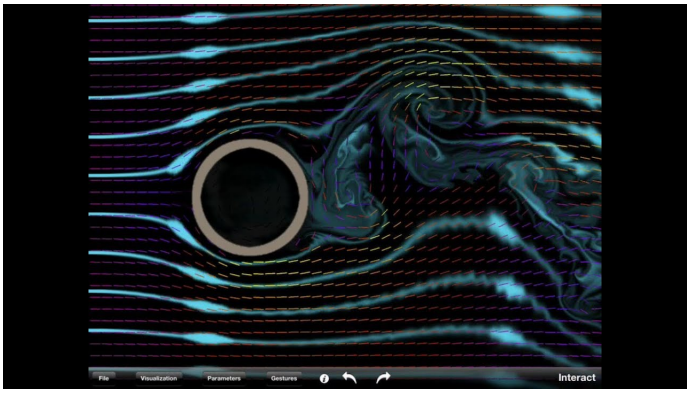
The Stream Function

MATH1091: ODE methods for a reaction diffusion equation

http://people.sc.fsu.edu/~jburkardt/classes/math1091_2020/stream/stream.pdf

---



Stream lines reveal the paths of fluid flow.

The Stream Function

The flow of an incompressible fluid in a 2D region, which is usually described by a vector field (u,v), can also be represented by stream function $\psi(x,y)$.

1 The continuity equation

At a given moment in time, the flow of a fluid in a 2D region can be represented by a velocity field, which we might represent as the vector field $\vec{u}(x,y)$ or as a pair of horizontal and vertical velocity components $(u(x,y),v(x,y))$. In general, the law of mass conservation must be applied to the *mass velocity*, that is, to the product of mass and velocity. But if the fluid is incompressible, then the mass conservation law can be applied directly to the velocity field itself, and has the form:

$$\nabla(u,v) \equiv \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \quad \text{(Continuity equation)}$$

This is generally referred to as the *continuity equation* since it can be interpreted to say that, at any point, the flow coming in must equal the flow going out. The operatator ∇ is computing the **divergence** of the vector field, and so the continuity equation is often stated as: *"The divergence of an incompressible flow is zero everywhere."*

2 Some sample flows

We will start by considering some simple examples of velocity flow fields:

- *channel*: parabolic flow through a straight channel in $[0,5] \times [-1,+1]$;
- *corner*: flow that turns around a corner in $[0,1] \times [0,1]$;
- *shear*: layers of flow at different speeds in $[0,1] \times [0,1]$;
- *vortex*: flow that rotates around a center in $[-1,1] \times [-1,1]$;

Each of the flows will be described using data in matrices. To discretize a flow, the user specifies `nr` and `nc`, the number of rows and columns in these matrices, then calling an *xy* function for the node coordinates, followed by a *uv* function for the velocity field. The MATLAB command *quiver()* or the web page function *arrows()* can be used to display the flow.

For instance, for the shear flow, we might use these commands:

```
nr = 11;
nc = 11;
[ X, Y ] = xy_shear ( nr, nc );
[ U, V ] = uv_shear ( X, Y );
arrows ( X, Y, U, V );
```

Listing 1: Set up the shear flow.

3 Exercise #1: Set up and view the example flows

Create a MATLAB file *exercise1.m* which sets up and displays each of the four example flows. Because the channel region is not square, you might use `nr=9, nc = 21` to get a better sense of the flow. For the other regions, you can use `nr=nc=11`.

If you call the *arrows()* function, you will see plots of both the velocity, and the velocity direction field. This can be helpful in cases where the velocities vary greatly in magnitude, so that the plot returned by *quiver()* would not show all the information clearly.

4 Computational representation of flow

The example flow functions create $nr \times nc$ arrays `X, Y, U, V` to represent the flow. There are many ways to store 2D data in an array, and unfortunately, every method can seem sensible, but then becomes confusing in certain applications.

For these example flows, I have chosen to set up the arrays so that the location of a value in the array roughly corresponds to its physical location. In particular, if $nr = 3$ and $nc = 4$, and we are working in the unit square, the values in the node coordinate arrays `X` and `Y` would be

```
i = 1  (0.0, 1.0)  (0.3, 1.0)  (0.6, 1.0)  (1.0, 1.0)
i = 2  (0.0, 0.5)  (0.3, 0.5)  (0.6, 0.5)  (1.0, 0.5)
i = 3  (0.0, 0.0)  (0.3, 0.0)  (0.6, 0.0)  (1.0, 0.0)
          j = 1       j = 2       j = 3       j = 4
```

and the same ordering is used for the velocity arrays `U` and `V`.

Question: Suppose you want to estimate a derivative of `U` at the node with (i, j) coordinates (2,3). How would you express:

1. a forward X difference?
2. a centered X difference?
3. a backward X difference?
4. a forward Y difference?
5. a centered Y difference?
6. a backward Y difference?

Answers:

1. $\frac{\partial U}{\partial X} \approx \frac{U(2,4)-U(2,3)}{dx}$
2. $\frac{\partial U}{\partial X} \approx \frac{U(2,4)-U(2,2)}{2dx}$
3. $\frac{\partial U}{\partial X} \approx \frac{U(2,3)-U(2,2)}{dy}$
4. $\frac{\partial U}{\partial Y} \approx \frac{U(1,3)-U(2,3)}{dy}$
5. $\frac{\partial U}{\partial Y} \approx \frac{U(1,3)-U(3,3)}{2dy}$
6. $\frac{\partial U}{\partial Y} \approx \frac{U(2,3)-U(3,3)}{dy}$

There are at least two surprises here. To increase the x coordinate (which we think of as the first coordinate), we must increase the j index (which is the second (i,j) coordinate). To increase the y coordinate, we must **decrease** the i coordinate, because y values increase going "up" the array, but row indices i decrease when we go "up". You don't need to memorize these rules, but just be aware that any time you try to express a difference using array data, you are going to be confused at first, and if you can't figure it out, you should come back and refer to these notes.

If you have the `X, Y, U, V` data for a flow, you can check the continuity equation. To do this, at every node (i, j), you want to approximate the divergence of the vector field. At any interior node, we can use centered differences. But if a node is on the boundary, then we can probably only use a forward or backward difference to estimate a derivative. In the next exercise, we will try to compute the divergence of each of our example flows.

5 Exercise #2: Test the continuity equation

Create a MATLAB file *divergence.m* by copying the file *skeleton2.m* and finishing it.

This function can be used to test the continuity equation for a given flow. This function will use a combination of backward, forward, and centered differences to estimate $\frac{du}{dx}$ and $\frac{dv}{dx}$ and then add them to get the divergence. The file is missing a few lines, which are indicated by question marks. You need to replace the question marks by the appropriate finite difference estimates:

```
function D = divergence ( X, Y, U, V )
  [ nr, nc ] = size ( U );

  dx = X(1,2) - X(1,1);
  dy = Y(1,1) - Y(2,1);

  for c = 1 : nc
    if ( c == 1 )
      dUdX(1:nr,c) = ( U(1:nr,c+1) - U(1:nr,c) ) / dx;
    elseif ( c < nc )
      dUdX(1:nr,c) = ?
    else
      dUdX(1:nr,c) = ?
    end
  end

  for r = 1 : nr
    if ( r == 1 )
      dVdY(r,1:nc) = ?
    elseif ( r < nr )
      dVdY(r,1:nc) = ?
    else
      dVdY(r,1:nc) = ?
    end
  end

  D = dUdX + dVdY
```

Fill in the missing formulas. Then test your code on each of the example flows, and print the value of `norm(D`. Do you get the expected value?

6 The stream function

Any pair of velocity component functions $u(x,y)$ and $v(x,y)$ that satisfies the continuity equation will represent a legitimate incompressible fluid flow. We will now look at a simple way to manufacture such velocity fields automatically, starting with a scalar function $\psi(x,y)$ whose only requirement is that it have continuous first and second partial derivatives. If a flow is derived in this way, $\psi(x,y)$ is called the **stream function** associated with the flow.

To see how the stream function can have this property, let us start with a given $\psi(x,y)$ and define

$$u(x,y) = \frac{\partial \psi}{\partial y}$$
$$v(x,y) = -\frac{\partial \psi}{\partial x}$$

In that case, we can verify that

$$\nabla(u,v) \equiv \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = \frac{\partial^2 \psi}{\partial x \partial y} - \frac{\partial^2 \psi}{\partial y \partial x} = 0$$

so any velocity field defined in this way represents an incompressible flow field.

For each of our example flows, it is possible to determine a corresponding stream function, and these functions are available on the web page. Thus, for example, to compute and display the stream function for the corner flow, we could write

```
nr = 11;
nc = 11;
[ X, Y ] = xy_corner ( nr, nc );
PSI = psi_corner ( X, Y );
contourf ( X, Y, PSI );
```

Listing 2: Compute and display $\psi(x\,y)$ for the corner flow.

7 Exercise #3: Compute the velocity field from $\psi(x,y)$

Create the file *uv_from_psi.m* by copying *skeleton3.m* and modifying it.

This function will use a combination of backward, forward, and centered differences, applied to the stream function ψ, in order to estimate the velocity components u and v. Where possible, derivatives should be estimated with centered differences, but for nodes on the boundary, it may be necessary to use forward or backward differences. The determination of how to compute the difference is similar to what you did in the divergence function.

The current version of the file is missing some information, indicated by question marks, which you should replace by the appropriate finite difference estimates:

```
function [ U, V ] = uv_from_psi ( X, Y, PSI )

  [ nr, nc ] = size ( PSI );

  dx = X(1,2) - X(1,1);
```

```
dy = Y(1,1) - Y(2,1);

U = zeros ( nr, nc );
V = zeros ( nr, nc );

for r = 1 : nr
  if ( r == 1 )
    U(r,1:nc) = ?
  elseif ( r < nr )
    U(r,1:nc) = ?
  else
    U(r,1:nc) = ?
  end
end

for c = 1 : nc
  if ( c == 1 )
    V(1:nr,c) = ?
  elseif ( c < nc )
    V(1:nr,c) = - ( PSI(1:nr,c+1) - PSI(1:nr,c-1) ) / dx / 2.0;
  else
    V(1:nr,c) = ?
  end
end

figure ( );
hold ( 'on' );
contour ( X, Y, PSI );
quiver ( X, Y, U, V );
hold ( 'off' );
```

Listing 3: Determine velocity from stream function.

Fill in the missing formulas. Then test your code on each of the example flows. As designed, the code automatically plots the flow field and the stream function together. You may notice a relationship between the velocity vectors and the stream function contour lines.

8 Computing ψ from the velocity

Now suppose that have velocity component functions u and v, and we want to determine the stream function $\psi(x, y)$ so that

$$u(x, y) = \frac{\partial \psi}{\partial y}$$
$$v(x, y) = -\frac{\partial \psi}{\partial x}$$

Now $\psi(x, y)$ is unique up to an integration constant, so we can choose some point (x_0, y_0) for which $\psi(x_0, y_0) = 0$. Having done this, we can now evaluate $\psi(x_1, y_1)$ using integration along a path from (x_0, y_0) to (x_0, y_1), and then from (x_0, y_1) to (x_1, y_1):

$$\psi(x_1, y_1) = \int_{y_0}^{y_1} u(x_0, y)\, dy - \int_{x_0}^{x_1} v(x_1, y_0)\, dx$$

For our problems, it is natural to choose the reference point (x_0, y_0) as the point of minimum x and y, which occurs at row nr and column 1. To estimate the value of $\psi(x, y)$ at location (r, c), we must integrate $u(x, y)$ up from row nr to row r, and then integrate $-v(x, y)$ to the right, from column 1 to column c.

9 Exercise #4: Compute ψ from a velocity field

Create the file *psi_from_uv.m*by copying *skeleton4.m* and modifying it.

This function will repeatedly use the trapezoid rule to estimate $\psi(x,y)$ at every node. Recall that the trapezoid rule averages the value of the integrand at the beginning and end of an interval, times the interval length.

Rather than make a separate calculation for every node, we will do our integration in steps:

- the value $\psi(nr,1)$ is 0.
- for $r = nr$ down to 1...
  - if $r == nr$, set $\psi(r,1) = 0$, else $\psi(r,1) = \psi(r+1,1)$+integral of u from $y(r+1)$ to $y(r)$;
  - for $c = 2:nc$, $\psi(r,c) = \psi(r,c-1)$+integral of $-v$ from $x(c-1)$ to $x(c)$;

Create your code by filling in the missing information that is indicated by question marks:

```
function PSI = psi_from_uv ( X, Y, U, V )

  [ nr, nc ] = size ( U );

  dx = X(1,2) - X(1,1);
  dy = Y(1,1) - Y(2,1);

  PSI = zeros ( nr, nc );

  for r = nr : -1 : 1

    if ( r == nr )
      PSI(r,1) = 0.0;
    else
      PSI(r,1) = PSI(r+1,1) + ?
    end

    for c = 2 : nc
      PSI(r,c) = PSI(r,c-1) - ?
    end

  end
```

Listing 4: Outline of code for computing stream function from velocities.

Try your code on each of the example flows. Contour plots of your computed value of PSI should match the plots you made earlier in exercise 3.

10 Streamline/vorticity equations

The stream function can be used in a program that computes a time dependent fluid flow, but in that case we need an additional variable, called the **vorticity**, which measures the rotating part of the flow. The vorticity is usually symbolized by $\omega(x,y)$, and is related to the velocity by the formula

$$\omega(x,y) = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}$$

If we are given the values of the vorticity field, then the stream function can be determined by

$$\nabla^2 \psi(x,y) = \omega(x,y)$$

Just as with the heat equation, where we also dealt with the ∇^2 operator, we generally need boundary conditions on ψ in order to make the solution unique.

To start a time-dependent fluid flow problem at time t_0, we assume we are given the initial values of the vorticity field, so that we can solve for the corresponding stream function. Now, assuming we have $\omega(x, y, t_0)$ and $\psi(x, y, t_0)$ at the current time, we can estimate the vorticity at the next time $t_1 = t_0 + dt$ by using a discretized version of the following differential equation:

$$\frac{\partial \omega}{\partial t} = -\frac{\partial \psi}{\partial x}\frac{\partial \omega}{\partial y} + \frac{\partial \psi}{\partial y}\frac{\partial \omega}{\partial x} + \nu\frac{\partial^2 \omega}{\partial x^2} + \nu\frac{\partial^2 \omega}{\partial y^2}$$

where ν is a physical constant known as the viscosity. Once we have the new $\omega(x, y, t_1)$, we can again solve for the corresponding $\psi(x, y, t_1)$, and continue marching forward in time.

11 Exercise #5: A streamline/vorticity code

The file *skeleton5.m* contains an almost complete code for taking 8 time steps in a fluid flow problem, using the stream function and vorticity formulation. The stream function is represented by the variable `P` and vorticity by `O`. The code defines quantities `C, S, N, E, W` which are lists of the indices necessary to form all the finite difference estimates of derivatives in the differential equation. For instance, the quantity $\nu\frac{\partial^2 \omega}{\partial y^2}$ is to be approximated by `( nu / dy^2 ) * ( O(N) - 2.0 * O(C) + O(S) )`.

Copy *skeleton5.m*, and go to the line where the forward Euler approximation is being set up, which currently looks like this:

```
O(C) = O(C) + dt * (
  - dP/dX * dO/dY
  + dP/dY * dO/dX
  + nu * d^2 O/dX^2
  + nu * d^2 O/dY^2
  );
```

Listing 5: Euler approximation to be filled in.

Replace each mathematical expression by the appropriate finite difference quantity. Run the code. It should compute 8 time steps, pausing after each step to display the current plots of ω and ψ. If your code runs correctly, you should see the vorticity plot start as a tall thin oval shape, which then begins to turn and twist gradually.

12 Homework: Turn raw fluid data into a stream function plot

Create a MATLAB file *hw8.m*.

From the web page, get the two data files *xy_inout.txt* and *uv_inout.txt*. The "inout" flow problem was solved on a 41×41 grid = 1681 nodes. The file *xy_inout.txt* consists of 1681 lines; the k-th line specifies the (x, y) coordinates of the k-th node. Similarly, the k-th line of *uv_inout.txt* lists the (u, v) velocity values of the k-th node. The ordering of the nodes is similar to what we have seen in the example flow problems, and is suggested by this diagram:

```
 1641 1642 1643 1644 ... 1680 1681
.... .... .... .... ... .... ....
  83   84   85   85 ...  122  123
  42   43   44   45 ...   81   82
   1    2    3    4 ...   40   41
```

In other words, the first node has the lowest values of X and Y, the last node has the highest values, and in between the nodes are listed from left to right, and bottom to top.

Your program should use the MATLAB statement `result=load(filename)` to read the two files of data. For instance, read the node coordinates and separate them by the commands:

```
xy = load ( 'xy_inout.txt' );
x = xy(:,1);
y = xy(:,2);
```

Similarly, read the velocity data as `uv` and split it into two vectors `u` and`v`.

Now we need to change these items from vectors to matrices. Use the `reshape()` command to create a matrix `X` from the vector `x`, and similarly create `Y, U, V`.

Then call your function `psi_from_uv()` to compute the stream function for this data.

Use `contour()` or `contourf()` to plot your stream function, and save this image as the file *hw8.png*.

Send me the plot *hw8.png* at **jvb25@pitt.edu**. I would like to see your work by Friday, 26 June 2020.