Numerical PDEs Homework #1

In this mini-project assignment, we'll learn to work with **Immersed Boundary Methods (IBM)**, though technically we'll be using the **Force Coupling Method (FCM)** – the only difference is the kernel. You'll modify the 2D Poisson code from HW0 to account for an internal Dirichelet boundary condition. To do this, we'll implement a **Matrix-Free** linear solver with GMRES to solve the **Saddle Point System** that shows up.

Problem 1: Poisson Equation

First let's make sure our code from HW0 still works. Use a finite difference method to solve the Poisson equation $\nabla^2 u = f(x,y)$ in the domain $x \in [0, 2\pi], y \in [0, 2\pi]$ with Dirichelet boundary conditions on all sides. Choose f and the boundary data so that the test solution $u(x,y) = \cos(2x)\sin(2y)$ solves the problem. Then verify that your code converges with second order in the grid size Δx.

Problem 2: Poisson Equation with an Immersed Boundary

Now we'll consider the same setup in Problem 1, but with the additional Dirichelet condition:

$$u(x^{ib}, y^{ib}) = \cos\big(2x^{ib}\big)\sin\big(2y^{ib}\big), \tag{1}$$

$$x^{ib} = \pi + \cos(\theta), \quad y^{ib} = \pi + \sin(\theta), \quad \theta \in [0, 2\pi]. \tag{2}$$

Note that this **does not change the test solution from problem 1**, so we can still use it to check our work. To solve this problem using the IBM/FCM we'll use the following procedure

1. Discretize (x^{ib}, y^{ib}) using N_{ib} points so that the spacing between these IB points is roughly the same as the Δx from the finite difference grid.

   (a) **Once your code works, experiment with this spacing. What happens when the IB points are very far apart compared to the finite difference mesh size? What happens when they are very close together? Look at both the accuracy (Problem 3 will give a better idea of this) and the speed of your code?**

2. Imagine we placed a charge q_k on each IB point (x^{ib}_k, y^{ib}_k). We want to *solve* for those charges q_k so that the internal boundary condition $u(x^{ib}, y^{ib}) = \cos\big(2x^{ib}\big)\sin\big(2y^{ib}\big)$ is maintained. The addition of charges modifies our PDE to

$$\nabla^2 u = f(x,y) + \sum_{k=1}^{N_ib} q_k \delta_a\left(x - x^{ib}_k, y - y^{ib}_k\right) = f(x,y) + \boldsymbol{\mathcal{S}}^{ib}\left(\boldsymbol{q}\right),$$

   where we've introduced the compact notation $\boldsymbol{\mathcal{S}}$ for the *spreading* operation involving the regularized delta functions δ_a. **For this assignment, take δ_a to be a 2D Gaussian with variance $a = 1.2\Delta x$ (this is also what's used in the example code on canvas).**

3. If we discretize our new PDE

$$\nabla^2 u = f(x,y) + \boldsymbol{\mathcal{S}}^{ib}(\boldsymbol{q})$$

directly, we'll have too many unknowns ($u(x_i, y_j$ and $\boldsymbol{q}$) and not enough equations. We can fix this by *interpolating* u onto the IB grid and enforcing our internal boundary condition

$$\iint_{(x,y)} u(x,y)\delta_a\left(x - x_k^{ib}, y - y_k^{ib}\right) dx\, dy = \boldsymbol{\mathcal{J}}^{ib}(u) = \cos\left(2x^{ib}\right)\sin\left(2y^{ib}\right).$$

4. After rearranging so that all of the unknowns are to one side, these equations together give the following **saddle point system**

$$\begin{bmatrix} \nabla^2 & -\boldsymbol{\mathcal{S}}^{ib} \\ \boldsymbol{\mathcal{J}}^{ib} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} u(x,y) \\ \boldsymbol{q} \end{bmatrix} = \begin{bmatrix} f(x,y) \\ \cos\left(2x^{ib}\right)\sin\left(2y^{ib}\right) \end{bmatrix}.$$

Discretize and solve this system

(a) **Don't try to construct all of this matrix directly!** Write a routine to *apply* the matrix using our functions for $\boldsymbol{\mathcal{S}}^{ib}$ and $\boldsymbol{\mathcal{J}}^{ib}$ discussed in class and implemented in the example code. Then solve the system with GMRES

(b) GMRES may have trouble converging with all of the default options. try using something like [what should 'tol' be?]

```
gmres(Amult, RHS, 1000, tol, 1)
```

(c) **Don't forget:** when discretizing you'll have to modify the right hand side to account for the Dirichlet boundary data on the box $x \in [0, 2\pi], y \in [0, 2\pi]$.

5. Once you've solved this system, **verify that you solution is still converging in Δx, but the order should decrease to $\Delta x^{1.5}$ when the IB is included.**

6. If everything has gone according to plan, it should look like you haven't done anything at all – see Fig. 2

Problem 3: An almost minimal surface

Now that the code from Problem 2 works, let's use it to solve a problem we don't know the answer to. Solve the following PDE

$$\nabla^2 u = \boldsymbol{\mathcal{S}}^{ib}(\boldsymbol{q}), \tag{3}$$
$$u(0,y) = u(2\pi,y) = u(x,0) = u(x,2\pi) = 0 \tag{4}$$
$$u(x^{ib}, y^{ib}) = \cos\left(2x^{ib}\right)\sin\left(2y^{ib}\right), \tag{5}$$
$$x^{ib} = \pi + \cos(\theta), \quad y^{ib} = \pi + \sin(\theta), \quad \theta \in [0, 2\pi]. \tag{6}$$

This is just Problem 2 with $f(x,y) = 0$ and homogeneous Dirichlet boundary conditions. The solution should look something like Fig. 1. Note that the shape of $u(x,y)$ is closely related (though not the same) as something called a minimal surface, which is the shape a soap film would make on a given wire-frame[1].

Problem 4: Charlie's rule

Charlie Peskin (the inventor of the IBM) says that you should never just stop one you have a cool code, **the best part is to play around with it.** So do something fun with your code and show it off! The following are a few ideas

1. The code for spreading $\boldsymbol{\mathcal{S}}^{ib}$ and interpolation $\boldsymbol{\mathcal{J}}^{ib}$ is extremely slow, but it doesn't need to be. The Gaussian kernel we use for δ_a is close to zero most of the time. Modify the code to *only* evaluate these functions when the value will be above some prescribed tolerance.

2. Try adding multiple shapes in the simulation to see what sort of surfaces you can get out of it

3. We can add *pseduo-dynamics* by moving our immersed boundary around or deforming it in time and re-solving for u. Make some fun videos of this.

4. GMRES performs much better with a good initial guess. Is there a way to make a good initial guess for problem 3? GMRES performs *even* better with an approximate inverse to the linear system, called a **preconditioner**. Read up on these and see if you have any ideas for this system.

5. We have code to impose a Dirichelet condition on $u(x^{ib}, y^{ib})$. Can we modify the code to impose a Neumann condition on $\frac{\partial u(x^{ib},y^{ib})}{\partial n}$? (i.e can we prescribe the normal derivative on the immersed boundary)

[1] $\nabla^2 u = 0$ doesn't quite define a minimal surface. The actual PDE is

$$\text{Div}\left[\frac{\nabla u}{1 + \|\nabla u\|^2}\right] = 0.$$

while non-linear equations are a bit harder, they're still doable and this may make for a nice project

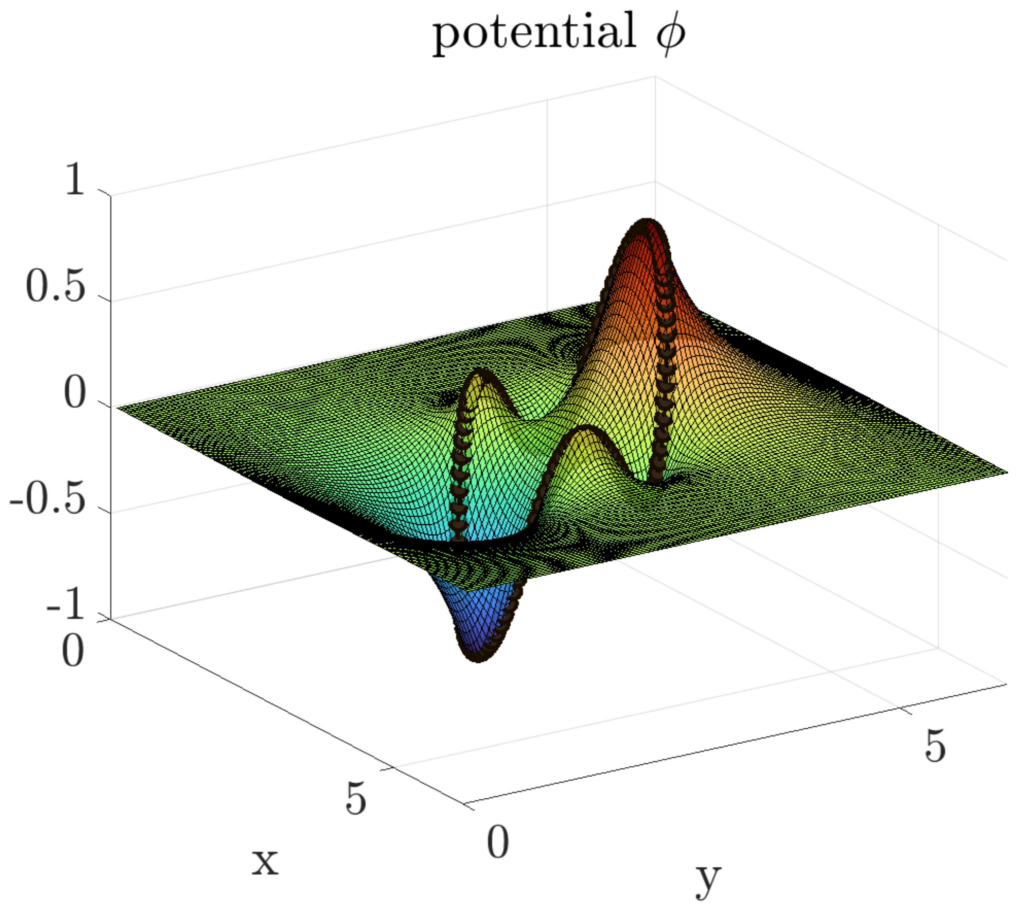

Figure 1: A plot of the solution $u(x, y)$ with the interpolated solution on the immersed boundary $u(x^{ib}, y^{ib})$, shown as a black line with markers representing the IB point locations.

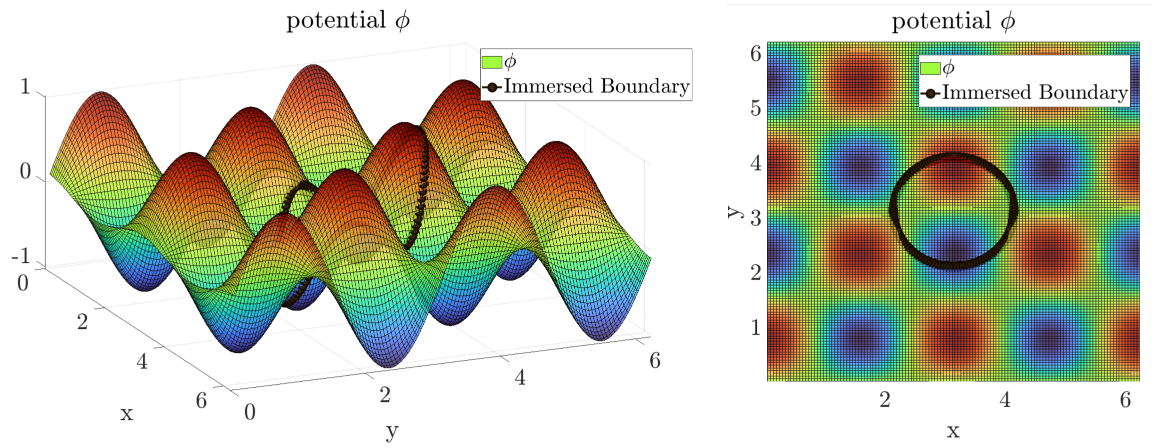

Figure 2: Left: A plot of the solution $u(x, y)$ with the interpolated solution on the immersed boundary $u(x^{ib}, y^{ib})$, shown as a black line with markers representing the IB point locations. Right: Top view of the left plot