Contents

validation section
convergence plots
mass conservation

1 Model Equations

Different equations for different examples later?

2 Multishape Implementation

In the previous section we have discussed the implementation of the methods for different single shapes. While 2DChebClass supports solutions to PDEs on various single shapes, the method is now extended to compute the solution to a PDE on a multishape. A multishape is a complex domain, which is not fully described by one of the single shapes introduced in the previous section. However, it is possible to discretize the multishape domain in such a way that each of the elements is either a quadrilateral or a wedge. While that does not include all possible complex shapes, it does describe most physically relevant domains. The philosophy of this multishape code is to use the existing code library [1], which is designed to efficiently and accurately solve PDEs on individual shapes, to do the same on a multishape with minimal additional effort for the user.

The solution of a PDE on such a multishape domain is achieved by employing the spectral element method (SEM), since the multishape is discretized into elements. This method is similar in spirit to the finite element method (FEM). FEM discretizes a domain into elements and computes the solution to a given PDE on each of those elements and matches the solution at the boundaries. Expansions of basis functions are used, which are low order polynomials, for interpolation on an equispaced grid. SEM follows the same philosophy but uses higher order basis functions such as Chebyshev or Lagrange polynomials and Chebyshev-Lobatto points on an interpolation grid on each element, as opposed to an equispaced grid, to avoid the Runge phenomenon. At the intersections between the elements, C^0 continuity is enforced, by imposing two matching conditions, usually the solution and the first derivative, see [2]. SEM was first introduced by Patera [3] using Chebyshev polynomials as basis functions and later adapted to Lagrange polynomials by Komatitsch and Vilotte [4], which is now the standard choice [2]. While this method is widely used to solve PDEs in their weak form, in this work the strong form of the PDE is considered, since this aligns best with the existing framework. Furthermore, instead of matching the first derivative of the solution at the intersection of two elements, the flux is matched.

2.0.1 Setting up the multishape

In order to set up a multishape, each of the discretized elements have to be specified. The information that has to be given for each element is the number of discretization points, N_1 in the x-direction, N_2 in the y-direction, whether the element is a quadrilateral or a wedge and whether to match the internal boundaries between elements. Note that, in order to create a functioning multishape, the number of points of the neighbouring faces of two elements have to match. For a quadrilateral, the four coordinates of the edges have do be given. For a wedge, inner and outer radius, maximum and minimum angle as well as the origin of the wedge have to be given. As done throughout in the code, the main idea is to stack the vectors of the individual elements to result into a long vector of size $M \times 1$, where $M = \sum_i N_1^i N_2^i$, where i counts the number of elements in the multishape. An equivalent approach is taken for matrices. This stacking results in computations being done directly on the multishape, instead of individually on each element. (++ improve explanation ++)

Once this information is given, the vectors of computational points $\mathbf{x}_1^M$, $\mathbf{x}_2^M$ and physical points $\mathbf{y}_1^M$, $\mathbf{y}_2^M$ on the whole multishape can be set up. This is done as described in Algorithm 1. One important thing to notice is that the vectors $\mathbf{y}_1^M$, $\mathbf{y}_2^M$ are now defined in Cartesian coordinates, even if some of the underlying shapes are wedges, for which polar coordinates are the natural choice.

Algorithm 1: Multishape points

```
for ishape in multishape do
    • Get computational points $\mathbf{x}_1^i$, $\mathbf{x}_2^i$ and physical points $\mathbf{y}_1^i$, $\mathbf{y}_2^i$.
    • Add points to vector containing multishape points:
        $\mathbf{x}_1^M = [\mathbf{x}_1^M; \mathbf{x}_1^i], \quad \mathbf{x}_2^M = [\mathbf{x}_2^M; \mathbf{x}_2^i], \quad \mathbf{y}_1^M = [\mathbf{y}_1^M; \mathbf{y}_1^i], \quad \mathbf{y}_2^M = [\mathbf{y}_2^M; \mathbf{y}_2^i].$
end
for ishape in multishape do
    if ishape is polar then
        • Convert polar $\mathbf{y}_1^i$, $\mathbf{y}_2^i$ to cartesian $\mathbf{y}_{1,\text{Cart}}^i$, $\mathbf{y}_{2,\text{Cart}}^i$.
        • Replace in $\mathbf{y}_1^M$, $\mathbf{y}_2^M$:
            $\mathbf{y}_1^M(\text{ishape}) = \mathbf{y}_{1,\text{Cart}}^i, \quad \mathbf{y}_2^M(\text{ishape}) = \mathbf{y}_{2,\text{Cart}}^i.$
    end
end
```

2.0.2 Boundaries and intersections

In order to determine the points that lie on the boundary of a multishape, we first have to determine which faces of which shapes intersect. Once we have this information, we can take the boundaries of the individual shapes and substract the intersection boundaries from these to get the multishape boundary. The intersections between shapes are found by the code automatically, as explained in Algorithm 2. We iterate through all pairs of shapes to check whether any of their faces intersect by comparing the points of each shape on these faces. One thing to note is that we also have to check whether the points on face i are equal to the flipped vector of points on face j. This is to account for the fact that there are different ways of constructing these points on each shape. Having found the intersections between the shapes, the boundary of the multishape is defined by a boolean vector of size M, containing ones at the boundary and zeros everywhere else, as in the case for single shapes. Algorithm 3 explains the steps.

Algorithm 2: Determining intersections between shapes in a multishape

```
for ishape in multishape do
    for jshape in multishape do
        for iface in ishape do
            for jface in jshape do
                if $Pts_{iface} == Pts_{jface}$ then
                    Intersections(ishape,jshape).Pts = $\mathrm{Pts}_{\mathrm{iface}}$
                    Intersections(ishape,jshape).Face = iface
                    Intersections(ishape,jshape).Corners = $\mathrm{Corners}_{\mathrm{iface}}$
                    Intersections(ishape,jshape).Flip = False
                else if iface == flip(jface) then
                    Intersections(ishape,jshape).Pts = $\mathrm{Pts}_{\mathrm{iface}}$
                    Intersections(ishape,jshape).Face = iface
                    Intersections(ishape,jshape).Corners = $\mathrm{Corners}_{\mathrm{iface}}$
                    Intersections(ishape,jshape).Flip = True
                end
            end
        end
    end
end
```

Algorithm 3: Determining the boundary of a multishape

```
for ishape in multishape do
    for iface in ishape do
        for jface in ishape do
            if Intersections(ishape, jshape).Face = iface then
                Set IntersectionTest(iface) = True;
            end
        end
        if IntersectionTest(iface) = False then
            Set Boundary(iface) = True;
        end
    end
end
```

Constructing the normal vector for the multishape is quite complex and explained in Algorithm 4. One thing to be mindful of is the change from polar to Cartesian coordinates and back. The final normal vector contains polar values when corresponding to a wedge element and Cartesian coordinates for quadrilateral elements of the multishape. Extra care has to be taken at the corners of the boundary where two shapes intersect. There the normals are averaged as explained in Algorithm 4. However, when the discretization of the multishape is more complex, this may sometimes not be sufficient, and the resulting outward normal is not sensible. For this reason it is possible to override any of the normals manually as a user. The effect is demonstrated in the validation tests in Section +++ ref validation tests +++.

Algorithm 4: Determining the outward normals of a multishape

```
for ishape in multishape do
    • Get normal vector inormal for ishape.
    • Set normalVec(ishape) = inormal.
    if ishape is polar then
        • Get Cartesian normal vector inormalCart for ishape.
        • Set normalCart(ishape) = inormalCart.
    else
        • Set normalCart(ishape) = inormal.
    end
    • Delete entries that do not lie on the boundary of the multishape.
end
Fixing normals at corners. Find entries in normalCart that share same points (up to a
  tolerance). Store points in duplicates.
for dup in duplicates do
    if override is True then
        Normal vector $n$ at dup is specified by the user.
    else
        Average and normalize normals in normalCart(dup) to get $n$.
    end
end
Set normalCart(dup)= $n$.
for ishape in multishape do
    if ishape is polar then
        Convert normals from normalCart(ishape) to polar coordinates and assign to
          normal(ishape).
    else
        Set normal(ishape) = normalCart(ishape).
    end
end
```

2.0.3 Interpolation, differentiation, integration and convolution

The interpolation matrix for the multishape is constructed by computing the individual interpolation matrices on each shape and stacking them together in a blockdiagonal matrix. The gradient, divergence and Laplacian operators for the multishape are constructed in an equivalent way. The integration vector is constructed by simply stacking the integration vectors for each shape. Each of these constructions is demonstrated in Algorithm 5.

Algorithm 5: Constructing the interpolation matrix, gradient, divergence and Laplacian as well as the integration vector

```
for ishape in multishape do
    • Get $\text{Interp}_{\text{ishape}}$, $\text{Grad}_{\text{ishape}}$, $\text{Div}_{\text{ishape}}$, $\text{Lap}_{\text{ishape}}$, $\text{Int}_{\text{ishape}}$.
    • Set
      $\text{Interp} = \text{blkdiag}(\text{Interp}, \text{Interp}_{\text{ishape}})$,
      $\text{Grad} = \text{blkdiag}(\text{Interp}, \text{Grad}_{\text{ishape}})$,
      $\text{Div} = \text{blkdiag}(\text{Interp}, \text{Div}_{\text{ishape}})$,
      $\text{Lap} = \text{blkdiag}(\text{Interp}, \text{Lap}_{\text{ishape}})$,
      $\text{Int} = (\text{Int}, \text{Int}_{\text{ishape}})$.
end
```

While computing the standard interpolation matrix is straightforward, computing the interpolation matrix for a set of physical points in multishape is a bit more complex. Given a set of points $\mathbf{y}_1^I$ and $\mathbf{y}_2^I$ that we want to interpolate onto, we need to first determine which of these points are in which shape, since the interpolation is done shapewise as explained in Algorithm

5. Algorithm 6 illustrates this. (+++ check algorithm after meeting +++)

Algorithm 6: Constructing the interpolation matrix for interpolating onto points in physical space

```
for ishape in multishape do
    Get $\mathbf{y}_1^I$ and $\mathbf{y}_2^I$ to interpolate onto;
    for ishape in multishape do
        • Get computational points $x_1$, $x_2$ for ishape.
        • Get $x_{1,\text{min}} = \min(x_1) - 10^{-10}$, $x_{1,\text{max}} = \max(x_1) + 10^{-10}$, $x_{2,\text{min}} = \min(x_2) - 10^{-10}$,
          $x_{2,\text{max}} = \max(x_2) + 10^{-10}$.
        if ishape is polar then
            Get polar version of $\mathbf{y}_1^I$ and $\mathbf{y}_2^I$.
        end
        • Transform $\mathbf{y}_1^I$ and $\mathbf{y}_2^I$ to $\mathbf{x}_1^I$ and $\mathbf{x}_2^I$ in computational space, using the linear mapping
          associated to the shape.
        • Define iMask = $(x_1 >= x_{1,min})$ & $(x_1 <= x_{1,max})$ & $(x_2 >= x_{2,min})$ & $(x_2 <= x_{2,max})$ &
          doneMask.
        • Create new vectors that only contain the points of the shape:
          $\mathbf{x}_1^I$(iMask), $\mathbf{x}_2^I$(iMask).
        for i in length($\mathbf{x}_1^I$(ishape)) do
            • Compute interpolation matrix pointwise for $x_1(i) \in \mathbf{x}_1^I$(ishape), $x_2(i) \in \mathbf{x}_1^I$(ishape).
            • Set Interp$_{\text{ishape}}$ = [ Interp$_{\text{ishape}}$, InterpolationMatrix$(x_1(i), x_2(i))$].
        end
        • Stack interpolation matrices in a blockdiagonal:
          Interp = blkdiag(Interp, Interp$_{\text{ishape}}$).
        • Set doneMask(iMask) = True.
    end
    • Set Interp( doneMask) = [].
    • Set $\mathbf{y}_1^I = \mathbf{y}_1^I$(doneMask), $\mathbf{y}_2^I = \mathbf{y}_2^I$(doneMask).
end
```

The convolution matrix cannot be taken from the individual shapes, since convolution is a global operation. We compute it in the exact same way as for a single quadrilateral, see Section **??**, now using the multishape points $\mathbf{y}_1^M$ and $\mathbf{y}_2^M$ and the integration vector that was constructed for the multishape.

2.0.4 Boundary matching and solving the PDE

As discussed above, the code automatically identifies the intersection boundaries between two shapes when setting up the multishape. Once the intersections between the neighboring shapes are identified, user-defined boundary conditions can be applied. There are currently two options, although the addition of further boundary conditions is straightforward. In general, both the solution to the PDE and the flux are matched at these intersection boundaries to create a coherent solution over the whole shape. Alternatively, hard walls between two shapes can be simulated easily, by applying a no-flux boundary condition at that intersection boundary. On boundaries which are on the outside of the multishape, the boundary conditions of the PDE, such as no-flux and Dirichlet conditions, can be applied in the same way as for single shapes. The application of the boundary conditions and solution of the PDE follows the method for single shapes and is illustrated in Algorithm 7. Note that the mass matrix for a collocation method is diagonal. The boundary and matching conditions are applied by setting the relevant entries in the mass matrix to zero.

Algorithm 7: Applying boundary and intersection conditions, solving the PDE.

- Get discretized PDE dfdt.
- Get boundary condition equation BCeq and intersection boundary condition Intereq.
- Set the mass matrix M = diag(ones).
- Set M(boundary) = 0, M(intersection) = 0.
- Set dfdt(boundary) = BCeq.
- Set dfdt(intersection)= Intereq.
- Compute M dfdt using `ode15s`.

2.0.5 Exact Tests

The solution to known testproblems with Dirichlet and no-flux boundary conditions are considered in this section. The errors are calculated as an l_2 error in space and an l_∞ error in time.

Exact Tests - Dirichlet Conditions

(MS_TestJonnaADExactDisectBoxes and MS_TestJonnaADExactDisectBoxesPart2 and MS_TestJonnaADExactDisectWedges) Several examples are run, using exact solutions, to validate the multishape code. This is done using an exact solution to the advection diffusion equation on an infinite domain, so that Dirichlet boundary conditions can be applied, by matching the value of the exact solution on the boundary of the multishape. The exact solution is [5]

$$\begin{aligned}\rho &= \exp(\alpha t + \beta_1 y_1 + \beta_2 y_2) \qquad (1)\\ \mathbf{v} &= \left(\beta_1 - \frac{\alpha}{2\beta_1} + p_1 \exp(-\beta_1 y_1), \beta_2 - \frac{\alpha}{2\beta_2} + p_2 \exp(-\beta_2 y_2)\right),\end{aligned}$$

where $\beta_1 = 0.1$, $\beta_2 = 0.1$, $\alpha = -0.5$, $p_1 = -1$ and $p_2 = 1$. We compare the exact solution on a box of dimensions $[0,2] \times [0,2]$ with different discretizations of the box using multishape, signified by letters (a) to (g), see Figure **??**. Each of the shapes are discretized with $N = 10$, $N = 20$ and $N = 30$ points in each spatial direction, which means that the dissected box has more points in total than the original box. The ODE solver tolerances are 10^{-9}. The solution can be seen in Figure 1. The question is whether the results of the PDE on the box and the different discretizations of the box have a similar error when compared to the exact solution. The absolute and relative errors, $\mathcal{E}_{Abs}$ and $\mathcal{E}_{Rel}$, are measured in an l_2 norm in space and an l_∞ norm in time and are displayed in Table 1. The errors for the non-rectangular quadrilateral, which is shown in Figure **??**, are displayed on the row with the letter (q).
The same test can be done for a wedge. Here, a single wedge and discretized versions are considered, denoted by (h) to (k), see Figure **??**. Table 1 also shows the errors measured against the exact solution for different discretizations of the wedge, for $N = 10$, $N = 20$ and $N = 30$. The solution can be seen in Figure 2.
Next the advection diffusion equation is solved on a multishape which is composed of four quadrilaterals, see Figure 3. The same is done for a second example involving a wedge, see Figure 4. The errors, as compared to the exact solution (1), are displayed in Table 2. There, the first multishape is denoted ms1 and the second ms2.

Exact Tests - Dirichlet Conditions, Solution 2

We solve an exact Dirichlet problem which does not have an exponential form but a quadratic form. In order to do this, we add on a source term f to the advection-diffusion equation. We define the exact solution as

$$\begin{aligned}\rho &= t y_1^2 y_2^2,\\ f &= y_1^2 y_2^2 - 2y_1^2 t - 2y_2^2 t + 2y_1 y_2^2 t,\end{aligned}$$

	$N = 10$		$N = 20$		$N = 30$	
	$\mathcal{E}_{Abs}$	$\mathcal{E}_{Rel}$	$\mathcal{E}_{Abs}$	$\mathcal{E}_{Rel}$	$\mathcal{E}_{Abs}$	$\mathcal{E}_{Rel}$
a	2.5869×10^{-7}	1.0894×10^{-9}	2.2063×10^{-7}	1.1235×10^{-9}	2.1913×10^{-7}	1.1159×10^{-9}
b	3.4991×10^{-7}	1.1308×10^{-9}	3.2073×10^{-7}	1.1377×10^{-9}	3.1877×10^{-7}	1.1307×10^{-9}
c	4.1386×10^{-7}	1.1596×10^{-9}	3.9107×10^{-7}	1.1500×10^{-9}	3.8858×10^{-7}	1.1426×10^{-9}
d	4.2622×10^{-7}	1.0268×10^{-9}	3.9019×10^{-7}	1.0026×10^{-9}	3.8768×10^{-7}	9.9616×10^{-10}
e	4.4595×10^{-7}	1.0704×10^{-9}	3.3852×10^{-7}	9.8259×10^{-10}	3.3718×10^{-7}	1.0085×10^{-9}
f	4.1985×10^{-7}	1.0275×10^{-9}	4.2547×10^{-7}	1.0412×10^{-9}	4.2291×10^{-7}	1.0350×10^{-9}
g	5.0161×10^{-7}	1.1254×10^{-9}	4.4470×10^{-7}	1.1323×10^{-9}	4.3978×10^{-7}	1.1198×10^{-9}
q	2.4145×10^{-7}	1.0389×10^{-9}	2.2844×10^{-7}	1.1160×10^{-9}	2.2505×10^{-7}	1.0995×10^{-9}
h	1.8507×10^{-3}	4.4410×10^{-6}	3.1141×10^{-7}	7.4763×10^{-10}	2.7613×10^{-7}	7.5178×10^{-10}
i	3.4678×10^{-6}	6.5183×10^{-9}	3.9525×10^{-7}	7.5983×10^{-10}	3.8485×10^{-7}	7.4376×10^{-10}
j	2.6220×10^{-3}	4.4489×10^{-6}	4.2335×10^{-7}	7.4943×10^{-10}	3.8922×10^{-7}	7.5064×10^{-10}
k	2.6814×10^{-6}	4.0877×10^{-9}	4.4144×10^{-7}	7.0896×10^{-10}	4.3211×10^{-7}	6.9682×10^{-10}

Table 1: Table Dirichlet Exact Solution 1

	$N = 10$		$N = 20$		$N = 30$	
	$\mathcal{E}_{Abs}$	$\mathcal{E}_{Rel}$	$\mathcal{E}_{Abs}$	$\mathcal{E}_{Rel}$	$\mathcal{E}_{Abs}$	$\mathcal{E}_{Rel}$
ms1	5.9588×10^{-7}	1.3030×10^{-9}	6.1246×10^{-7}	1.3259×10^{-9}	6.0034×10^{-7}	1.2997×10^{-9}
ms2	1.6829×10^{-3}	3.6591×10^{-6}	3.7239×10^{-7}	8.9375×10^{-10}	3.7589×10^{-7}	8.9532×10^{-10}

Table 2: Table Dirichlet Exact Solution 1 on two multishapes

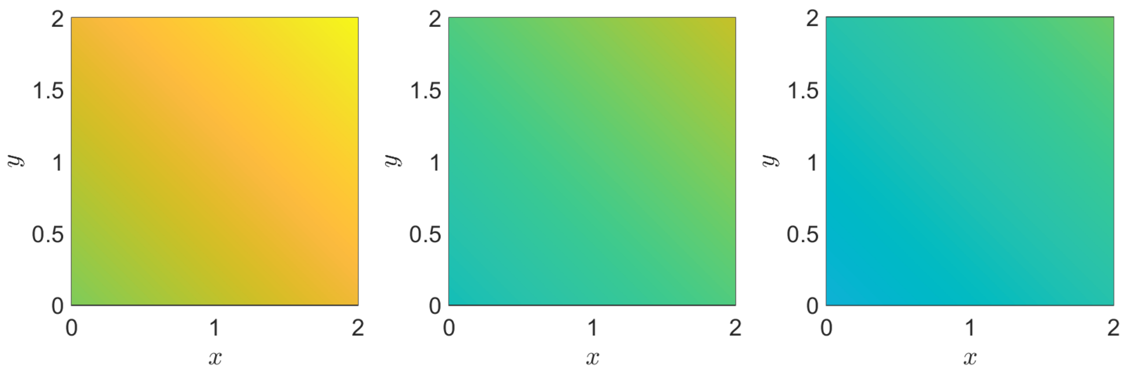

Figure 1: Exact solution on the box.

and with a velocity field of strength one acting in the y_1 direction. We run this on the box and the wedge discretizations (a) to (g) and the quadrilateral (q). The results are displayed in Table 3.

Exact Tests - No-Flux Conditions

We consider an exact solution on a box with no-flux boundary conditions for the advection diffusion equation. The solution is

$$\rho = 2 + \exp(-(\mu_1^2 + \mu_2^2)t)\cos(\mu_1 y_1)\cos(\mu_2 y_2),$$

where $\mu_1 = n\pi/L_1$, $\mu_2 = n\pi/L_2$ and $n = 2$, $L_1 = d - c$, $L_2 = b - a$ for a domain $[a, b] \times [c, d]$. We use the discretizations of the box displayed in Figure **??** with $N = 10$, $N = 20$ and $N = 30$. The exact solution for this problem can be seen in Figure 5. In Table 4 we can see the errors made on the different discretizations. We can observe that the errors in multishapes (c), (d) and (f) are considerably larger than the errors for the other shapes. This is because of the complex discretization chosen in these multishapes. Due to this discretization, the standard averaging of the two normals at an intersection corner does not result in reasonable normals. This becomes an issue when computing no-flux boundary conditions, since this requires the normal information. The code library offers the option of overriding individual normals. We redefine the normals at the intersection corner to be the outward normals of the box. This

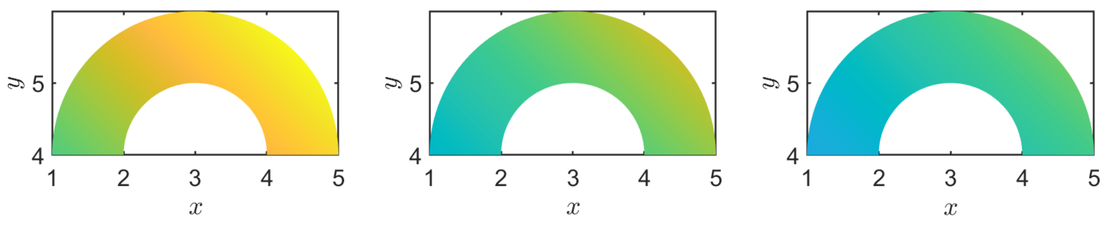

Figure 2: Exact solution on the wedge.

improves the errors greatly as can be seen in Table 5. The change in normals for these three multishapes is shown in Figure 6.

Forward Problems on multishapes

We first consider a forward problem on the different discretizations of the box, see Figure **??**. We compare the results of the discretized boxes with the result for box (a) with large N. We choose the initial condition for ρ to be

$$\rho_0 = \exp(-2((y_1 - 0.7)^2 + (y_2 - 0.2)^2)),$$

and impose a constant flow of strength 0.8 acting upward. We choose $N = 10$, $N = 20$ and $N = 30$ as before. The errors are displayed in Table 6 and the result can be seen in Figure 7.
We follow the same idea, but now consider the wedge discretizations, see Figure **??**. We choose the initial condition for ρ to be

$$\rho_0 = \exp(-2((y_1 - 1.5)^2 + (y_2 - 4.5)^2)),$$

and impose a constant flow of strength 3 acting from left to right, along the angular direction. We choose $N = 20$ and $N = 30$. The errors are displayed in Table 7 and the result can be seen in Figure 8.
Finally, two more complex multishape examples are considered. The first of these examples is solving an advection diffusion problem on a multishape consisting of two quadrilaterals and two

	$N = 10$		$N = 20$		$N = 30$	
	$\mathcal{E}_{Abs}$	$\mathcal{E}_{Rel}$	$\mathcal{E}_{Abs}$	$\mathcal{E}_{Rel}$	$\mathcal{E}_{Abs}$	$\mathcal{E}_{Rel}$
a	2.2747×10^{-13}	3.9799×10^{-16}	3.9207×10^{-13}	6.3023×10^{-16}	4.7291×10^{-13}	7.5978×10^{-16}
b	1.1317×10^{-12}	1.1097×10^{-15}	6.1443×10^{-13}	5.9848×10^{-16}	2.5646×10^{-12}	2.5682×10^{-15}
c	7.0072×10^{-13}	7.0033×10^{-16}	1.4877×10^{-12}	1.5185×10^{-15}	2.0691×10^{-12}	2.4102×10^{-15}
d	6.8357×10^{-13}	5.1274×10^{-16}	1.4899×10^{-12}	2.9801×10^{-15}	2.6521×10^{-12}	2.0667×10^{-15}
e	2.2548×10^{-12}	6.7534×10^{-15}	5.5358×10^{-13}	6.3464×10^{-16}	8.8186×10^{-13}	2.3179×10^{-15}
f	8.1603×10^{-13}	1.5216×10^{-15}	1.4019×10^{-12}	8.7680×10^{-16}	3.3732×10^{-12}	1.8904×10^{-15}
g	4.9611×10^{-13}	1.9913×10^{-15}	5.0195×10^{-12}	8.5065×10^{-15}	1.4727×10^{-12}	1.1481×10^{-15}
q	6.0563×10^{-13}	1.7871×10^{-15}	9.9468×10^{-13}	4.3977×10^{-15}	1.1245×10^{-12}	3.2386×10^{-15}
h	$9.2590 \times 10^{+00}$	1.5820×10^{-4}	3.6221×10^{-6}	6.1886×10^{-11}	5.9042×10^{-11}	1.0457×10^{-15}
i	2.6170×10^{-2}	3.1608×10^{-7}	5.9610×10^{-11}	7.4688×10^{-16}	8.2340×10^{-11}	1.0439×10^{-15}
j	$1.3082 \times 10^{+01}$	1.5808×10^{-4}	5.1279×10^{-6}	6.1963×10^{-11}	2.1720×10^{-10}	2.7449×10^{-15}
k	1.9298×10^{-2}	2.0145×10^{-7}	6.8410×10^{-11}	7.4568×10^{-16}	9.9529×10^{-11}	1.0842×10^{-15}

Table 3: Table Dirichlet Exact Solution 2

	$N = 10$		$N = 20$		$N = 30$	
	$\mathcal{E}_{Abs}$	$\mathcal{E}_{Rel}$	$\mathcal{E}_{Abs}$	$\mathcal{E}_{Rel}$	$\mathcal{E}_{Abs}$	$\mathcal{E}_{Rel}$
a	1.0458×10^{-2}	2.5230×10^{-5}	2.8350×10^{-7}	6.8623×10^{-10}	2.6819×10^{-7}	6.4918×10^{-10}
b	1.0178×10^{-2}	1.7387×10^{-5}	3.5488×10^{-7}	6.0816×10^{-10}	3.5538×10^{-7}	6.0901×10^{-10}
c	7.1323×10^{-2}	9.8994×10^{-5}	7.0449×10^{-3}	9.7842×10^{-6}	2.6254×10^{-3}	3.6463×10^{-6}
d	8.4906×10^{-2}	1.0217×10^{-4}	8.2570×10^{-3}	9.9356×10^{-6}	3.0338×10^{-3}	3.6506×10^{-6}
e	5.6960×10^{-3}	8.1264×10^{-6}	4.3060×10^{-7}	6.0669×10^{-10}	4.3142×10^{-7}	6.0784×10^{-10}
f	1.2934×10^{-1}	1.5474×10^{-4}	1.3465×10^{-2}	1.6099×10^{-5}	4.0715×10^{-3}	4.8753×10^{-6}
g	8.7462×10^{-6}	1.0753×10^{-8}	5.1501×10^{-7}	6.2332×10^{-10}	5.1469×10^{-7}	6.2294×10^{-10}

Table 4: Table No-Flux Exact Solution

	$N = 10$		$N = 20$		$N = 30$	
	$\mathcal{E}_{Abs}$	$\mathcal{E}_{Rel}$	$\mathcal{E}_{Abs}$	$\mathcal{E}_{Rel}$	$\mathcal{E}_{Abs}$	$\mathcal{E}_{Rel}$
c	1.8668×10^{-2}	2.5911×10^{-5}	4.5961×10^{-7}	6.4023×10^{-10}	4.5881×10^{-7}	6.3911×10^{-10}
d	2.4277×10^{-2}	2.9194×10^{-5}	5.2874×10^{-7}	6.3810×10^{-10}	5.2815×10^{-7}	6.3740×10^{-10}
f	2.1887×10^{-3}	2.6634×10^{-6}	5.3703×10^{-7}	6.3964×10^{-10}	5.3681×10^{-7}	6.3938×10^{-10}

Table 5: Table No-Flux Exact Solution with Corrected Normal Vectors at Intersections

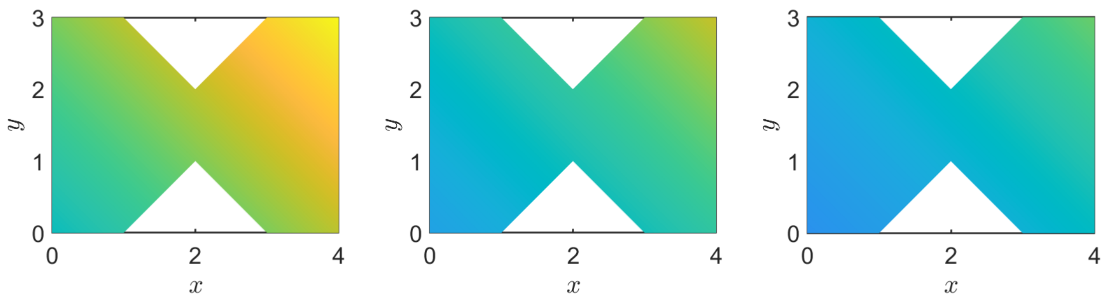

Figure 3: Example 1 multishape

wedges, with constant velocity of strength ten. The initial condition for this problem is:

$$\rho_0 = \exp(-2(y_1 - 0.5)^2 - 2(y_2 + 1)^2).$$

The result, evaluated for $N = 20$ on each shape, can be seen in Figure 9.
In a second example, the velocity is of strength 5 and the initial condition is:

$$\rho_0 = \exp(-2(y_1 - 0.5)^2 - 2(y_2 - 1.5)^2).$$

The result, which is computed on a multishape made up of four quadrilaterals into a channel, can be seen in Figure 10.
In Table 8 the solution to these two examples is evaluated with $N = 30$ and compared to the solution evaluated with $N = 10$, $N = 15$ and $N = 20$, using the same error measure as before. We can see that the solution converges with the number of points. It is especially noticeable for Example 1, that $N = 10$ does not give a reliable solution at all, while $N = 15$ provides a solution that is reasonably close to the one with $N = 30$. The reason that Example 1 is much more sensitive to the number of points than Example 2 could be that it contains polar shapes. As discussed in Section **??**, for accurate interpolation of the wedge, the angular direction needs more points than the radial direction, and in particular was not very accurate with $N = 10$ in the angular direction.

	$N=10$		$N=20$		$N=30$	
	$\mathcal{E}_{Abs}$	$\mathcal{E}_{Rel}$	$\mathcal{E}_{Abs}$	$\mathcal{E}_{Rel}$	$\mathcal{E}_{Abs}$	$\mathcal{E}_{Rel}$
b	6.8019×10^{-2}	1.7873×10^{-2}	1.9441×10^{-2}	2.4954×10^{-3}	7.8355×10^{-3}	6.6545×10^{-4}
c	4.5117×10^{-2}	6.9628×10^{-3}	1.2944×10^{-2}	9.9990×10^{-4}	5.3658×10^{-3}	2.7644×10^{-4}
d	1.1864×10^{-1}	1.7013×10^{-2}	3.3678×10^{-2}	2.3638×10^{-3}	1.3708×10^{-2}	6.3705×10^{-4}
e	1.2516×10^{-1}	1.9815×10^{-2}	3.1984×10^{-2}	2.5133×10^{-3}	1.3367×10^{-2}	6.9858×10^{-4}
f	5.8413×10^{-2}	1.4025×10^{-2}	1.6467×10^{-2}	1.9292×10^{-3}	6.3541×10^{-3}	4.9237×10^{-4}
g	4.0532×10^{-2}	5.7827×10^{-3}	9.9059×10^{-3}	7.0226×10^{-4}	2.5623×10^{-3}	1.2085×10^{-4}

Table 6: Table Forward Problem on Box

	$N=10$		$N=20$		$N=30$	
	$\mathcal{E}_{Abs}$	$\mathcal{E}_{Rel}$	$\mathcal{E}_{Abs}$	$\mathcal{E}_{Rel}$	$\mathcal{E}_{Abs}$	$\mathcal{E}_{Rel}$
i	8.1092×10^{-2}	1.8792×10^{-2}	2.3281×10^{-2}	2.6395×10^{-3}	7.0628×10^{-3}	5.3006×10^{-4}
j	2.8569×10^{-1}	5.2137×10^{-2}	7.5030×10^{-2}	6.8191×10^{-3}	3.1628×10^{-2}	1.9130×10^{-3}
k	7.0422×10^{-2}	9.7166×10^{-3}	1.8884×10^{-2}	1.2887×10^{-3}	1.1284×10^{-2}	5.1154×10^{-4}

Table 7: Table Forward Problem on Wedge

	$N=10$		$N=15$		$N=20$	
	$\mathcal{E}_{Abs}$	$\mathcal{E}_{Rel}$	$\mathcal{E}_{Abs}$	$\mathcal{E}_{Rel}$	$\mathcal{E}_{Abs}$	$\mathcal{E}_{Rel}$
Ex1	$1.7380 \times 10^{+121}$	$1.3801 \times 10^{+121}$	2.9942×10^{-1}	5.0366×10^{-2}	1.5166×10^{-1}	1.7141×10^{-2}
Ex2	3.1819×10^{-2}	8.9836×10^{-3}	1.4655×10^{-2}	2.2034×10^{-3}	9.7279×10^{-3}	1.0914×10^{-3}

Table 8: Table Forward Examples 1 and 2

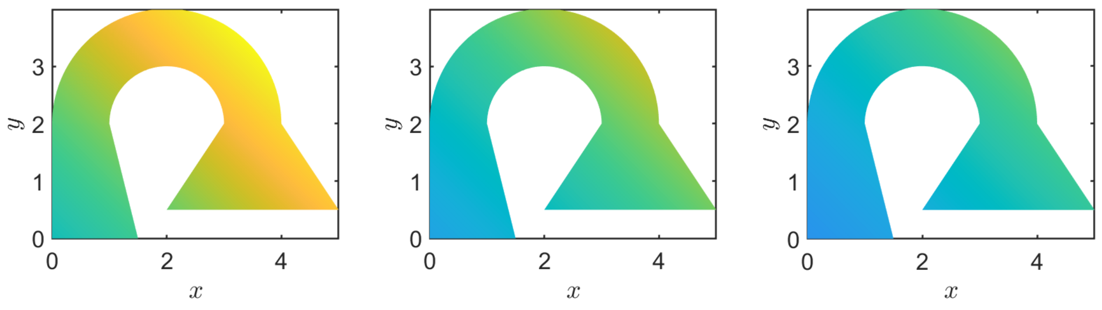

Figure 4: Example 2 multishape

References

[1] Andreas Nold, Benjamin D. Goddard, Peter Yatsyshin, Nikos Savva, and Serafim Kalliadasis. Pseudospectral methods for density functional theory in bounded and unbounded domains. *CoRR*, abs/1701.06182, 2017.

[2] John P. Boyd. *Chebyshev and Fourier Spectral Methods.* Dover Publications, Inc, 2000.

[3] Anthony T Patera. A spectral element method for fluid dynamics: Laminar flow in a channel expansion. *Journal of Computational Physics*, 54(3):468–488, 1984.

[4] Dimitri Komatitsch and Jean-Pierre Vilotte. The spectral element method: an efficient tool to simulate the seismic response of 2d and 3d geological structures. *Bulletin of the Seismological Society of America*, 88:368–392, 04 1998.

[5] G D Hutomo, J Kusuma, A Ribal, A G Mahie, and N Aris. Numerical solution of 2-d advection-diffusion equation with variable coefficient using du-fort frankel method. *Journal of Physics: Conference Series*, 1180:012009, feb 2019.

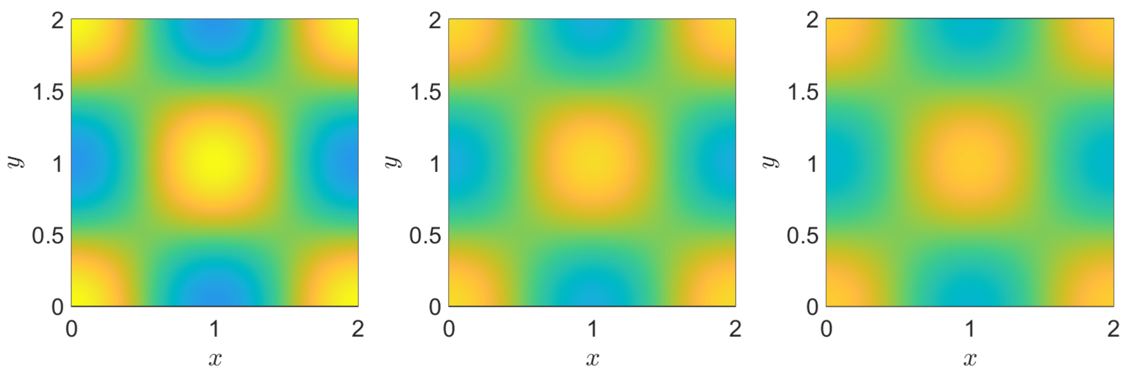

Figure 5: Exact solution on the box with no-flux boundary conditions.

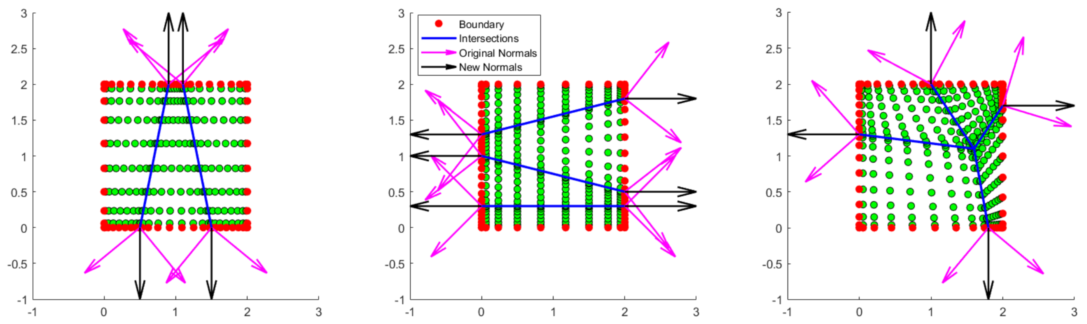

Figure 6: Original, non-sensible normals are compared to new, customized normals.

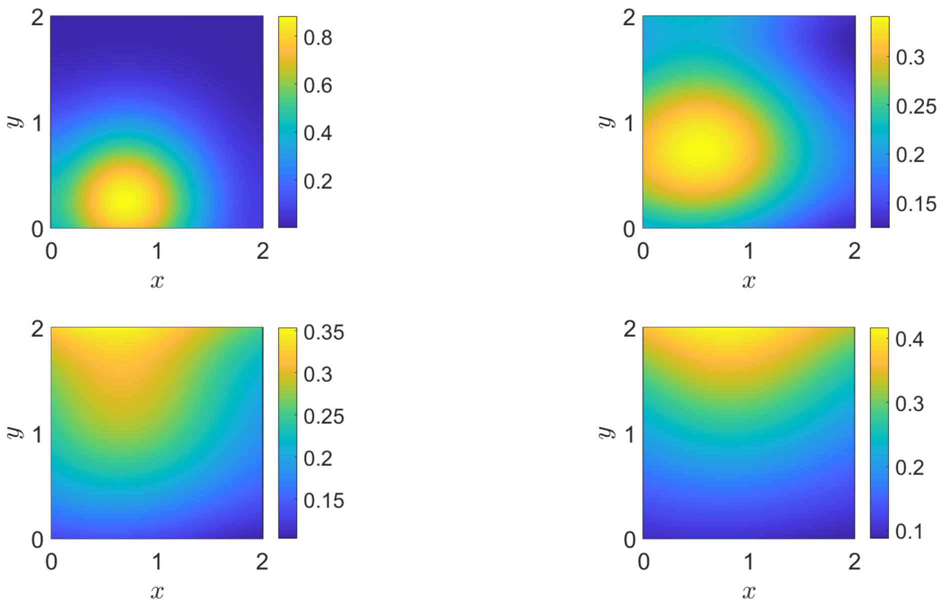

Figure 7: Forward Example on box discretizations

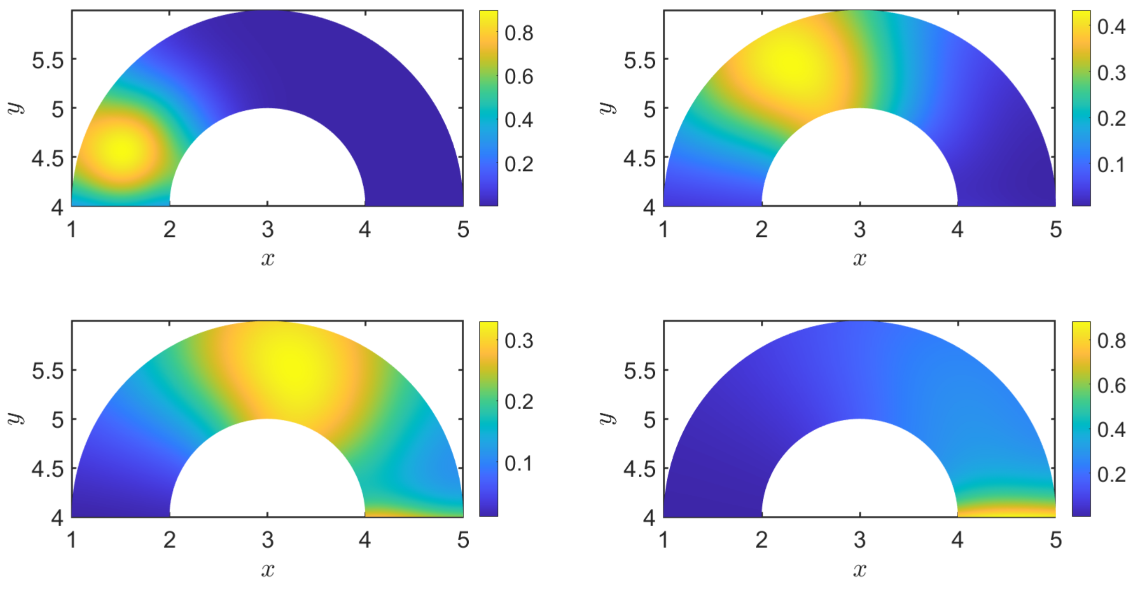

Figure 8: Forward Example on wedge discretizations

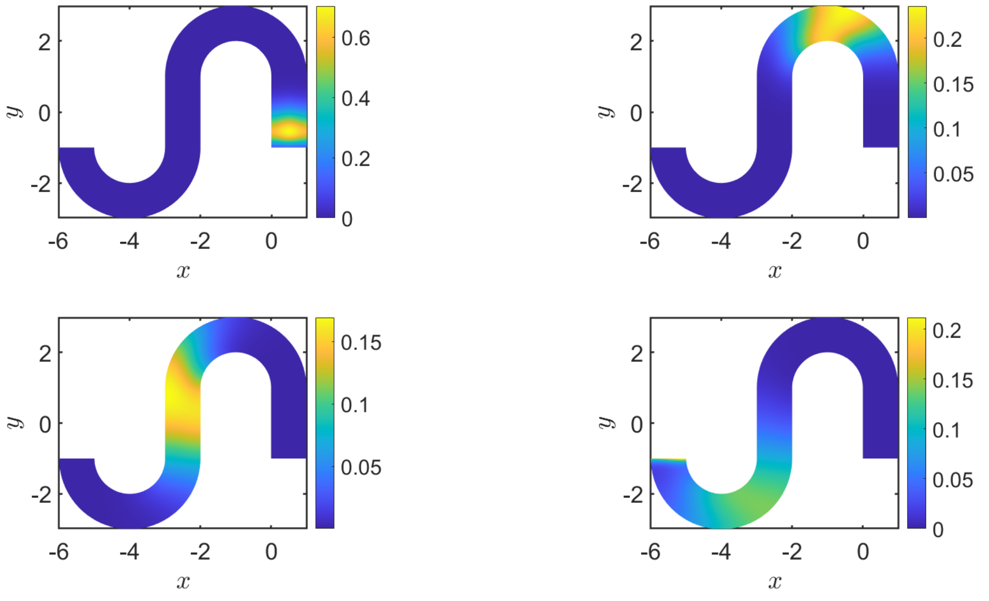
Figure 9: Forward Problem 1, different colour scale for each plot to highlight particle mass location

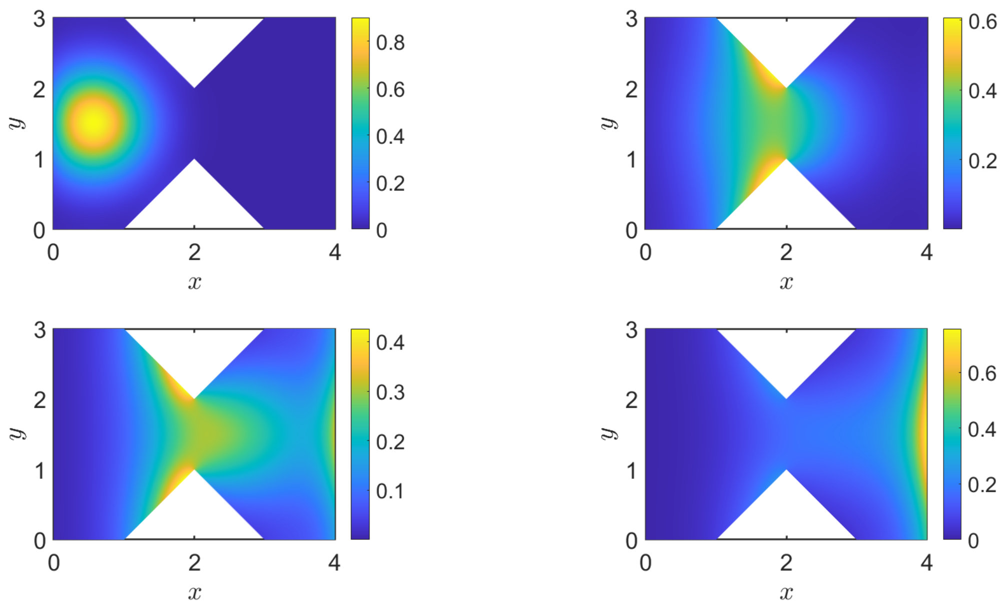
Figure 10: Forward Problem 2, different colour scale for each plot to highlight particle mass location