Computational Science: Computational Methods in Engineering

Building Geometries into Data Arrays

Outline

- Visualizing MATLAB Data and Arrays
- 3D → 2D → 1D
- Arrays, x and y in MATLAB
- Building Geometries in Arrays
  - Initializing arrays
  - Array indexing
  - Squares and rectangles
  - Simple triangles and arbitrary polygons
  - Circles and ellipses
  - Formed half-spaces
  - Linear half-spaces
  - Boolean operations
  - Scaling data in arrays
WARNING: Not Meant for Graphics!

This lecture teaches techniques that are NOT intended for generating graphics. See previous lecture if that is your purpose.

Instead, the techniques in this lecture are intended for you to build arrays containing different shapes and geometries so that you can do numerical computation on those shapes and geometries.

Visualizing MATLAB Data and Arrays
1D Data Arrays

Row Vectors
Row vectors are most commonly used to store one-dimensional data. They can be used to label axes on grids, store functions, and more. Row vectors are used in some matrix algorithms, but less frequently.

\[
\begin{bmatrix}
1 & 2 & 3 & 4 & 5
\end{bmatrix}
\]

Column Vectors
Column vectors can be used the same way as row vectors, but column vectors are used more commonly in linear algebra and matrix manipulation.

\[
\begin{bmatrix}
0.8 & 0.6 & 1.0 & 1.0 & 0.4 \\
0.9 & 0.1 & 1.0 & 0.5 & 0.9 \\
0.1 & 0.3 & 0.2 & 0.8 & 0.8 \\
0.9 & 0.5 & 1.0 & 0.1 & 1.0 \\
\end{bmatrix}
\]

2D Data Arrays
A 2D array could be a matrix, a JPEG image, a 2D set of data, or many other things. MATLAB does not differentiate between these and treats them the same. It is up to you to know the difference and stay consistent in your code.

\[
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{bmatrix}
\]
Visualizing 1D Arrays

\[
\begin{align*}
\text{phi} &= \text{linspace}(0,2\pi,10); \\
y &= \sin(\text{phi}); \\
\text{plot}(\text{phi},y);
\end{align*}
\]

Visualizing 2D Arrays

\[
A = \begin{bmatrix}
1 & 2 & 3 \\ 
4 & 5 & 6 \\ 
7 & 8 & 9
\end{bmatrix};
\]
\[
\text{imagesc}(A); \\
\text{colorbar};
\]

\[
A = \begin{bmatrix}
1 & 2 & 3 \\ 
4 & 5 & 6 \\ 
7 & 8 & 9
\end{bmatrix};
\]
\[
\text{pcolor}(A); \\
\text{colorbar};
\]
**linspace() vs. [a:Δ:b]**

**LINSPACE**
\[
xa = \text{linspace}(0,10,10);
\]
\[
xa = 0  1.11  2.22  3.33  4.44  5.55  6.66  7.77  8.88  10.00
\]
- Easier to control number of points.
- Easier to control position of end points.
- More difficult to control step size.

**DIRECT ARRAY**
\[
Nx = 10;
dx = 1;
xa = [0:Nx-1]*dx;
\]
\[
xa = 0  1  2  3  4  5  6  7  8  9
\]
- Easier to control step size.
- More difficult to control number of points.
- More difficult to control position of last point.

3D → 2D → 1D
3D

All physical devices are three-dimensional.

Numerical Complexity

Typical grid required to model a 3D device.

Size of 3D Problem
20x20x100 = 40,000 points

Size of 2D Problem
20x100 = 2,000 points

Size of 1D Problem
100 points

Can we simulate 3D devices in one or two dimensions?
Sometimes it is possible to describe a physical device using just two dimensions. Doing so dramatically reduces the numerical complexity of the problem and is ALWAYS GOOD PRACTICE.

Many times it is possible to approximate a 3D device in two dimensions. It is very good practice to at least perform the initial simulations in 2D and only moving to 3D to verify the final design.

Effective indices are best computed by modeling the vertical cross section as a slab waveguide. A simple average index can also produce good results.
Sometimes it is possible to describe a physical device using just one dimension. Doing so dramatically reduces the numerical complexity of the problem and is ALWAYS GOOD PRACTICE.

Arrays, \( x \) and \( y \) in MATLAB
How MATLAB Indexes Arrays

MATLAB uses matrix notation for indexing arrays.

\[
\begin{bmatrix}
  a_{11} & a_{12} & a_{13} & a_{14} \\
  a_{21} & a_{22} & a_{23} & a_{24} \\
  a_{31} & a_{32} & a_{33} & a_{34} \\
  a_{41} & a_{42} & a_{43} & a_{44}
\end{bmatrix}
\]

\( a_{mn} \)  \( m \) is the row number  \( n \) is the column number

In MATLAB notation, \( a_{mn} \) is indexed as \( A(m,n) \).

In this sense, the first number is the vertical position and the second number is the horizontal position. This is like \( f(y,x) \) and so it is awkward for us to think about when the array is not a matrix.

To be consistent with matrix notation, the index of the first element in an array is 1, not zero like in other programming languages like C or Fortran.

A More Intuitive Way of Indexing Arrays

Experience suggests that one of the most challenging tasks in numerical modeling is representing devices on a grid.

To be more intuitive, we would like the first argument when indexing an array to be the horizontal position and the second to be the vertical position so that it looks like \( f(x,y) \) instead of \( f(y,x) \).

For this reason, we will treat the first argument of an array as the horizontal position and the second as the vertical position. This is consistent with the standard \( f(x,y) \) notation.

Think \( A(nx, ny) \) instead of \( A(m, n) \).

This is fine, but MATLAB still treats the array otherwise. We only need to consider how MATLAB handles things when using the \texttt{meshgrid()} command or when using plotting commands.
The `meshgrid()` command allows complex equations involving grid coordinates to be typed directly into MATLAB without the need of using `for` loops to iterate across the grid.

### MATLAB Standard Use of `meshgrid()`

```matlab
xa = [0:Nx-1]*dx;
ya = [0:Ny-1]*dy;
[X,Y] = meshgrid(xa,ya);
```

### Revised Use of `meshgrid()`

```matlab
xa = [0:Nx-1]*dx;
YA = [0:Ny-1]*dy;
[Y,X] = meshgrid(YA,xa);
```

We will do it this way.
Revised Plot Commands

MATLAB Standard Use of `imagesc()`

```
imagesc(xa,ya,A);
```

Revised Use of `imagesc()`

```
imagesc(xa,ya,A.);
```

This fails to properly convey our sense of x and y.

```
>> A = zeros(4,4);
>> A(2,3) = 1;
>> A
```

```
A =
     0     0     0     0
     0     0     1     0
     0     0     0     0
     0     0     0     0
```

```
>> A.'
```

```
ans =
     0     0     0     0
     0     0     0     0
     0     1     0     0
     0     0     0     0
```

Building Geometries into Data Arrays
Building a Geometry?

A geometry is “built” into an array when you can visualize the array and see the desired geometry.

Suppose we wish to “build” a circle into this array.

We fill in our array so that when the array is plotted we see our circle.

Initializing Data Arrays

\[
\begin{align*}
\text{Nx} &= 10; \\
\text{Ny} &= 10; \\
A &= \text{zeros}(\text{Nx}, \text{Ny}); \\
\end{align*}
\]
Adding Rectangles to an Array

Consider adding rectangles by first computing the start and stop indices in the array, then filling in the array.

```
ny2 = 8;
nx1 = 3;
nx2 = 6;
A = zeros(Nx,Ny);
A(nx1:nx2,ny1:ny2) = 1;
```

Centering a Rectangle

It is often necessary to center a rectangle type structure in an array.

```
% DEFINE GRID
Sx = 1;     %physical size along x
Sy = 1;     %physical size along y
Nx = 10;    %number of cells along x
Ny = 10;    %number of cells along y

% DEFINE RECTANGLE SIZE
wx = 0.2;
wy = 0.6;

% COMPUTE POSITION INDICES
dx = Sx/Nx;
nx = round(wx/dx);
nx1 = 1 + floor((Nx - nx)/2);
nx2 = nx1 + nx - 1;

dy = Sy/Ny;
ny = round(wy/dy);
ny1 = 1 + floor((Ny - ny)/2);
ny2 = ny1 + ny - 1;

% CREATE A
A = zeros(Nx,Ny);
A(nx1:nx2,ny1:ny2) = 1;
```
A Simple Centered Triangle

% TRIANGLE
w = 0.8*Sx;
h = 0.9*Sy;

% CREATE CENTERED TRIANGLE
ER = zeros(Nx,Ny);
ny = round(h/dy);
ny1 = 1 + floor((Ny - ny)/2);
y2 = ny1 + ny - 1;
for ny = ny1 : ny2
   f = (ny - ny1 + 1)/(ny2 - ny1 + 1);
   nx = round(f*w/dx);
   nx1 = 1 + floor((Nx - nx)/2);
   nx2 = nx1 + nx - 1;
   ER(nx1:nx2,ny) = 1;
end

Creating Arbitrary Polygons

% DEFINE VERTICES OF POLYGON
p1 = [ 0.3 ; 0.1 ];
p2 = [ 0.8 ; 0.2 ];
p3 = [ 0.7 ; 0.9 ];
p4 = [ 0.6 ; 0.4 ];
p5 = [ 0.1 ; 0.8 ];
P = [ p1 p2 p3 p4 p5 ];

% CALL POLYFILL() TO FILL
% POLYGON IN ARRAY A
A = polyfill(xa,ya,P);
Get polyfill() from course website.
Circles

For circles and ellipses, consider using MATLAB's `meshgrid()` command.

```matlab
% DEFINE GRID
Sx = 1;     % physical size along x
Sy = 1;     % physical size along y
Nx = 10;    % number of cells along x
Ny = 10;    % number of cells along y

% GRID ARRAYS
dx = Sx/Nx;
x = (0:Nx-1)*dx;
xa = xa - mean(xa);
dy = Sy/Ny;
ya = (0:Ny-1)*dy;
ya = ya - mean(ya);
[Y,X] = meshgrid(ya,xa);

% CREATE CIRCLE
r = 0.4;
A = (X.^2 + Y.^2) <= r^2;
```

Ellipses

Ellipses are like circles, but have two radii. You can still use the `meshgrid()` command for these.

```matlab
% DEFINE GRID
Sx = 1;     % physical size along x
Sy = 1;     % physical size along y
Nx = 20;    % number of cells along x
Ny = 20;    % number of cells along y

% GRID ARRAYS
dx = Sx/Nx;
x = (0:Nx-1)*dx;
xa = xa - mean(xa);
dy = Sy/Ny;
ya = (0:Ny-1)*dy;
ya = ya - mean(ya);
[Y,X] = meshgrid(ya,xa);

% CREATE ELLIPSE
rx = 0.35;
ry = 0.45;
A = ((X/rx).^2 + (Y/ry).^2) <= 1;
```
Offset Ellipses

Ellipses and circles can be placed anywhere on the grid.

% DEFINE GRID
Sx = 1;     %physical size along x
Sy = 1;     %physical size along y
Nx = 20;    %number of cells along x
Ny = 20;    %number of cells along y

% GRID ARRAYS
dx = Sx/Nx;
xa = [0:Nx-1]*dx;
xa = xa - mean(xa);

dy = Sy/Ny;
ya = [0:Ny-1]*dy;
ya = ya - mean(ya);

[Y,X] = meshgrid(ya,xa);

% CREATE ELLIPSE
xc = -0.15;
yc = +0.25;
rx = 0.4;
ry = 0.2;
A  = ( ((X - xc)/rx).^2 + …
     ((Y - yc)/ry).^2 ) <= 1;

Radial & Azimuthal Geometries

Meshgrid

Radial Grid
RSQ = X.^2 + Y.^2;

Azimuthal Grid
THETA = atan2(Y,X);

Radial grid lets you create circles, ellipses, rings and more.

Azimuthal grid lets you create pie wedges and more.
Formed Half-Spaces

We can “fill in” half the grid under an arbitrary function like this...

% DEFINE GRID
Sx = 1; % physical size along x
Sy = 1; % physical size along y
Nx = 20; % number of cells along x
Ny = 20; % number of cells along y

% GRID ARRAYS
dx = Sx/Nx;
xa = [0:Nx-1]*dx;
xa = xa - mean(xa);
dy = Sy/Ny;
ya = [0:Ny-1]*dy;
ya = ya - mean(ya);

% CALCULATE SURFACE
y = 0.2 + 0.1*cos(4*pi*xa/Sx);

% FILL HALF SPACE
A = zeros(Nx,Ny);
for nx = 1 : Nx
    ny = round((y(nx) + Sy/2)/dy);
    A(nx,1:ny) = 1;
end

Linear Half-Spaces (1 of 2)

Given two points \((x_1, y_1)\) and \((x_2, y_2)\), an equation for the line passing through these two points is:

\[
(y - y_i) = m(x - x_i)
\]

\[
m = \frac{y_2 - y_1}{x_2 - x_1}
\]

This equation can be rearranged as

\[
(y - y_i) - m(x - x_i) = 0
\]

The space on one half of this line is called a half-space. It is defined as:

\[
(y - y_i) - m(x - x_i) > 0
\]

or

\[
(y - y_i) - m(x - x_i) < 0
\]
A half-space can be filled anywhere on the grid.

% DEFINE GRID
Sx = 1;     % physical size along x
Sy = 1;     % physical size along y
Nx = 20;    % number of cells along x
Ny = 20;    % number of cells along y

% GRID ARRAYS
dx = Sx/Nx;
xa = [0:Nx-1]*dx;
xa = xa - mean(xa);
dy = Sy/Ny;
ya = [0:Ny-1]*dy;
ya = ya - mean(ya);

[Y,X] = meshgrid(ya,xa);

% DEFINE TWO POINTS
x1 = -0.50;
y1 = +0.25;
x2 = +0.50;
y2 = -0.25;

% FILL HALF SPACE
m = (y2 - y1)/(x2 - x1);
A = (Y - y1) - m*(X - x1) > 0;

Creating Linear Gradients (1 of 2)

The distance from a point to a line is calculated as

Two Dimensions

\[ d = \frac{\left| (y_2 - y_1)x_0 - (x_2 - x_1)y_0 + x_2y_1 - x_1y_2 \right|}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}} \]

\[ d = \frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}} \quad \text{ if } \quad ax_0 + by_0 + c = 0 \]

Three Dimensions

\[ d = \frac{|\vec{r}_2 - \vec{r}_1 \times (\vec{r}_1 - \vec{r}_0)|}{|\vec{r}_2 - \vec{r}_1|} \]
Creating Linear Gradients (2 of 2)

\[ d = \frac{(y_2 - y_1)x_0 - (x_2 - x_1)y_0 + x_2y_1 - x_1y_2}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}} \]

\[ d = \frac{(y_2 - y_1)x_0 - (x_2 - x_1)y_0 + x_2y_1 - x_1y_2}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}} \]

Creating Thick Bars

We can use the linear gradients to create thick bars through our grid with any thickness and orientation.

\[ D = (y_2 - y_1)x - (x_2 - x_1)y + x_2y_1 - x_1y_2; \]
\[ D = \left| D \right| / \sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}; \]

\[ B = D < 0.7; \]
Creating Thick Bars on 3D Grids

\[ \delta = \frac{\vec{r}_2 - \vec{r}_1}{|\vec{r}_2 - \vec{r}_1|} \]
\[ d = |\delta \times (\vec{r}_1 - \vec{r}_2)| \]
\[ d = \sqrt{\left[ \delta_z (z_i - z_j) - \delta_y (y_i - y_j) \right]^2 + \left[ \delta_y (x_i - x_j) - \delta_z (z_i - z_j) \right]^2 + \left[ \delta_x (y_i - y_j) - \delta_z (z_i - z_j) \right]^2} \]

Masking and Boolean Operations

The figure below is the formed surfaced masked by a linear half-space.
Comparison of Boolean Operations

A + B  A&B  A|B  A.*B
xor(A, B)  not(A)  xor(A, A&B)  xor(B, A&B)

Blurring Geometries

Blurring is used to resolve surfaces that slice though the middle of cells or to build more realistic device geometries.

```
% DEFINE GRID
Sx = 1;  % physical size along x
Sy = 1;  % physical size along y
Nx = 21; % number of cells along x
Ny = 21; % number of cells along y

% GRID ARRAYS
dx = Sx/Nx;
xa = [0:Nx-1]*dx;
xa = xa - mean(xa);
dy = Sy/Ny;
yn = [1:Ny-1]*dy;
yn = yn - mean(yn);
[Y,X] = meshgrid(yn,xa);

% CREATE A CROSS
ER = abs(X)<=0.075 | abs(Y)<=0.075;

% CREATE BLUR FUNCTION
B = exp(-(X.^2 + Y.^2)/0.1^2);

% PERFORM BLUR OPERATION
ER = fft2(ER).*fft2(B)/sum(B(:));
ER = ifftshift(real(ifft2(ER)));

% PERFORM THRESHOLD OPERATION
ER = ER > 0.4;
```
Eventually, we need to build devices on a grid. This is done by a dielectric constant to specific geometries in the array. Typically, the background will be air with a dielectric constant of 1.0.

\[
er_1 = 1.0; \\
er_2 = 2.4; \\
A = er_1*(1-A) + er_2*A; \\
A = er_1 + (er_2 – er_1)*A;
\]