution
- Test 3a: Same as 3 only checking that order of shapes don't matter.
- Test 4: Computing problem ADInf on wedge + quadrilateral
- Test 4a: Same as 4 only checking that order of shapes don't matter.
- ToyProblem 1: Computes no flux problem on two wedges and two quadrilaterals with constant 1 flow.

Optimization Tests (Figure 4):

- Test 5: Comparing MS and Box code on a box with Neumann Flow Exact Problem
- Test 6: Split MS box in two parts and compare to box in Box code (same Exact solution)
- Test 7: Comparing on the box an interacting problem (problem one from paper)
- Test 8: Splitting MS box and comparing to Box code for interacting problem

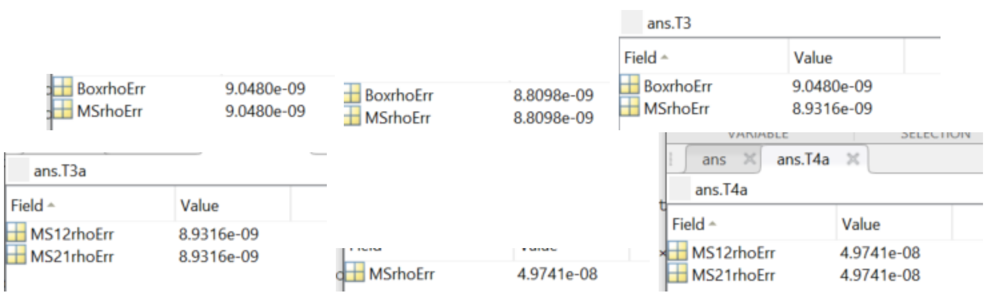


Figure 3: Forward Test Solutions

| Field | Value |
|---|---|
| BoxrhoErr | 8.8098e-09 |
| BoxpErr | 1.7637e-08 |
| BoxwErr | 2.8968e-06 |
| MSrhoErr | 8.8098e-09 |
| MSpErr | 1.7637e-08 |
| MSwErr | 2.8968e-06 |

ans.T6

| Field | Value |
|---|---|
| BoxrhoErr | 8.8098e-09 |
| BoxpErr | 1.7637e-08 |
| BoxwErr | 2.8968e-06 |
| MSrhoErr | 3.4497e-08 |
| MSpErr | 1.7640e-08 |
| MSwErr | 2.8965e-06 |

ans.T7

| Field | Value |
|---|---|
| BoxJFW | 0.0113 |
| BoxJOpt | 0.0112 |
| MSJFW | 0.0113 |
| MSJOpt | 0.0112 |

ans.T8

| Field | Value |
|---|---|
| BoxJFW | 0.0111 |
| BoxJOpt | 0.0110 |
| MSJFW | 0.0111 |
| MSJOpt | 0.0110 |

Figure 4: Optimization Test Solutions

## Comparing Matching Conditions in Forward problem

Compared with ADinf exact solution. Matching two boxes (vs full box) in Figure 5 and matching two wedges (vs full wedge) in Figure 6. Both show that the results are the same regardless of the matching method. Using the example with no flux and two wedges and two boxes (see Figure 7), the two matching methods are compared. The error in $\rho$ is $1.4292 \times 10^{-10}$.

## 2 Finding distances between points that leave the domain

There must be a better way of doing this, because this seems impossibly inefficient, but the way I think of this is the following, see Figure 8. For each pair of points $\vec{x}_1$, $\vec{x}_2$ do:

1. Check if points lie ON the boundary at first.
2. Find the linear function connecting $\vec{x}_1$, $\vec{x}_2$, $f$.
3. Consider each boundary in the domain $[x_1, x_2] \times [y_1, y_2]$, in particular, each side of each boundary.
4. If boundary is Cartesian:
   Choose two points $\vec{b}_1$, $\vec{b}_2$ on the boundary segment. Define the linear function connecting these two points, $h$.
5. Find the intersection between $f$ and $h$, in particular check $m_1 - m_2 \neq 0$. Check that the solution $(a, b) \in [x_1, x_2] \times [y_1, y_2]$.
6. If boundary is Polar:
   Either convert to Cartesian or get $f$ in polar.

```
>> MS_TestJonnaADExactDisect2
MS_UT_CompareInterpolation
MS_UT_CompareInterpolation: errUniform1 = 5.5511e-17
MS_UT_CompareInterpolation: errUniform2 = 1.249e-16

MS_UT_CompareConvolution
MS_UT_CompareConvolution: err = 9.0563e-12
MS_UT_CompareConvolution: errRel = 6.0663e-11

MS_UT_DiffusionExact
MS_UT_DiffusionExact: errL2Exact = 6.1376e-08
MS_UT_DiffusionExact: errL2ExactRel = 6.5235e-09

MS_UT_DiffusionExact
MS_UT_DiffusionExact: errL2Exact = 4.3625e-08
MS_UT_DiffusionExact: errL2ExactRel = 6.299e-09

MS_UT_DiffusionExact
MS_UT_DiffusionExact: errIC = 2.2204e-16
MS_UT_DiffusionExact: errICRel = 3.4156e-15
MS_UT_DiffusionExact: err12 = 1.4978e-11
MS_UT_DiffusionExact: err12Rel = 9.6269e-10

10  viewport.Units = fromunits;
MS_UT_AdvectionDiffusionExact
MS_UT_AdvectionDiffusionExact: errL2Exact = 4.3178e-07
MS_UT_AdvectionDiffusionExact: errL2ExactRel = 5.153e-09

MS_UT_AdvectionDiffusionExact
MS_UT_AdvectionDiffusionExact: errL2Exact = 3.0575e-07
MS_UT_AdvectionDiffusionExact: errL2ExactRel = 5.1351e-09

MS_UT_AdvectionDiffusionExact
MS_UT_AdvectionDiffusionExact: errIC = 6.2172e-15
MS_UT_AdvectionDiffusionExact: errICRel = 2.9518e-15
MS_UT_AdvectionDiffusionExact: err12 = 1.2017e-10
MS_UT_AdvectionDiffusionExact: err12Rel = 4.9392e-11

MS_UT_InteractingAdvectionDiffusion
MS_UT_InteractingAdvectionDiffusion: errIC = 2.1649e-15
MS_UT_InteractingAdvectionDiffusion: errICRel = 4.3299e-15
MS_UT_InteractingAdvectionDiffusion: err12 = 1.7618e-11
MS_UT_InteractingAdvectionDiffusion: err12Rel = 2.6302e-11
```

```
>> MS_TestJonnaADExactDisect2
MS_UT_CompareInterpolation
MS_UT_CompareInterpolation: errUniform1 = 5.5511e-17
MS_UT_CompareInterpolation: errUniform2 = 1.249e-16

MS_UT_CompareConvolution
MS_UT_CompareConvolution: err = 9.0563e-12
MS_UT_CompareConvolution: errRel = 6.0663e-11

MS_UT_DiffusionExact
MS_UT_DiffusionExact: errL2Exact = 6.1376e-08
MS_UT_DiffusionExact: errL2ExactRel = 6.5235e-09

MS_UT_DiffusionExact
MS_UT_DiffusionExact: errL2Exact = 4.3625e-08
MS_UT_DiffusionExact: errL2ExactRel = 6.299e-09

MS_UT_DiffusionExact
MS_UT_DiffusionExact: errIC = 2.2204e-16
MS_UT_DiffusionExact: errICRel = 3.4156e-15
MS_UT_DiffusionExact: err12 = 1.4978e-11
MS_UT_DiffusionExact: err12Rel = 9.6269e-10

MS_UT_AdvectionDiffusionExact
MS_UT_AdvectionDiffusionExact: errL2Exact = 4.3178e-07
MS_UT_AdvectionDiffusionExact: errL2ExactRel = 5.153e-09

MS_UT_AdvectionDiffusionExact
MS_UT_AdvectionDiffusionExact: errL2Exact = 3.0575e-07
MS_UT_AdvectionDiffusionExact: errL2ExactRel = 5.1351e-09

MS_UT_AdvectionDiffusionExact
MS_UT_AdvectionDiffusionExact: errIC = 6.2172e-15
MS_UT_AdvectionDiffusionExact: errICRel = 2.9518e-15
MS_UT_AdvectionDiffusionExact: err12 = 1.2017e-10
MS_UT_AdvectionDiffusionExact: err12Rel = 4.9392e-11

MS_UT_InteractingAdvectionDiffusion
MS_UT_InteractingAdvectionDiffusion: errIC = 2.1649e-15
MS_UT_InteractingAdvectionDiffusion: errICRel = 4.3299e-15
MS_UT_InteractingAdvectionDiffusion: err12 = 1.7618e-11
MS_UT_InteractingAdvectionDiffusion: err12Rel = 2.6302e-11
```

Figure 5: Boxes with matching first derivative and flux

# 3 OCP on MultiShape

## 3.1 Example 1

We choose the initial condition for $\rho$ to be $exp(-2((y1-0.5)^2+(y2+0.5)^2))$ and solve a forward problem with constant velocity of strength one, see Figure 9. Then we use this forward solution as a target in the OCP, with initial velocity zero. We set $\beta = 10^{-3}$, we solve with tolerances $10^{-7}/10^{-3}$, because of time constraints, and $n = 20$, $N = 20$. The solution can be seen in Figure 10. As expected, the control follows the particle mass. It takes 452 iterations, but the time it takes is $2 \times 10^4$. We get $J_{FW} = 0.0206$, $J_{Opt} = 0.0020$. We then choose $\kappa = -1$ and get $J_{FW} = 0.0251$, $J_{Opt} = 0.0020$, in 454 iterations taking $1 \times 10^4$ in time. We can see the results in Figures 11 and 12. We can do the same for $\kappa = 1$. We get $J_{FW} = 0.0176$, $J_{Opt} = 0.0020$, in 451 iterations taking $1 \times 10^4$ in time. We can see the results in Figures 13 and 14.

## 3.2 Example 2

We choose the initial condition for $\rho$ to be $exp(-2((y1-0.5)^2+(y2+0.5)^2))$ and solve a forward problem with constant velocity of strength five, see Figure 15. Then we use this forward solution as a target in the OCP, with initial velocity zero. We set $\beta = 10^{-3}$, we solve with tolerances

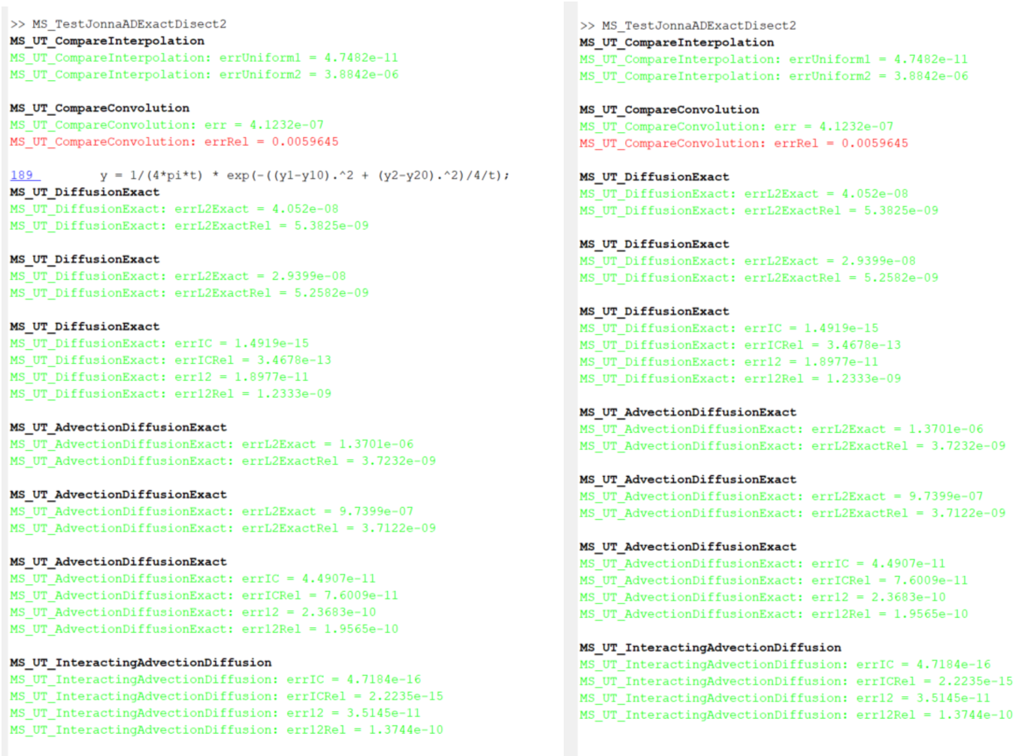


Figure 6: Wedge with matching first derivative and flux

$10^{-7}/10^{-3}$, because of time constraints, and $n = 20$, $N = 20$. The solution can be seen in Figure 16. Again, as expected, the control follows the particle mass. It takes 587 iterations, but the time it takes is $5 \times 10^4$. We get $J_{FW} = 0.1921$, $J_{Opt} = 0.0326$.

# 4 Things that do not work

Giving the forward problem with the constant velocity as initial guess AND as target for the OCP and ask it to do better. It immediately diverges. Maybe too advection dominant?
Considering problem 2 above with velocity strength 10 and interaction term. Diverges. Advection dominance?
In and outflow BCs. Diverges immediately. Correct implementation? Or maybe this restricts too much and we can't actually change the flow much...

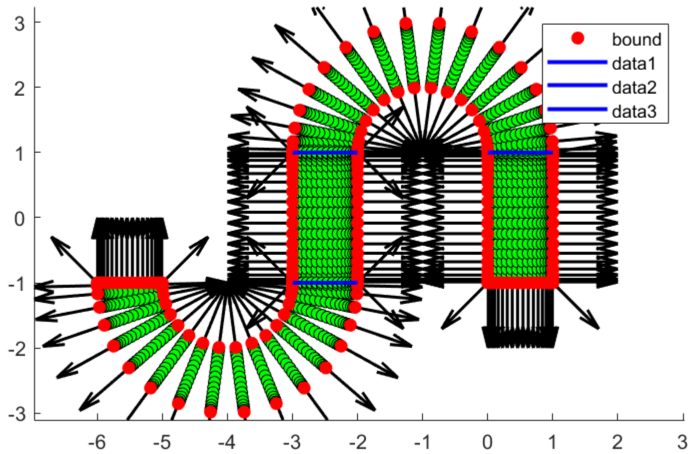


Figure 7: Domain

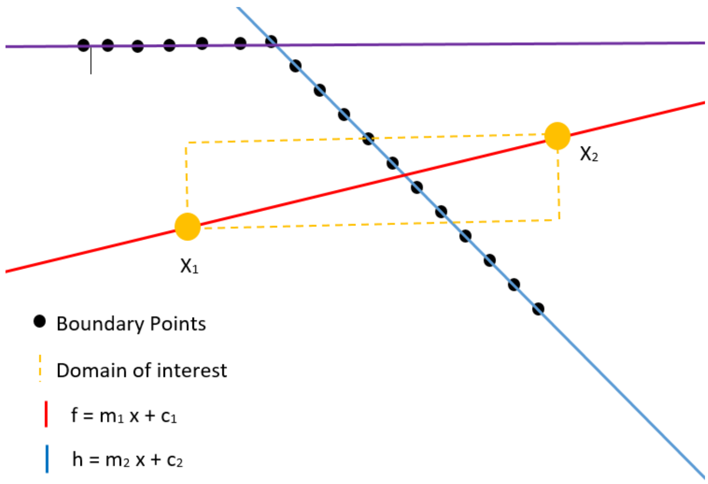


Figure 8: Intersection

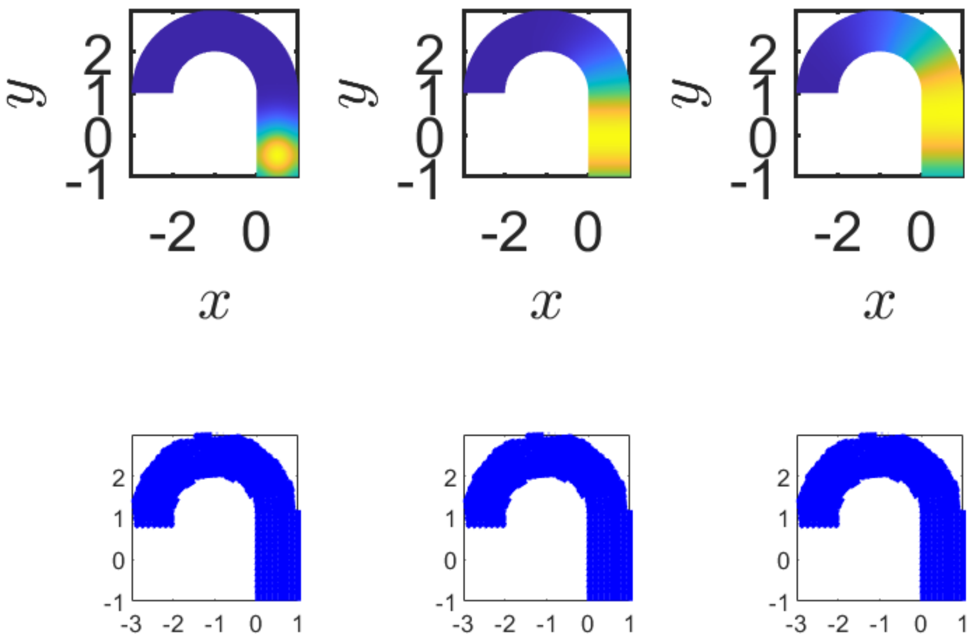


Figure 9: Test1 Forward $t = 1, 10, 19, n = 20$

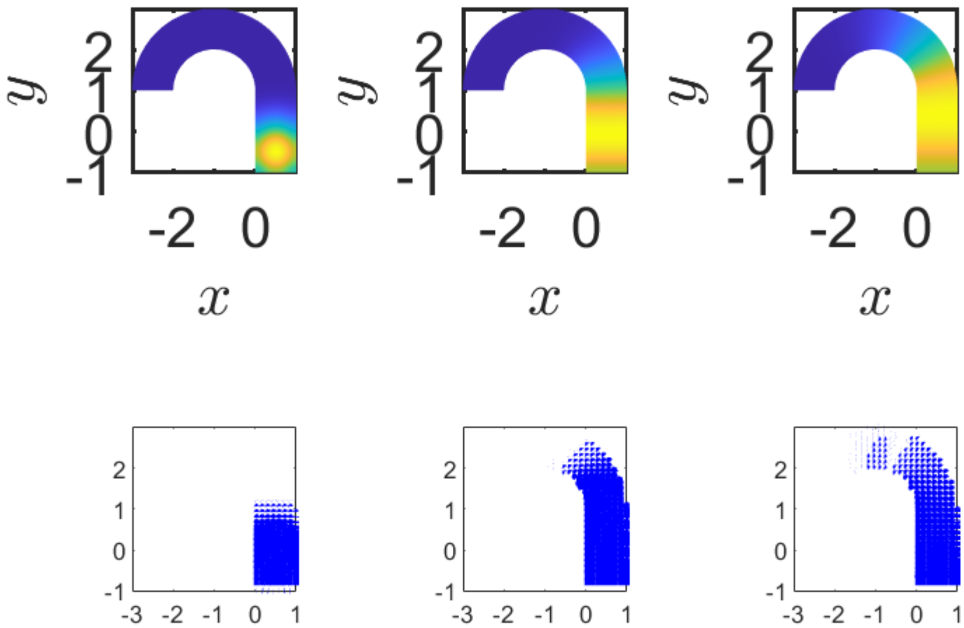


Figure 10: Test1 Optimization $t = 1, 10, 19, n = 20$

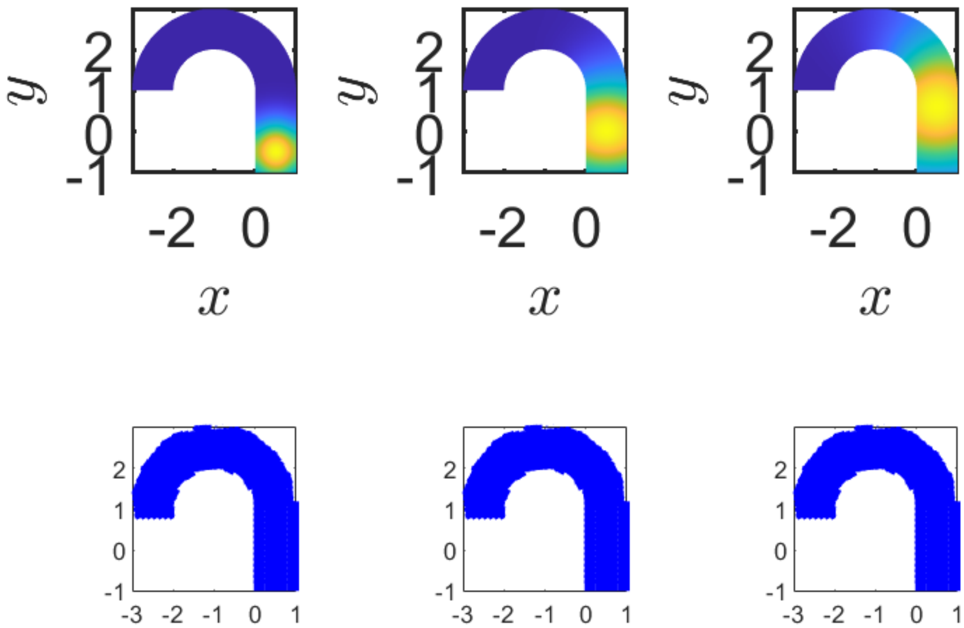


Figure 11: Test1 Forward $\kappa = -1$, $t = 1, 10, 19, n = 20$

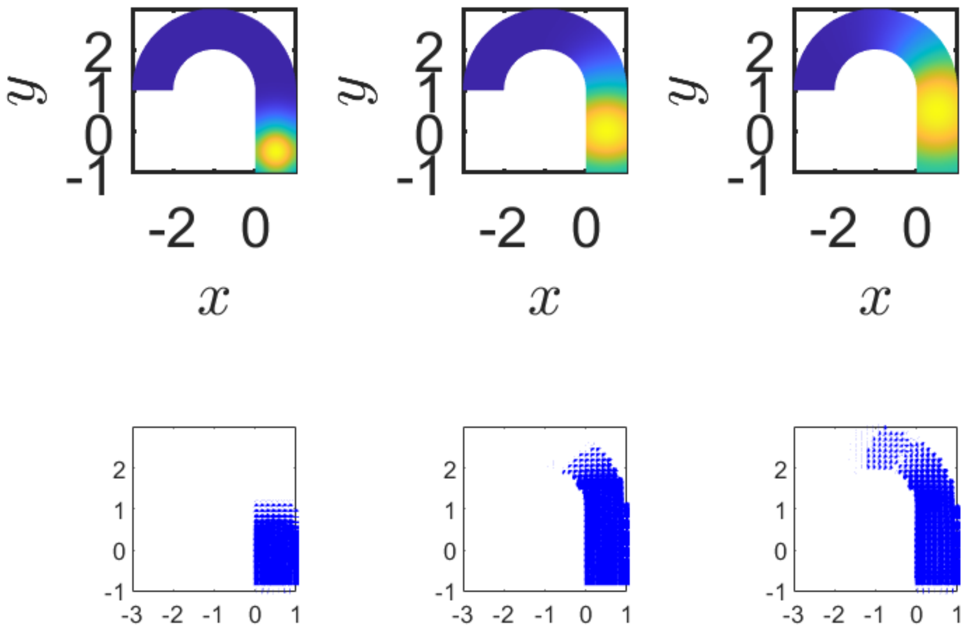


Figure 12: Test1 Optimization $\kappa = -1$, $t = 1, 10, 19, n = 20$

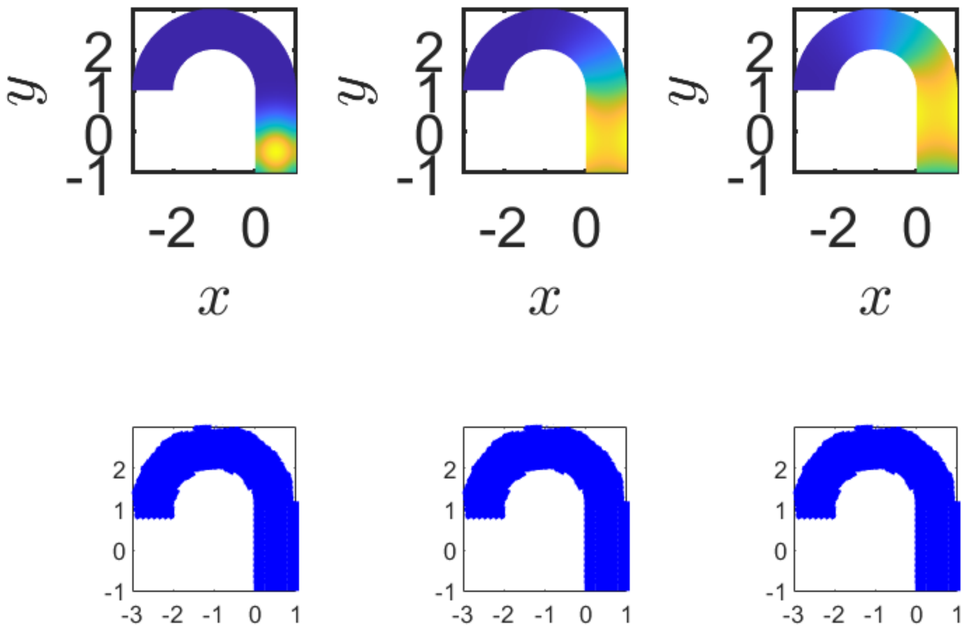


Figure 13: Test1 Forward $\kappa = 1$, $t = 1, 10, 19, n = 20$

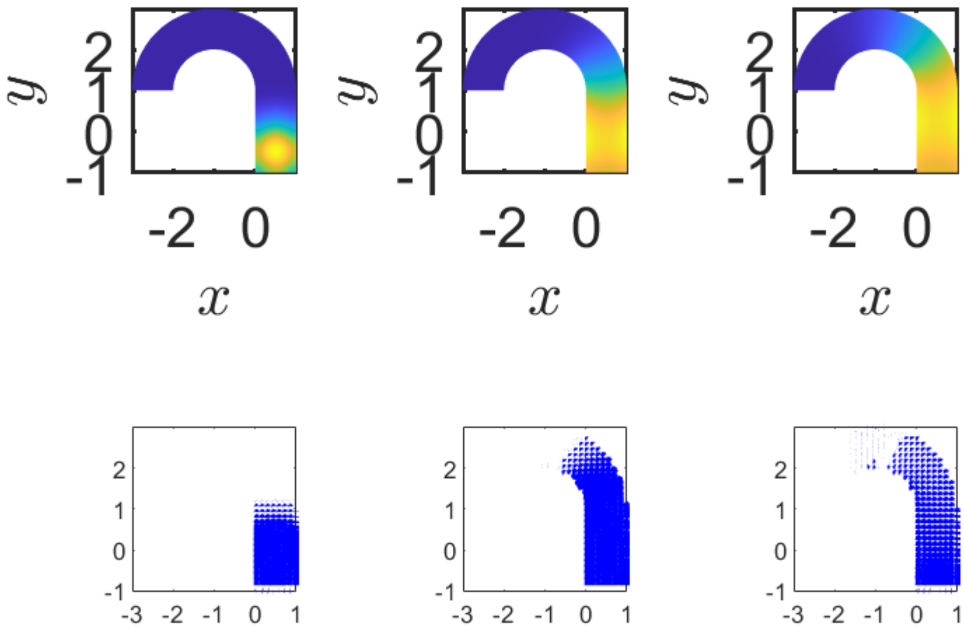


Figure 14: Test1 Optimization $\kappa = 1$ $t = 1, 10, 19, n = 20$

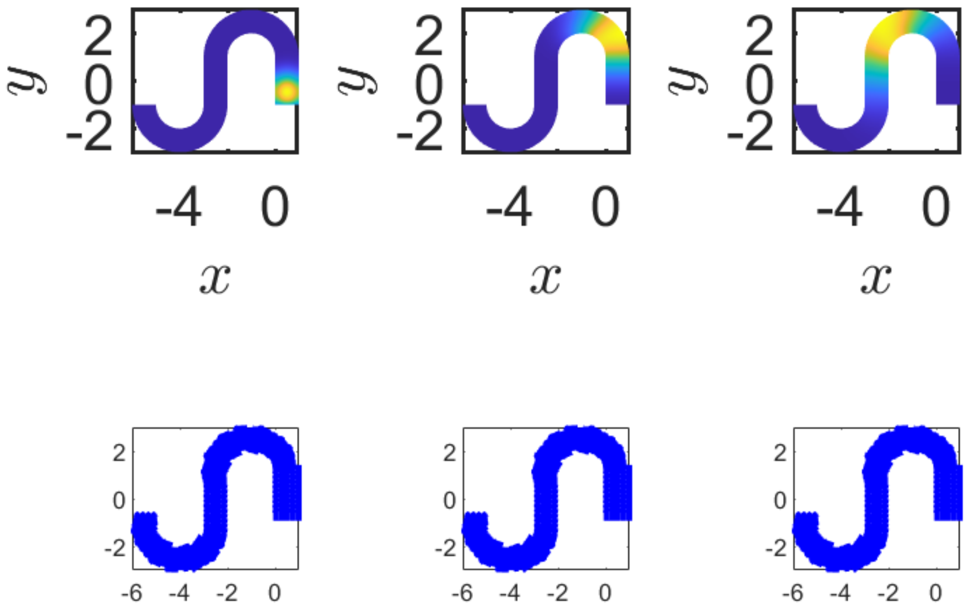


Figure 15: Test2 Forward $t = 1, 10, 19, n = 20$

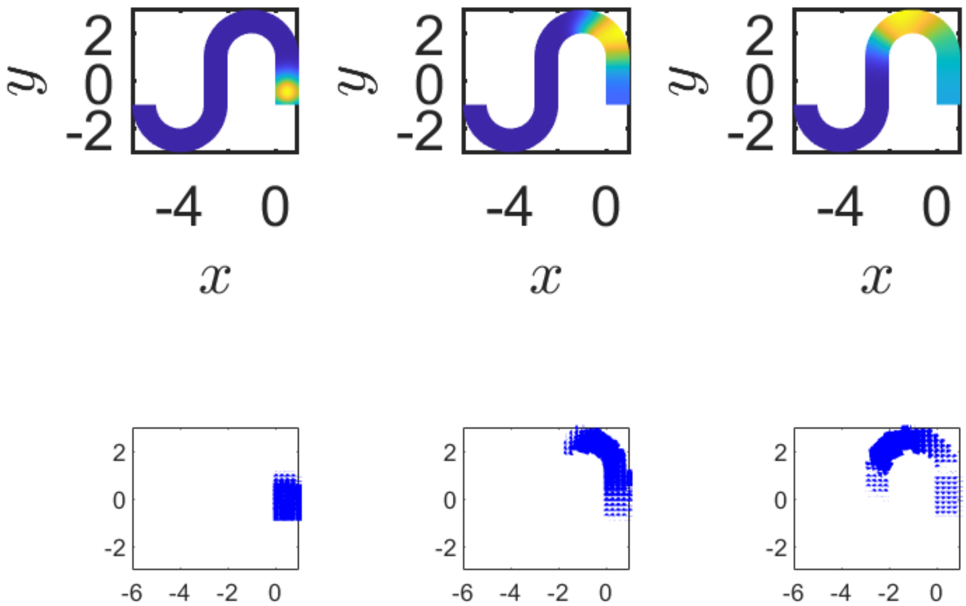


Figure 16: Test2 Optimization $t = 1, 10, 19, n = 20$