3D Matlab basics

```
clear all, close, clc
set_defaults()
```

In two dimensions we will extensively use two functions for plotting:

1. meshgrid()
2. reshape()

These functions have an internal logic that is counter-intuitive and forces us to use certain convections to avoid trouble later.

Meshgrid()

The function meshgrid() takes three vectors **x, y** and **z** that contain the location of the grid points and generates matrices **X, Y** and **Z** that are used by all 3D Matlab plotting functions, in particular slice()

```
f = @(x,y,z) (x.^2).*y.*z;
g = @(x,y,z) y;
h = @(x,y,z) z;

Nx = 4; Ny = 3; Nz = 2; N = Nx*Ny*Nz
```

```
N = 24
```

```
x = linspace(0,1,Nx)
```

```
x = 1×4
         0    0.3333    0.6667    1.0000
```

```
y = linspace(0,2,Ny)
```

```
y = 1×3
     0     1     2
```

```
z = linspace(0,3,Nz)
```

```
z = 1×2
     0     3
```

```
[X,Y,Z] = meshgrid(x,y,z)
```

```
X = 
X(:,:,1) =

         0    0.3333    0.6667    1.0000
         0    0.3333    0.6667    1.0000
         0    0.3333    0.6667    1.0000


X(:,:,2) =

         0    0.3333    0.6667    1.0000
         0    0.3333    0.6667    1.0000
         0    0.3333    0.6667    1.0000
```

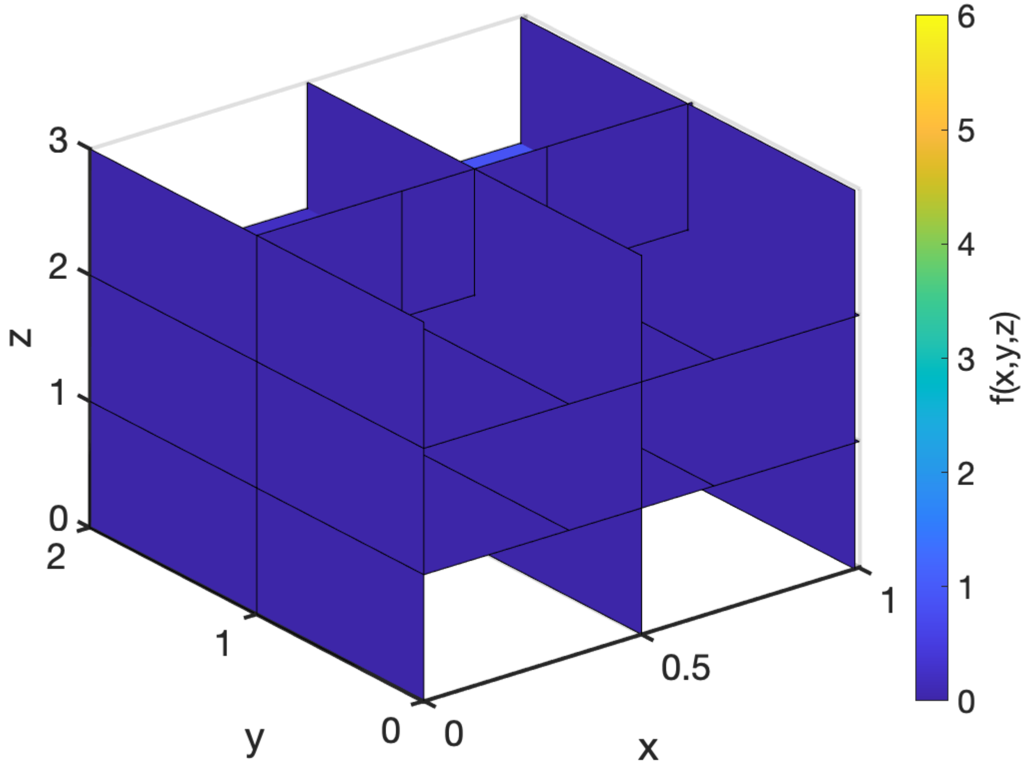
```
Y =
Y(:,:,1) =

     0     0     0     0
     1     1     1     1
     2     2     2     2


Y(:,:,2) =

     0     0     0     0
     1     1     1     1
     2     2     2     2
Z =
Z(:,:,1) =

     0     0     0     0
     0     0     0     0
     0     0     0     0


Z(:,:,2) =

     3     3     3     3
     3     3     3     3
     3     3     3     3
```

```
size(X)
```

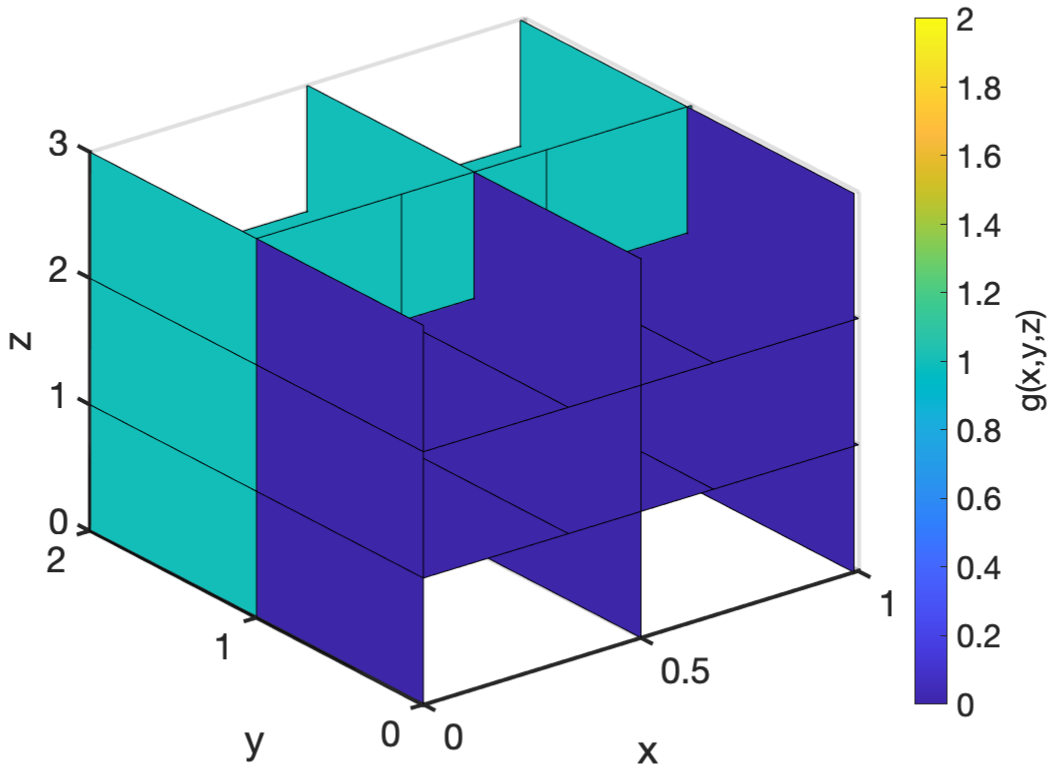
```
ans = 1×3
     3     4     2
```

In the matrices **X** and **Y**, the y-value increases with the row index, i, and the x-value increases with the column index, j and k in the third direction. Since we index matrices as **X**(i,j,k) and **Y**(i,j,k), and **Z**(i,j,k), the first index is the y-coordinate. **This makes it natural to order our grid y-first - see below!**

```
figure()

xslice = [0 0.5 0.99];                           % define the cross sections to vi
yslice = [1 ];
zslice = ([1 2]);

slice(X, Y, Z, f(X,Y,Z), xslice, yslice, zslice)    % display the slices
xlabel 'x', ylabel 'y', zlabel 'z'
cb = colorbar;                                  % create and label the colorbar
cb.Label.String = 'f(x,y,z)';
```




```
figure()

xslice = [0 0.5 0.99];                              % define the cross sections to vi
yslice = [1 ];
zslice = ([1 2]);

slice(X, Y, Z, g(X,Y,Z), xslice, yslice, zslice)    % display the slices
xlabel 'x', ylabel 'y', zlabel 'z'
cb = colorbar;                                      % create and label the colorbar
cb.Label.String = 'g(x,y,z)';
```



Reshape()

The solution of the PDE is calculated as a column vector, **u**. To plot the solution **u** has to be reshaped into a matrix that is compatible with the ordering of the **X, Y** and **Z** matrices produced by meshgrid. The Matlab function reshape() allows us to move from vectors to matrices and back.

1) From a matrix to a vector

There are two options to turn a matrix into a column vector.

1. Colon operator
2. reshape()

```
X
```

```
X =
X(:,:,1) =

         0    0.3333    0.6667    1.0000
         0    0.3333    0.6667    1.0000
         0    0.3333    0.6667    1.0000


X(:,:,2) =

         0    0.3333    0.6667    1.0000
         0    0.3333    0.6667    1.0000
         0    0.3333    0.6667    1.0000
```

```
x1 = X(:)
```

```
x1 = 24×1
         0
         0
         0
    0.3333
    0.3333
    0.3333
    0.6667
    0.6667
    0.6667
    1.0000
      ⋮
```

```
x2 = reshape(X,N,1)
```

```
x2 = 24×1
         0
         0
         0
    0.3333
    0.3333
    0.3333
    0.6667
    0.6667
    0.6667
    1.0000
      ⋮
```

Note, both ways stack the columns of **X** into a column.

Of course reshape() is the more general, it allows you to transform X into any matrix or vector with the same number of elements

```
reshape(X,1,N)      % row vector
```

```
ans = 1×24
         0         0         0    0.3333    0.3333    0.3333    0.6667    0.6667 ···
```

```
%reshape(X,Nx,Ny)   % flip the dimension of the 2D matrix
reshape(X,Ny,Nx,Nz) % flip the dimension of the 3D matrix
```

```
ans = 
ans(:,:,1) =

         0    0.3333    0.6667    1.0000
         0    0.3333    0.6667    1.0000
         0    0.3333    0.6667    1.0000


ans(:,:,2) =

         0    0.3333    0.6667    1.0000
         0    0.3333    0.6667    1.0000
         0    0.3333    0.6667    1.0000
```

1) From vector to matrix

Suppose the solution is given by $\mathbf{g} = g(\mathbf{x})$

```
soln = g(X(:),Y(:),Z(:))
```

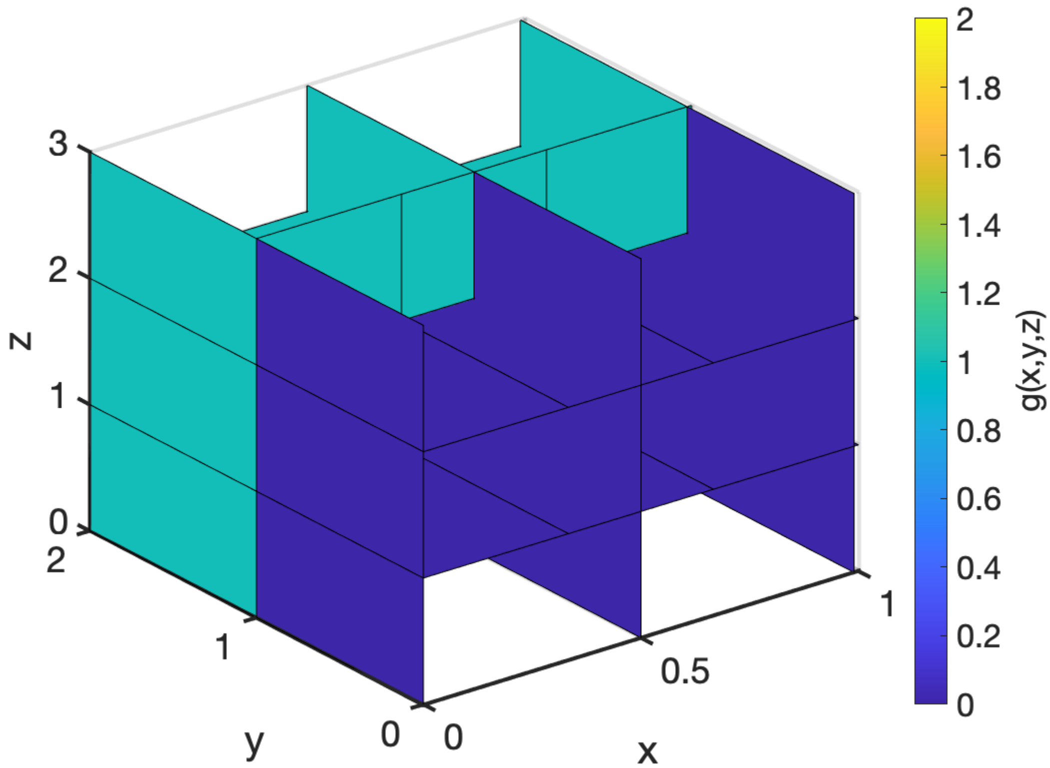
```
soln = 24×1
     0
     1
     2
     0
     1
     2
     0
     1
     2
     0
     ⋮
```

To plot this solution we need to transfer it back to a matrix. To be compatible with X and Y from meshgrid this matrix has to be of size Ny by Nx!

```
%SOLN = reshape(soln,Ny,Nx)
SOLN = reshape(soln,Ny,Nx,Nz)
```

```
SOLN = 
SOLN(:,:,1) =

     0     0     0     0
     1     1     1     1
     2     2     2     2


SOLN(:,:,2) =

     0     0     0     0
     1     1     1     1
     2     2     2     2
```

```
figure()

xslice = [0 0.5 0.99];                              % define the cross sections to vi
yslice = [1 ];
zslice = ([1 2]);

slice(X, Y, Z, SOLN, xslice, yslice, zslice)    % display the slices
xlabel 'x', ylabel 'y', zlabel 'z'
cb = colorbar;                                  % create and label the colorbar
cb.Label.String = 'g(x,y,z)';
```



Notice, that Ny is the first entry, because meshgrid() has a y-first ordering!

2D grid with y-first ordering

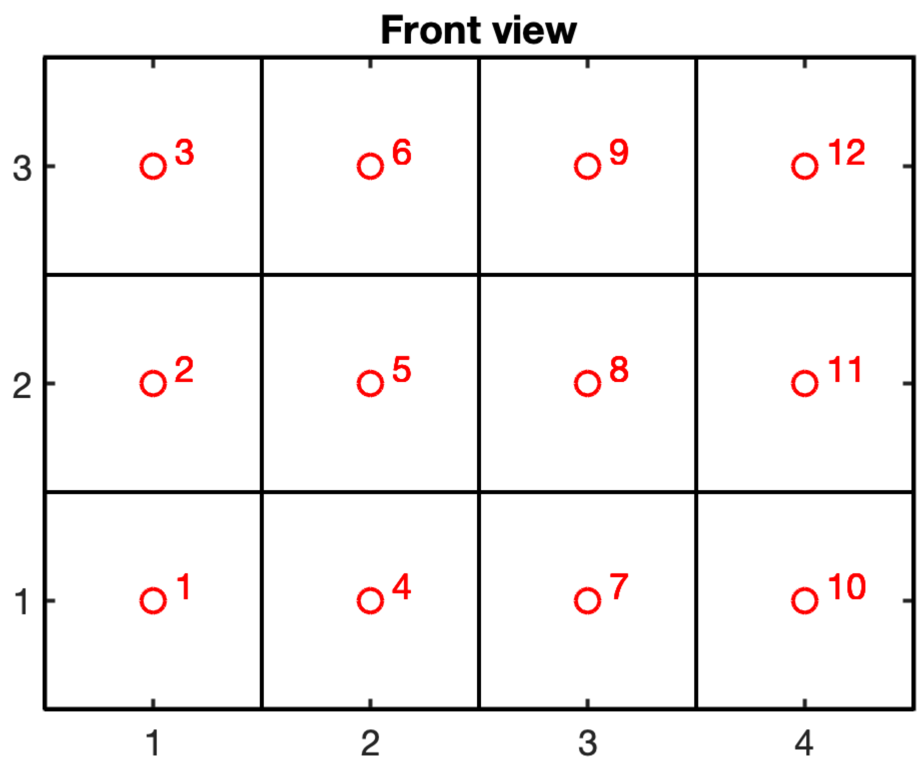
Given that meshgrid() has an internal y-first ordering, we use a computational grid with a y-first ordering. This way we avoid a lot of problems!

```
Nx = 4;
Ny = 3;
Nz = 2;
N = Nx*Ny*Nz;

x = 1:Nx;
y = 1:Ny;
z = 1:Nz;
dof = 1:N;

[X,Y,Z] = meshgrid(x,y,z);

DOF = reshape(dof,Ny,Nx,Nz)
```

```
DOF =
DOF(:,:,1) =

     1     4     7    10
     2     5     8    11
     3     6     9    12
```

```
DOF(:,:,2) =

    13    16    19    22
    14    17    20    23
    15    18    21    24
```

```
plot([.5 Nx+.5 Nx+.5 .5 .5],[.5 .5 Ny+.5 Ny+.5 .5],'k'), hold on
title 'Front view'
for i=1:Nx
    plot([x(i)+.5 x(i)+.5],[.5 Ny+.5],'k-')
    for j=1:Ny
        plot([.5 Nx+.5],[y(j)+.5 y(j)+.5],'k-')
        plot(X(j,i),Y(j,i),'ro','markerfacecolor','w')
        text(X(j,i)+.1,Y(j,i)+.07,num2str(DOF(j,i)),'fontsize',18,'color','r')
    end
end
set(gca,'xtick',[1:Nx],'ytick',[1:Ny])
axis equal tight
```



Tensor Product

Because we are discretizing the differential operators on a regular mesh the tensor product will be a key for the efficient, clean and simple implementation of the multi dimensional operators. The tensor product is also called the Kronecker product, hence the respective Matlab function is kron(). We will learn more about it later, but below is the basic use.

```
A = eye(4);
B = [1 2;...
     4 5];

AoB = kron(A,B)

BoA = kron(B,A)
```

Auxillary functions

set_defaults()

```
function [] = set_defaults()
set(0, ...
    'defaultaxesfontsize',   18, ...
    'defaultaxeslinewidth',   2.0, ...
    'defaultlinelinewidth',   2.0, ...
    'defaultpatchlinewidth',  2.0,...
    'DefaultLineMarkerSize', 12.0);
end
```