Lecture 20
Regular Expressions and More Image Processing

Lecture 20 Goals

Searching Strings and Filenames

- String Searching
- Directories and Filenames
- MATLAB Struct Arrays
- Searching Directories and Wildcards
- Regular Expressions

2D Data Visualization

- 2D plot types
- Lighting and Shading
- Color scales and colormap

String Searching

Finding strings in strings (i.e., substrings) is a simple concept.

We can detect whether a substring exists or not using `contains()`.

```
>>  s1 = "Puppies vs. Babies";
>>  contains(s1,'Babies')
ans =
  logical
    1
>>  contains(s2,'Kitties')
ans =
  logical
    0
```

String Searching

Note that `contains()` is case sensitive by default!

```
>>  s1 = "Puppies vs. Babies";
>>  contains(s1,'puppies')
ans =
  logical
    0
>>  contains(lower(s1),'puppies')
ans =
  logical
    1
>>  contains(s1,'puppies','IgnoreCase',true)
ans =
  logical
    1
```

String Searching

To obtain the location of each detected substring, we can use `strfind()`

```
>>  s1 = "Puppies vs. Babies";
>>  strfind(s1,'puppies')
ans =
    13
>>  strfind(s1,'ies')
ans =
    5   16
>>  strfind(s1,'dogs')
ans =
    []
```

Note: `findstr` *also works, but this is obsolete*

Directories and Filenames

Directory contents can be read using the `dir` function. The `dir` function returns a MATLAB struct array.

```
>>  f = dir
f =
  10×1 struct array with fields:
    name
    folder
    date
    bytes
    isdir
    datenum
>>  f = dir('data');              % specify a relative path
>>  f = dir('/home/cs4user/');    % specify an absolute path
```

What is a “struct array”?

MATLAB Structs

Structs are just arrays that have common keywords, but can have mixed data types (an arbitrary container class)

```
>>  a = struct('a',1,'b',2,'c',3)
a =
  struct with fields:

    a: 1
    b: 2
    c: 3
>>  b = struct('type',true,'color','red','data',[4 5 6])
b =
  struct with fields:

    type: 1
    color: 'red'
    data: [4 5 6]
>>  c.first = 1;
>>  c.second = 2;
```

MATLAB Structs

The fields of a struct are similar to the key values in a Python ‘dict’

```
>>  b = struct('type',true,'color','red','data',[4 5 6]);
>>  fieldnames(b)
ans =
  3×1 cell array
     {'type' }
     {'color'}
     {'data' }
>>  isfield(b,'data')
ans =
  logical
    1
>>  isfield(b,'date')
ans =
  logical
    0
```

More on ‘cell array’ later...

MATLAB Structs

We can access each *field* of a struct using ‘.’ notation

```
>>  b = struct('type',true,'color','red','data',[4 5 6]);
>>  b.type
ans =
  logical
    1
>>  b.color
ans =
    'red'
>>  c.data
ans =
    4    5    6
```

MATLAB Structs

A struct can be indexed just like an array (because that's what it is!)

```
>>  f = dir('*.m');
>>  for fnum = 1:numel(f)
        f(fnum).bytes
    end
ans =
    278
ans =
    44
ans =
    54
```

MATLAB Structs

New fields can also be added to an existing struct

```
>>  f(1).myVariable = true
>>  f(2).myVariable = 5
>>  f.myVariable
ans =
  logical
   1
ans =
     5
ans =
     []
```

Searching Directories

`dir` can also match a specific filename

```
>>  file = dir('data_20190415.mat');
>>  file = dir('data/data_20190415.mat');
>>  file = dir(['data',filesep,'data_20190415.mat']);
>>  file
  struct with fields:

    name: data_20190415.mat'
    folder: '/MATLAB Drive/data'
    date: '15-Apr-2019 21:50:21'
    bytes: 240
    isdir: 0
    datenum: 7.3753e+05
```

← Number of days since the beginning of (Gregorian) time:
Jan. 0th, 0 A.D. @ 00:00

Searching Directories

Find files with an extension by using wildcards

```
>>  files = dir('*.m');
f =
  3×1 struct array with fields:
    name
    folder
    date
    bytes
    isdir
    datenum
>>  f.name
ans =
    'genData.m'
ans =
    'myFunc.m'
ans =
    'test_myFunc.m'
```

Searching Directories

We can be more selective with wildcards:

```
>>  files = dir('data_2019*.mat');
>>  files = dir('*.m*');
```

What if we want to be even more specific?

Find `'data_20190415.mat'` and `'data_20190521.mat'`, but not `'data_2019_aux.mat'`?

Wildcards can’t do this… sad :(

Regular Expressions

There is a better way! Enter Regular Expressions…

https://xkcd.com/208/

Regular Expressions

Regular Expressions …

```
>>  help regexp
regexp Match regular expression
    S = regexp(STR,EXPRESSION) matches the regular
    expression, EXPRESSION, in the input argument, STR. The
    indices of the beginning of the matches are returned.
    ...

>>  help regexpi
regexpi Match regular expression, ignoring case
    START = regexpi(STR,EXPRESSION) matches the regular
    expression, EXPRESSION, in the input argument, STR,
    regardless of case. The indices of the beginning of the
    matches are returned.

See also:
```

https://www.regular-expressions.info

Regular Expressions

Examples:

```
>>  str = 'bat cat can car coat court cut ct caoueouat';
>>  regexp(str, 'can')
ans =
     9
>>  regexp(str, 'c[aeiou]+t')
ans =
     5    17    28    35

>>  regexp(str, 'Co[\w].')
ans =
     []
>>  regexpi(str, 'Co[\w].')
ans =
     17   22
```

Regular Expressions: Metacharacters

Expression	Usage
*	Matches the preceding element 0 or more times. Equivalent regular expression: {0,}
+	Matches the preceding element 1 or more times. Equivalent regular expression: {1,}
?	Matches the preceding element 0 times or 1 time, also minimizes. Equivalent regular expression: {0,1}
{n,m}	Must occur at least n times but no more than m times
{n,}	Must occur at least n times.
{n}	Must match exactly n times. Equivalent regular expression: {n,n}

Regular Expressions: Logical Operators

Expression	Usage
`(...)`	Groups regular expressions.
|	Matches either the expression preceding or following it.
`^`	Matches following expression only at the beginning of the string.
`$`	Matches preceding expression only at the end of the string.
`\<chars`	Matches the characters when they start a word.
`chars\>`	Matches the characters when they end a word.
`\<word\>`	Exact word match.

Regular Expressions: Quantifiers

Expression	Usage
.	Matches any single character
[ab...]	Matches any one of the characters, (a, b, etc.), contained within the brackets
[^ab...]	Matches any character except those contained within the brackets, (a, b, etc.).
[c_1-c_2]	Matches any characters in the range of c_1 through c_2.
\f	Form feed.
\n	New line.
\r	Carriage return.
\t	Tab.
\d	A digit. Equivalent regular expression: [0-9]
\D	A nondigit. Equivalent regular expression: [^0-9]
\s	A whitespace character. Equivalent regular expression: [\f\n\r\t]
\S	A non-whitespace character. Equivalent regular expression: [^ \f\n\r\t]
\w	A word character. Equivalent regular expression: [a-zA-Z_0-9]
\W	A nonword character. Equivalent regular expression: [^a-zA-Z_0-9]
\	If a character has special meaning in a regular expression, precede it with this character to match it literally.

What is the output of the following code?

```
>>  regexp('acoueoouat', 'c[aeiou]+t')
```

A) 1
B) 2
C) [1 9]
D) None of the above

What is the output of the following code?

```
>>  regexp('acoueouat', 'c[aeiou]+t')
```

A) 1
B) 2
C) [1 9]
D) None of the above

What is the output of the following code?

```
>>  regexp('aoueouat', 'c[aeiou]+t')
```

A) 1
B) 2
C) [1 9]
D) None of the above

What is the output of the following code?

```
>>  regexp('aoueouat', 'c[aeiou]+t')
```

A) 1
B) 2
C) [1 9]
D) None of the above

What is the output of the following code?

```
>>  regexp('aoueouat', 'c|[aeiou]+t')
```

A) 1
B) 2
C) [1 9]
D) None of the above

What is the output of the following code?

```
>>  regexp('aoueouat', 'c|[aeiou]+t')
```

A) 1
B) 2
C) [1 9]
D) None of the above

What is the output of the following code?

```
>>  regexp('puppy54.mat', '[\d]+')
```

A) 1
B) 6
C) [6 7]
D) None of the above

What is the output of the following code?

```
>>  regexp('puppy54.mat', '[\d]+')
```

A) 1
B) 6
C) [6 7]
D) None of the above

Saving Variables Using Regular Expressions

Regular Expressions can also be applied when saving variables from the workspace to .mat file

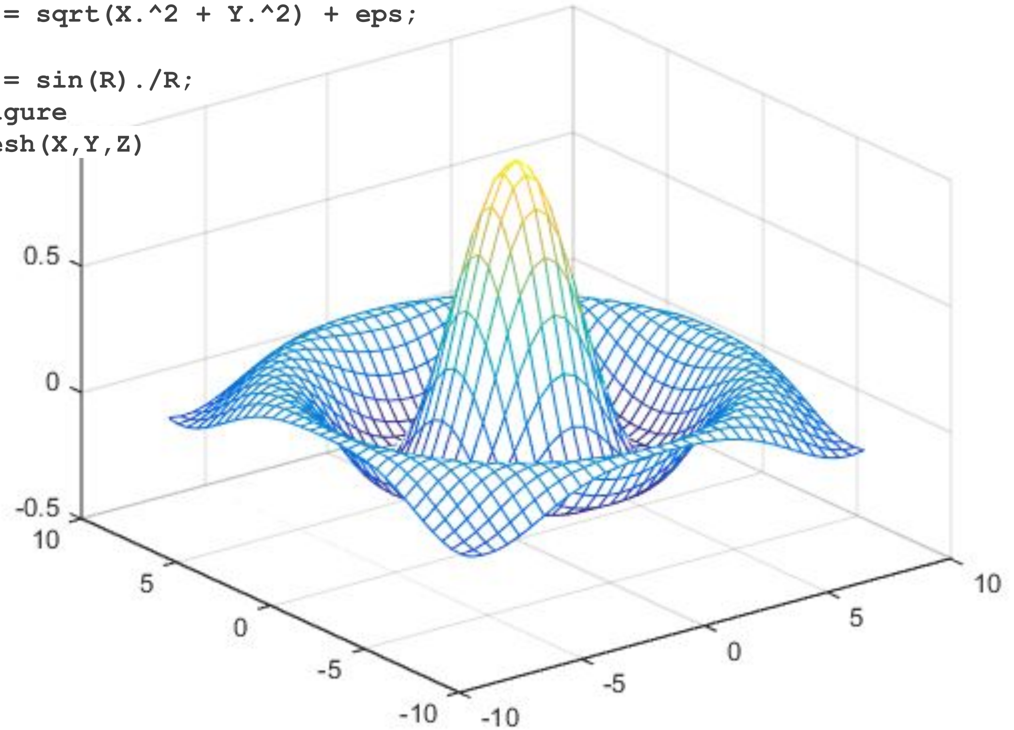
```
>>  save myfile.mat -regexp \d
>>  save myfile.mat -regexp ^data\_2019(04|05)
```

What do these regular expressions capture?

2D Data Visualization

```
[X,Y] = meshgrid(-8:.5:8);
R = sqrt(X.^2 + Y.^2) + eps;

Z = sin(R)./R;
figure
mesh(X,Y,Z)
```



2D Data Visualization

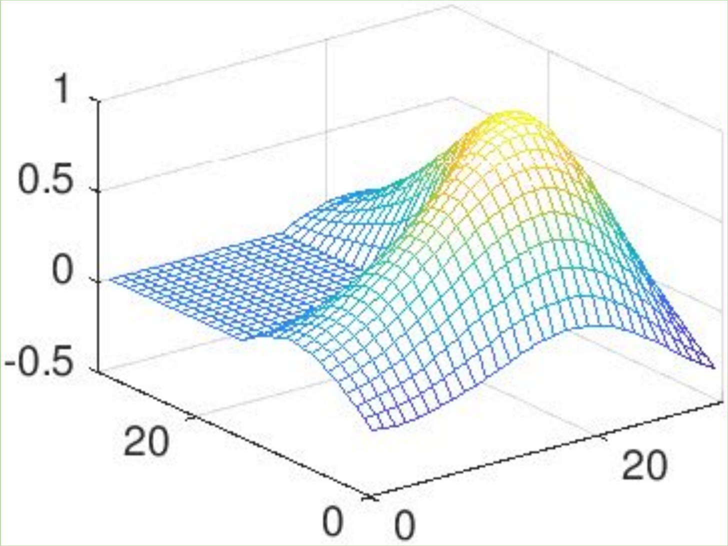
MATLAB contains various built-in 2D plotting tools:

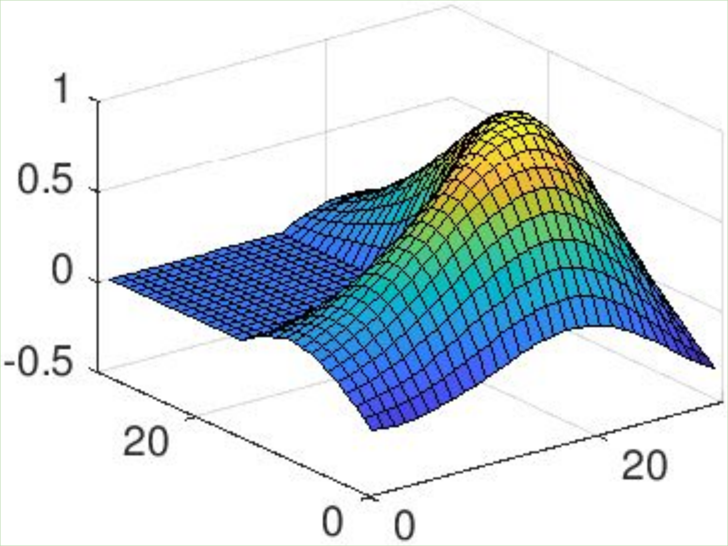
mesh, **surf** - Surface plot
meshc, **surfc** - Surface plot with contour plot beneath it
meshz - Surface plot with curtain plot (reference plane)
waterfall - Mesh plot, but without column lines
pcolor - Flat surface plot (value is proportional only to color)
surfl - Surface plot illuminated from specified direction
image, **imagesc** - x,y image plot
contour, contourf - Flat plot with equal isolines

*All of these plot types use *color* to represent each x,y value

Mesh vs. Surf

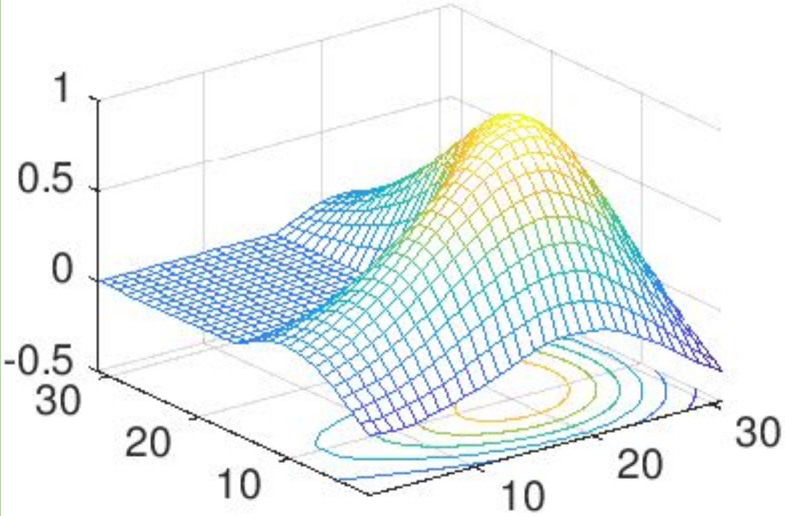
```
>> mesh(membrane)
>> surf(membrane)
```

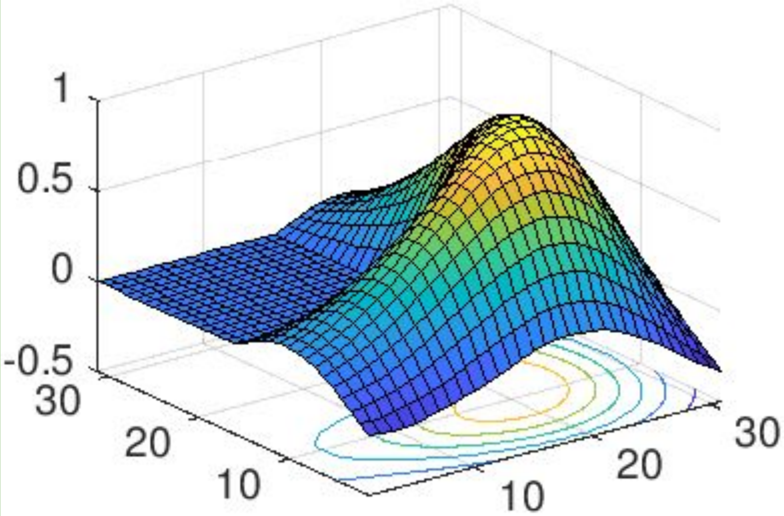




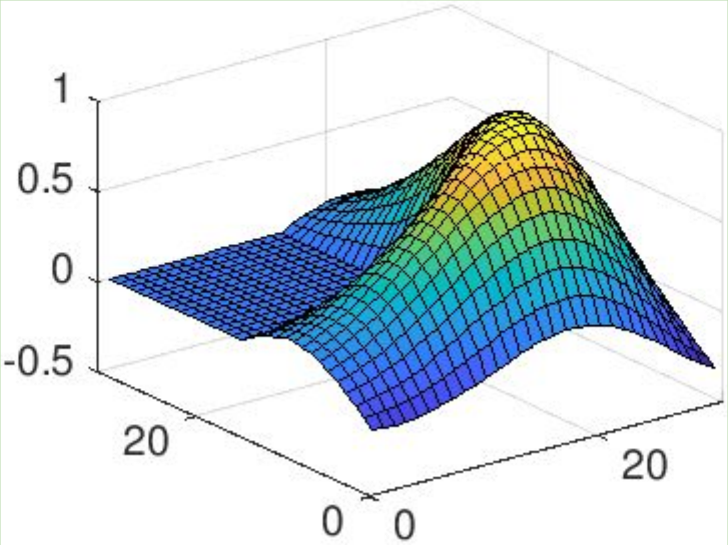
Meshc vs. Surfc

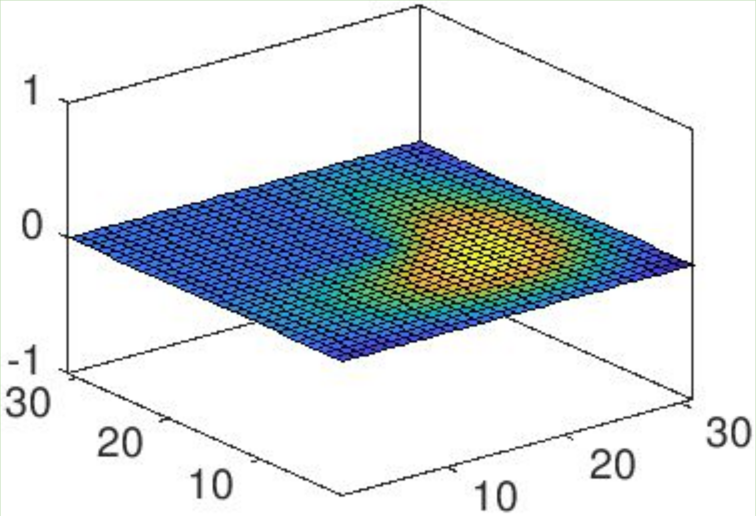
```
>> meshc(membrane)
>> surfc(membrane)
```

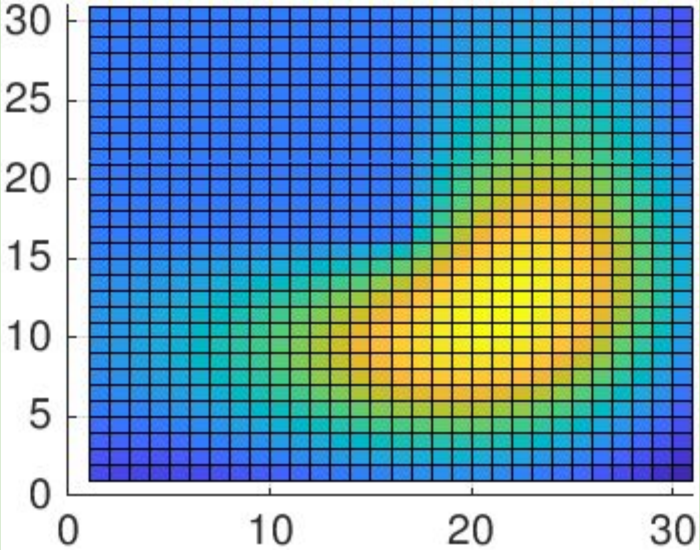




Surf vs. Pcolor







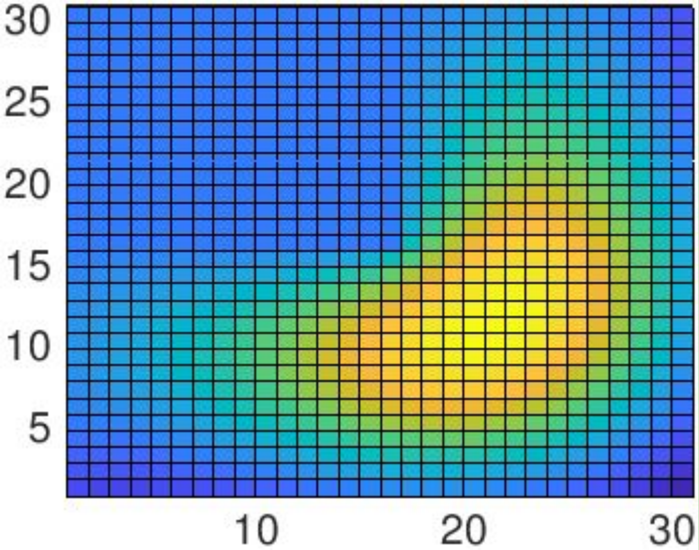
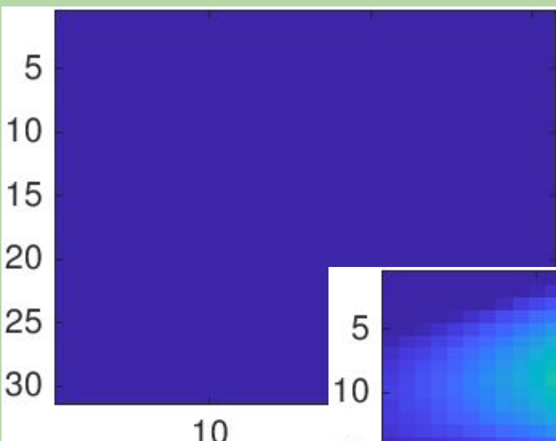
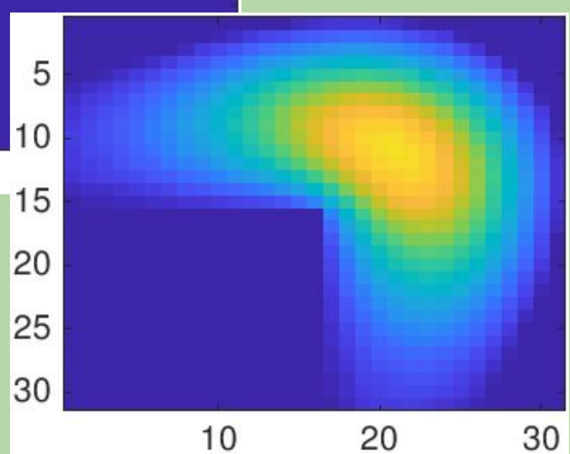
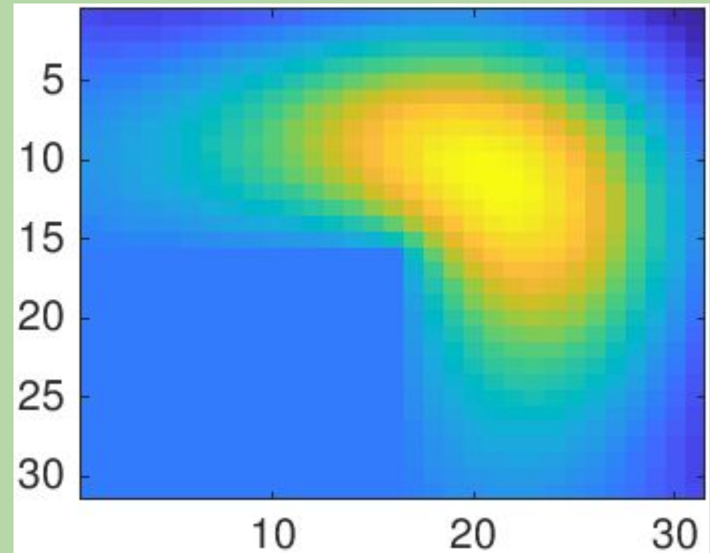


Image vs. Imagesc

```
>> image(membrane)
>> image(membrane*60)
>> imagesc(membrane)
```

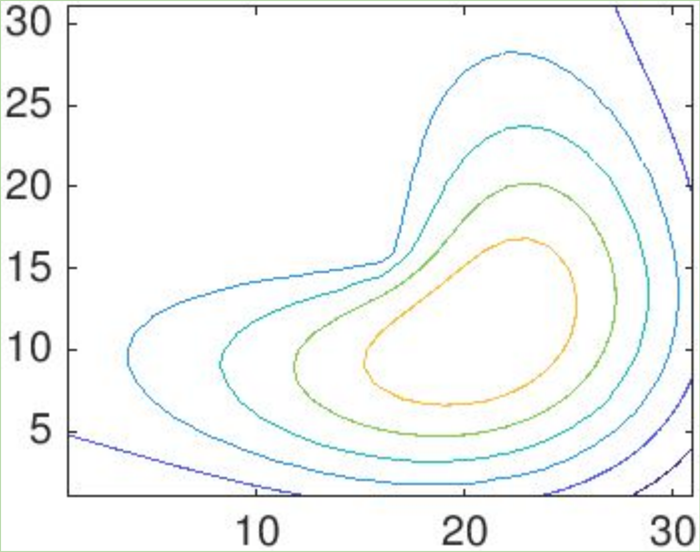


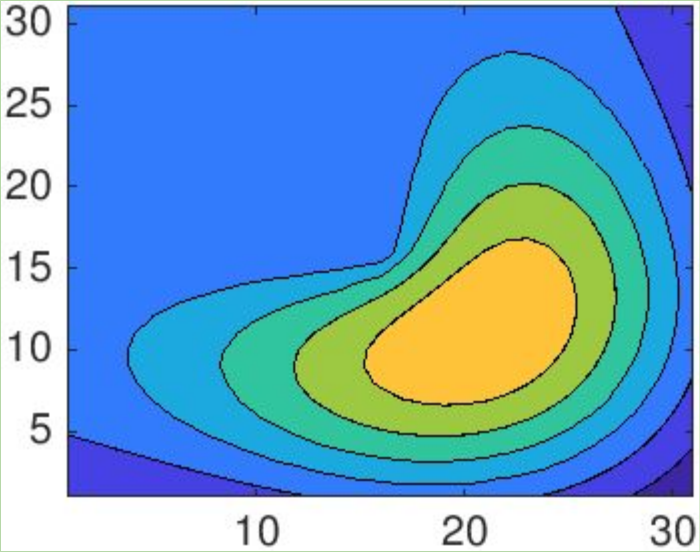




Contour vs. Contourf

```
>> contour(membrane)
>> contourf(membrane)
```

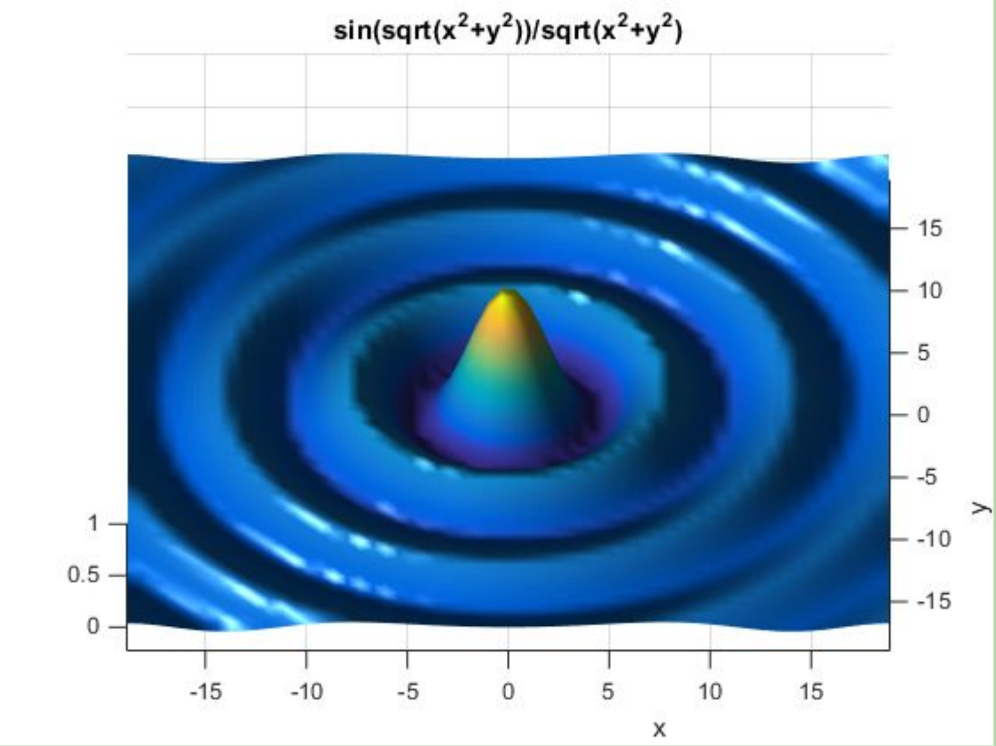




Lighting and Shading

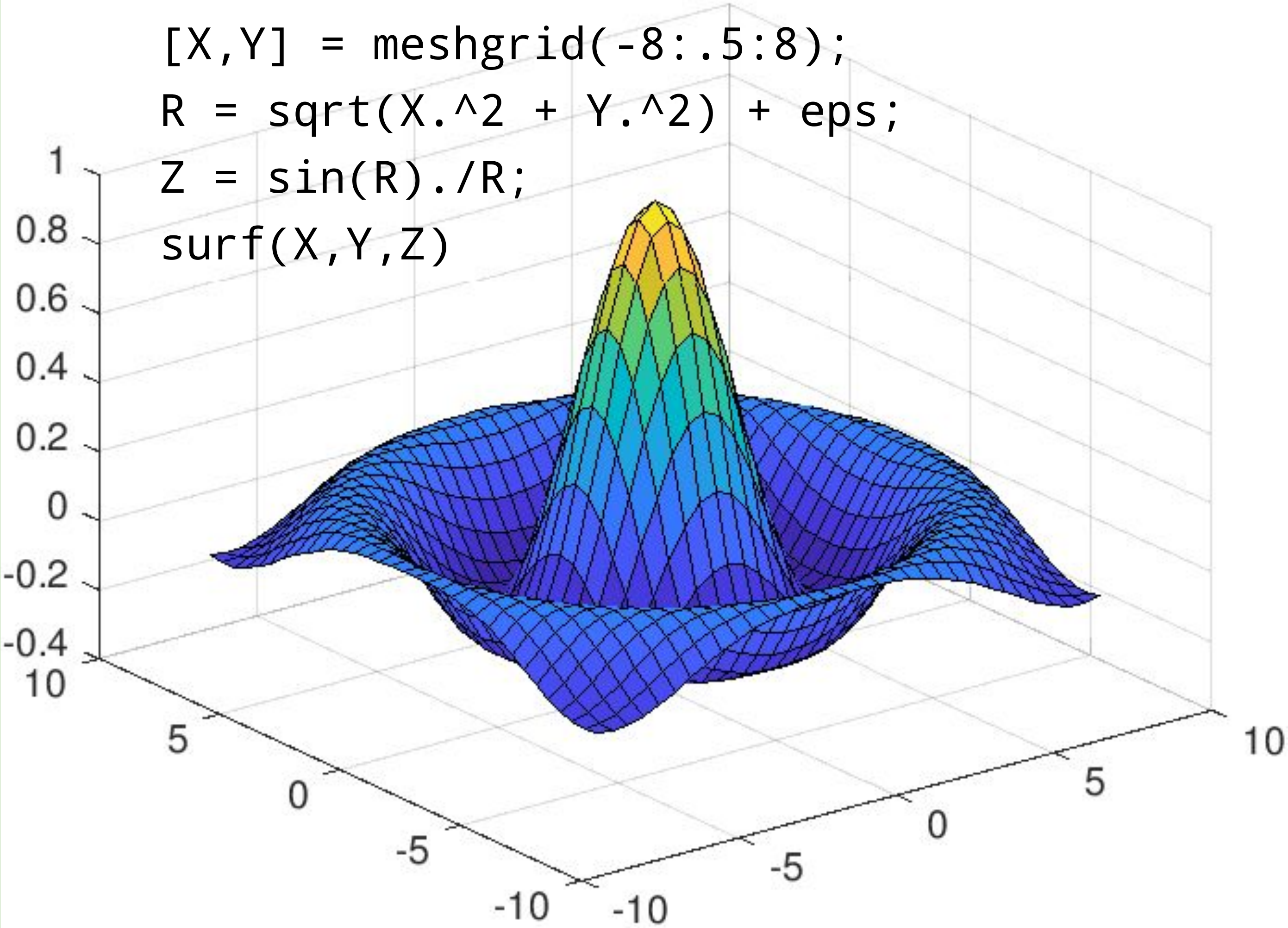
Lighting can help add the “finishing touches” for data visualization

Helps add depth perception to **surface** and **patch** objects



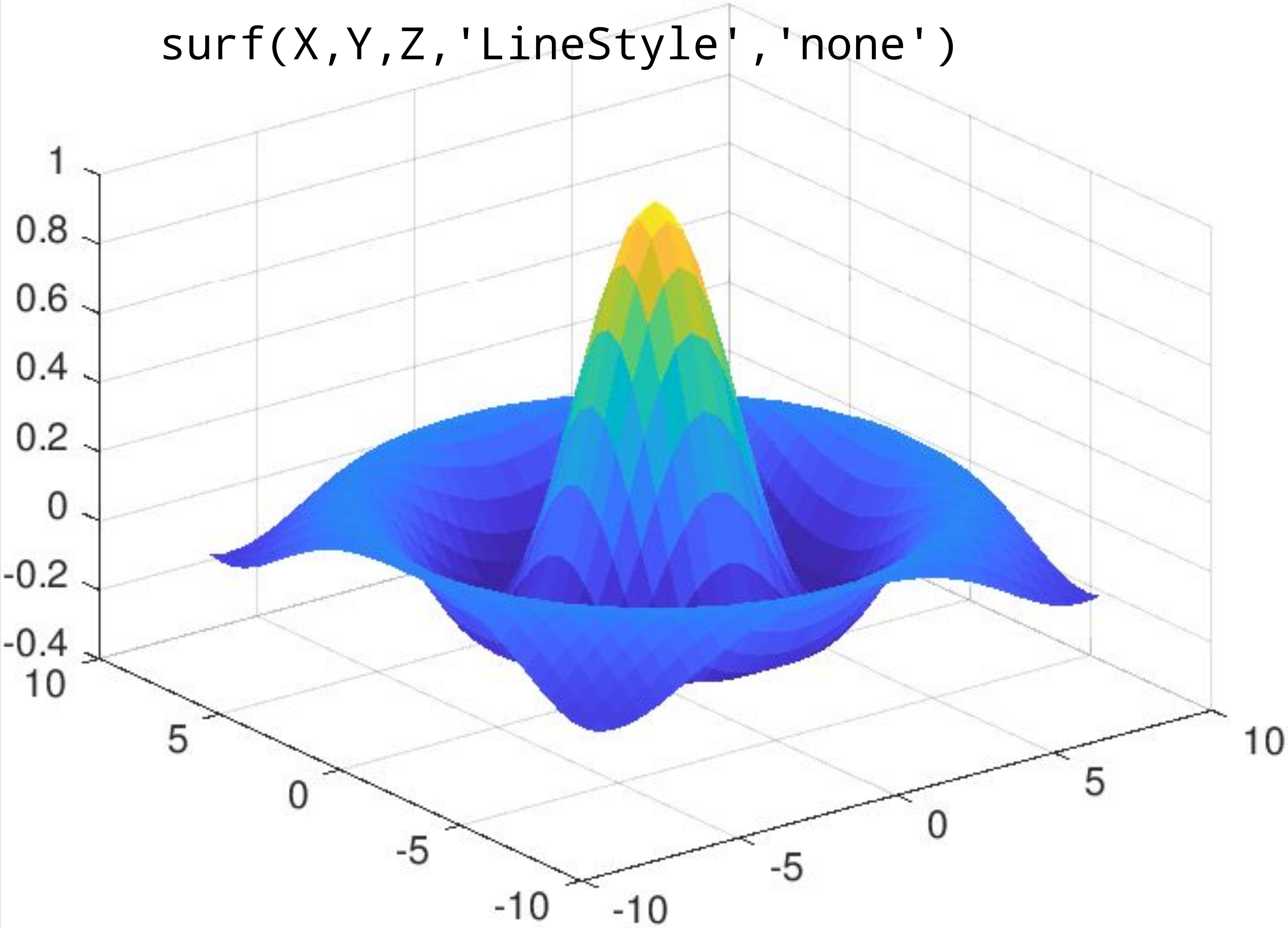
Lighting and Shading

```
[X,Y] = meshgrid(-8:.5:8);
R = sqrt(X.^2 + Y.^2) + eps;
Z = sin(R)./R;
surf(X,Y,Z)
```



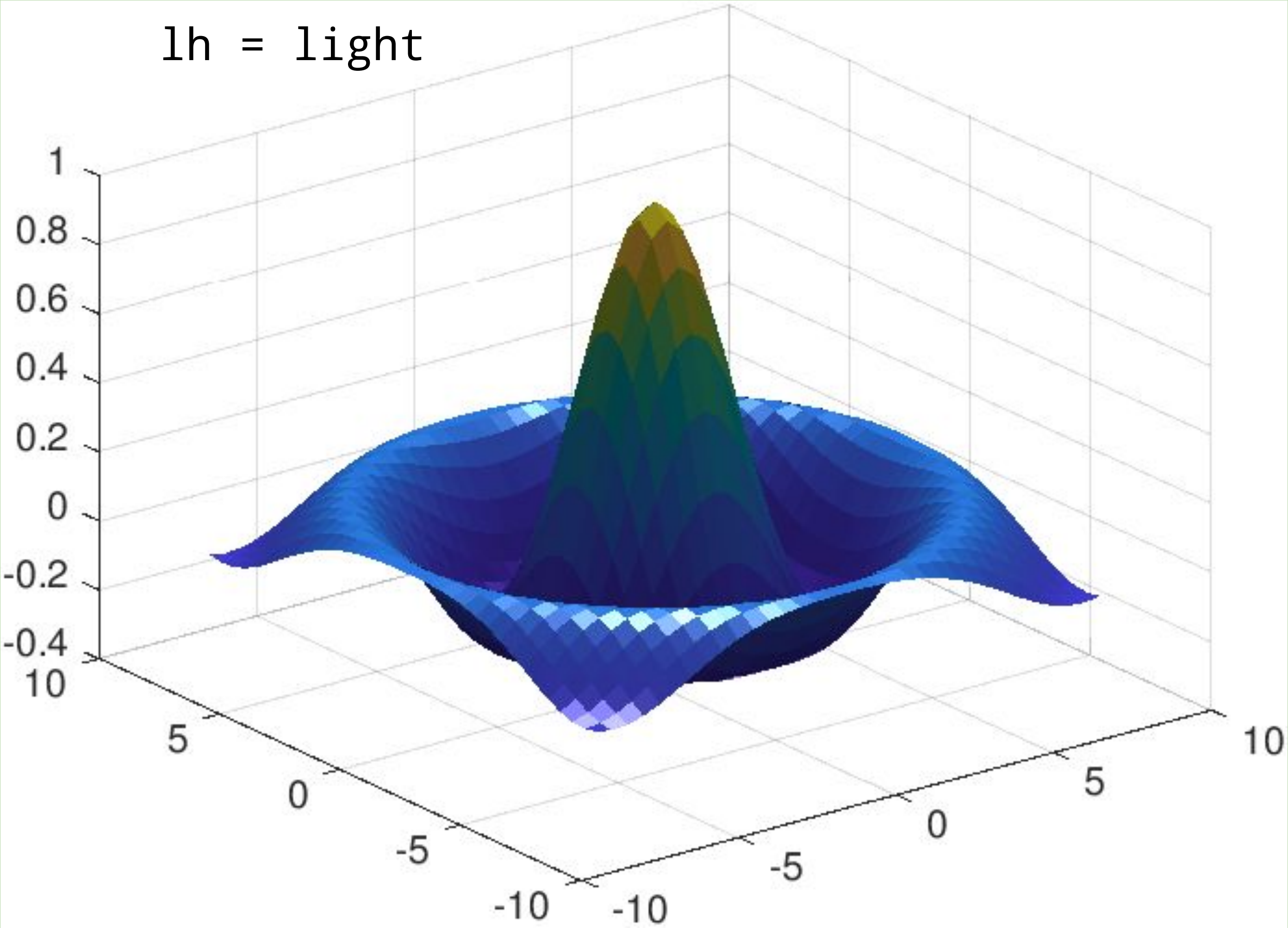
Note: eps is used to avoid div by 0

Lighting and Shading



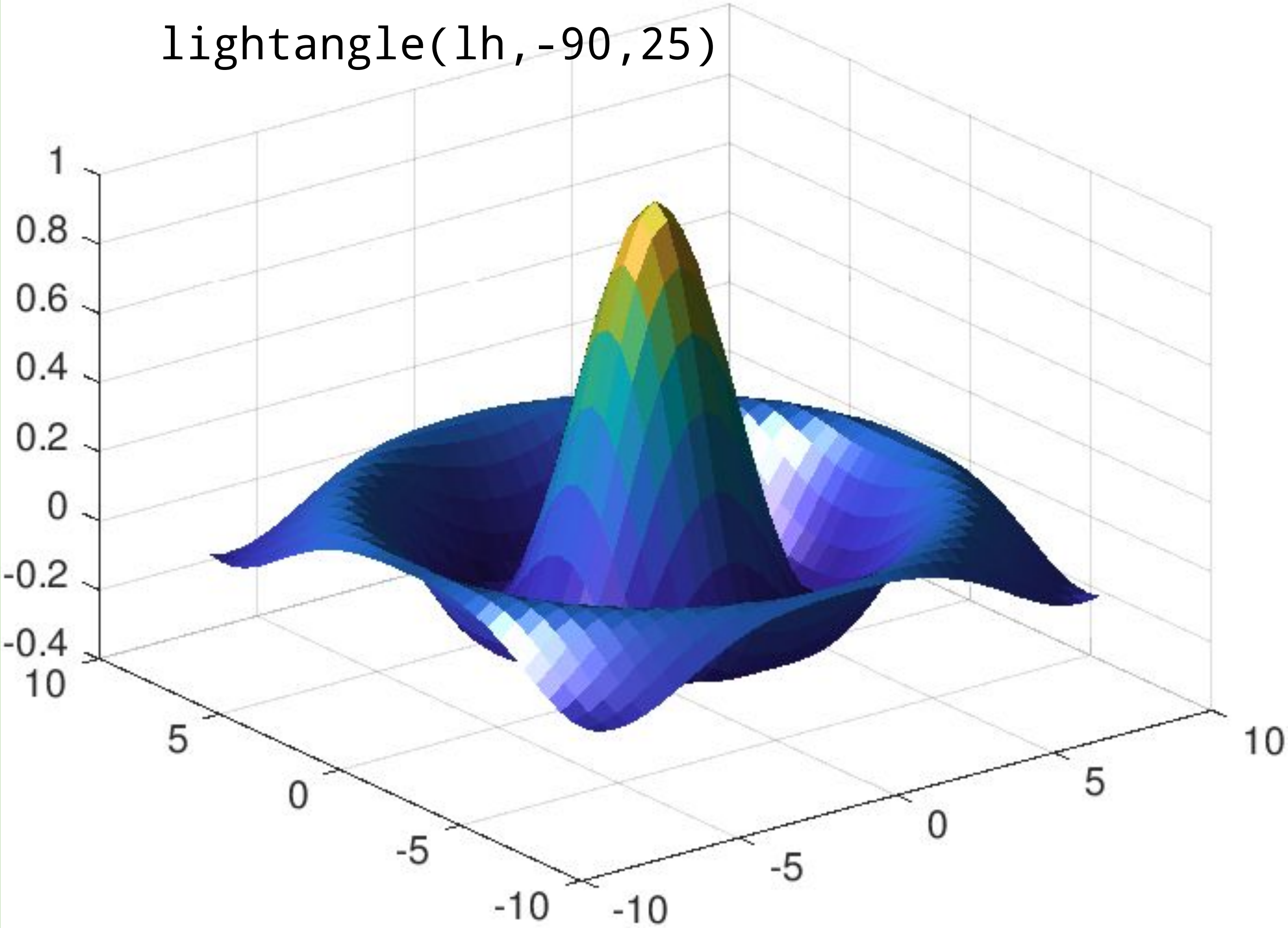
Lines are optional

Lighting and Shading



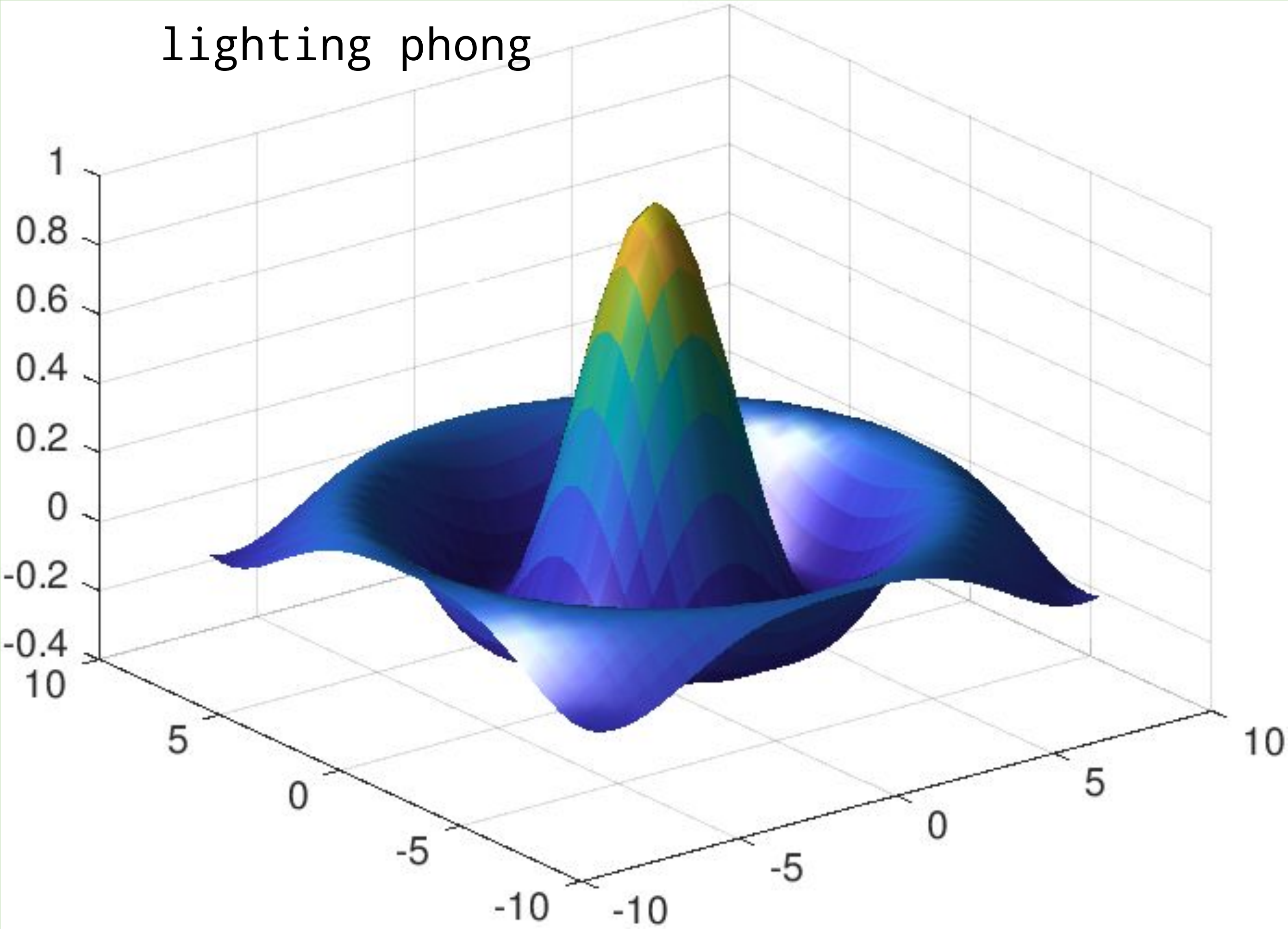
We can add lighting effects!

Lighting and Shading



Move the light source to change the effect

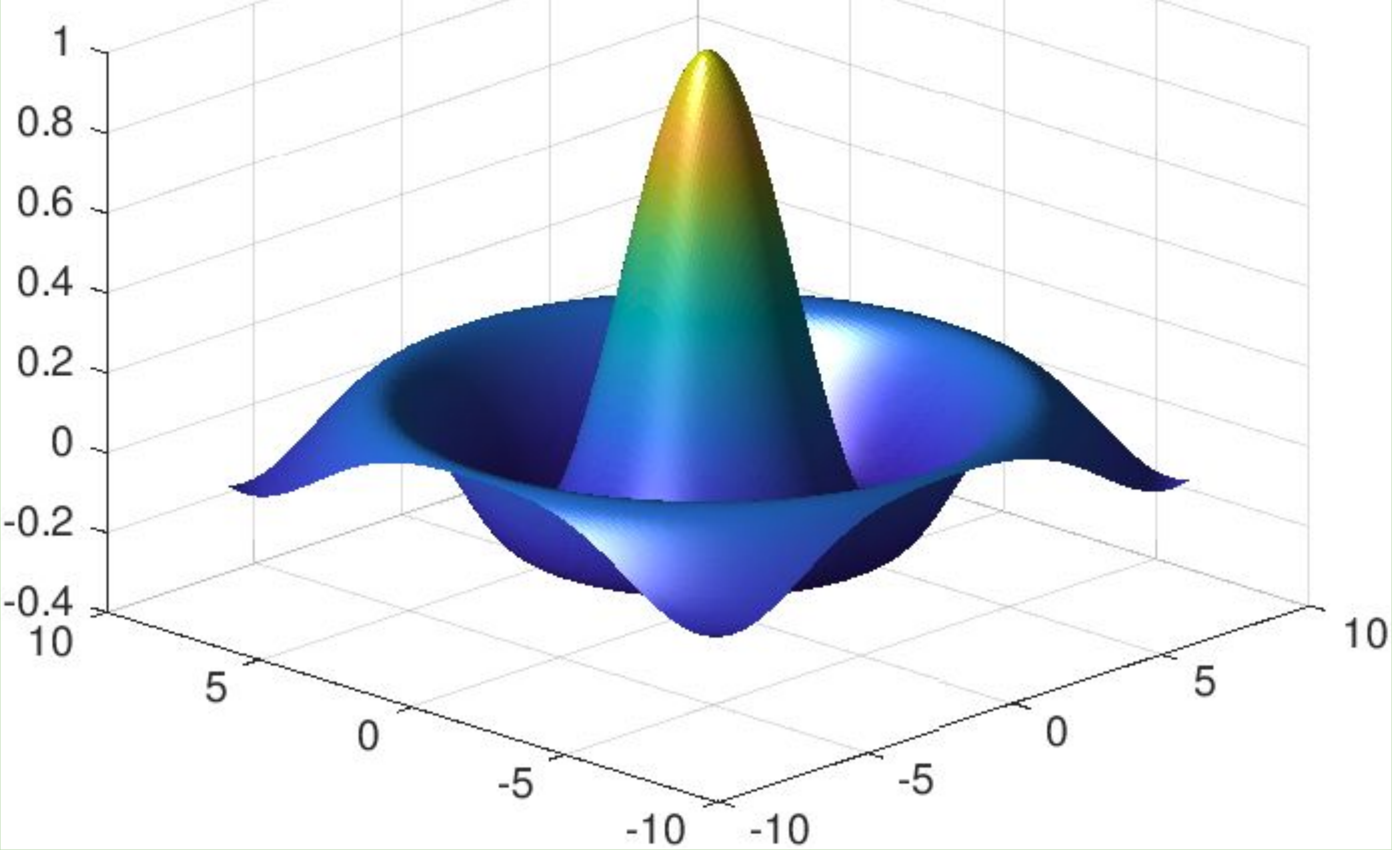
Lighting and Shading



Lighting algorithms can shade in each data point

Lighting and Shading

```
[X,Y] = meshgrid(-8:.1:8);
...
```



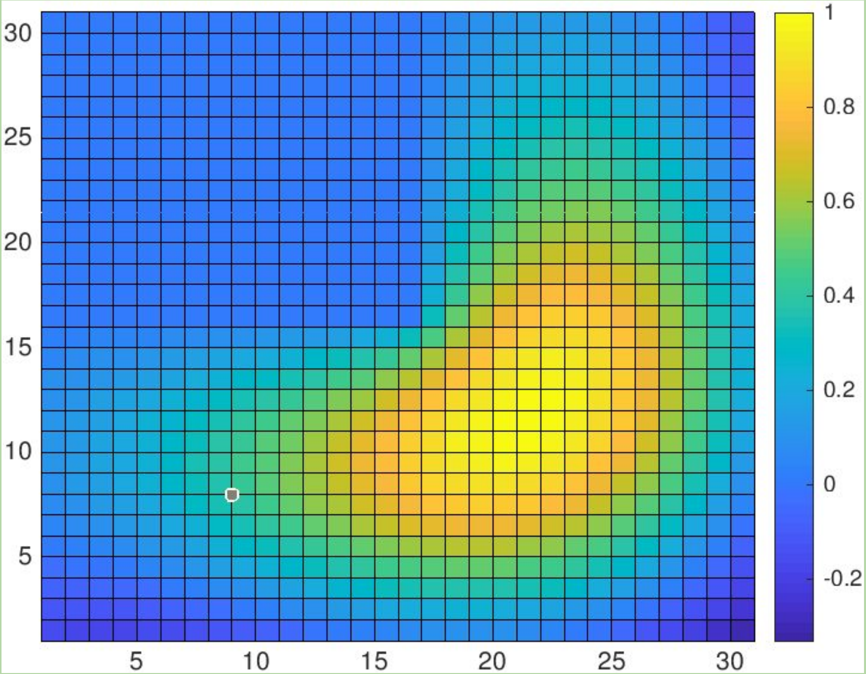
Higher resolution results in smoother looking images

Colorbar and 'colormap'

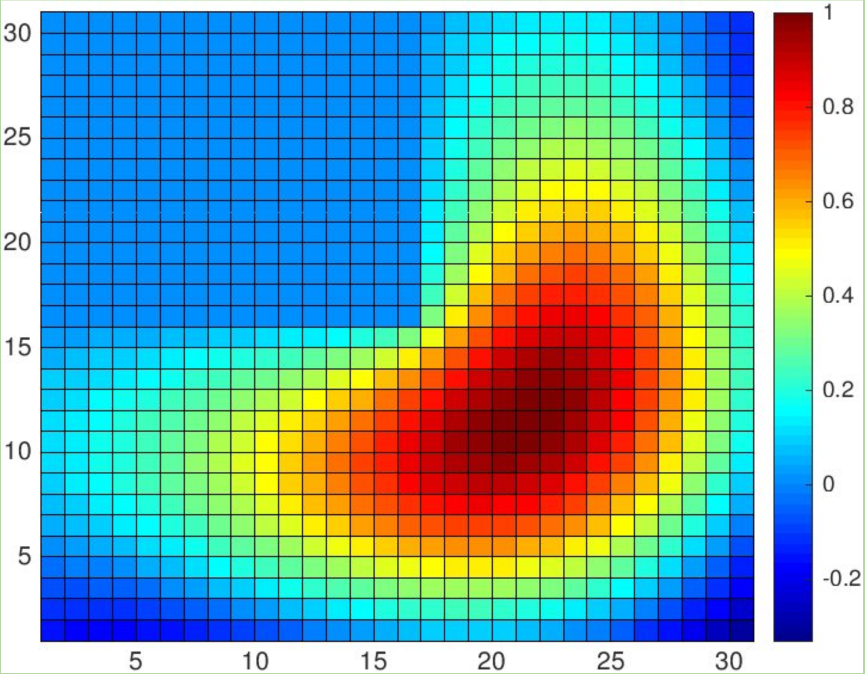
```
>>  pcolor(membrane)
>>  colorbar
```

We can select a new colormap to emphasize certain regions:

`colormap(parula)`



`colormap(jet)`

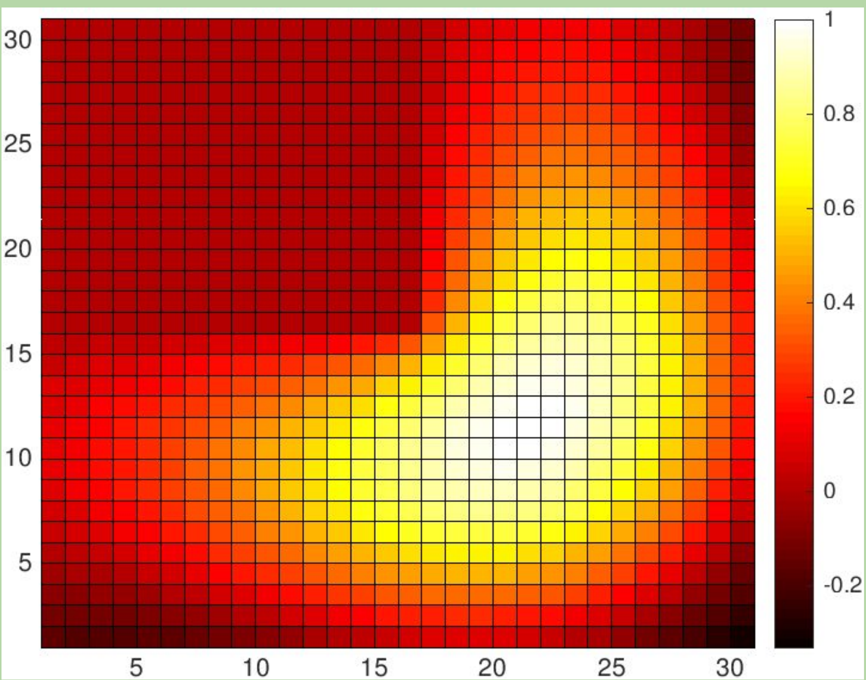


Colorbar and 'colormap'

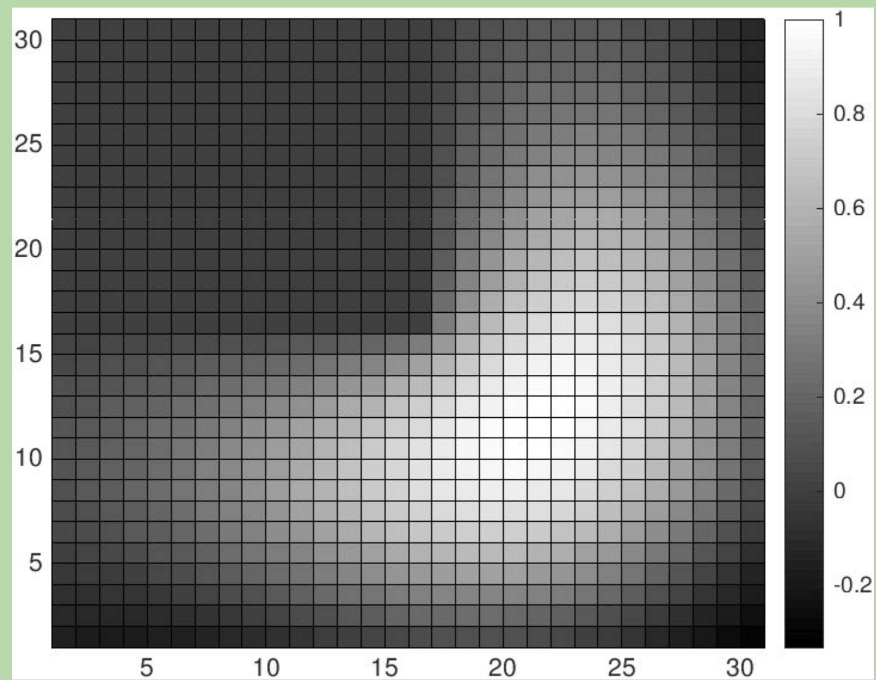
```
>>  pcolor(membrane)
>>  colorbar
```

We can select a new colormap to emphasize certain regions:

`colormap(hot)`



`colormap(gray)`

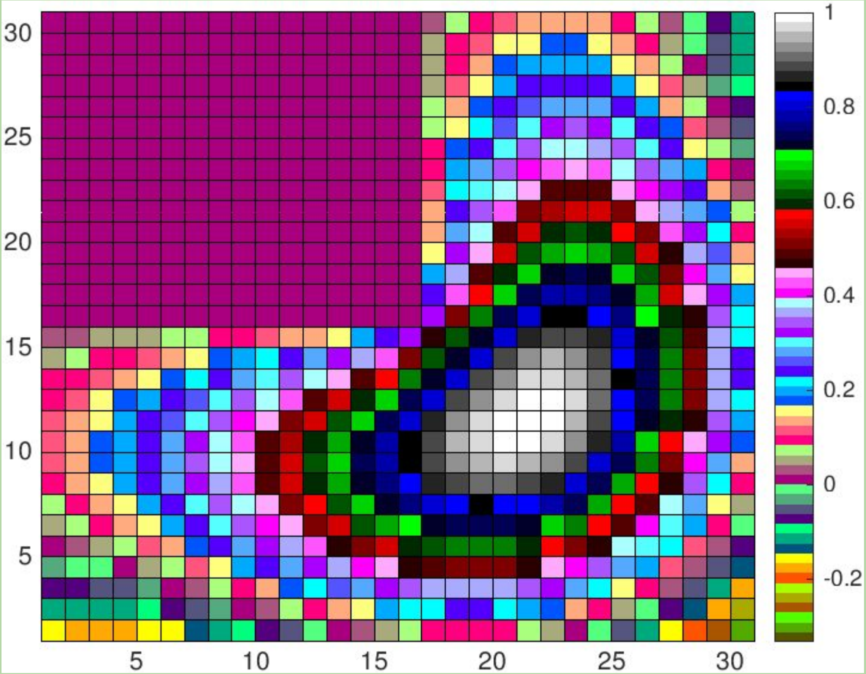


Colorbar and 'colormap'

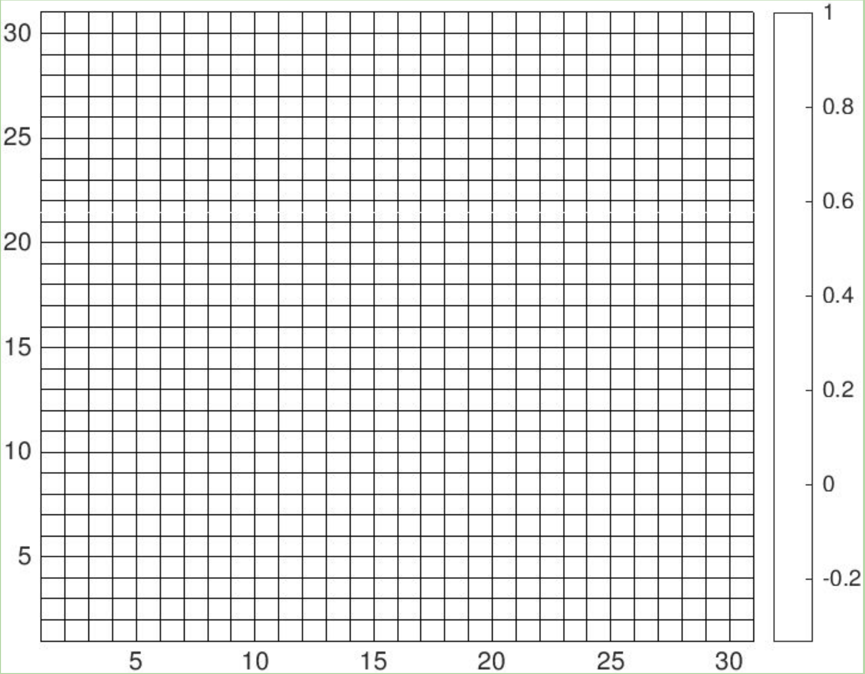
```
>>  pcolor(membrane)
>>  colorbar
```

We can select a new colormap to emphasize certain regions:

`colormap(colorcube)`



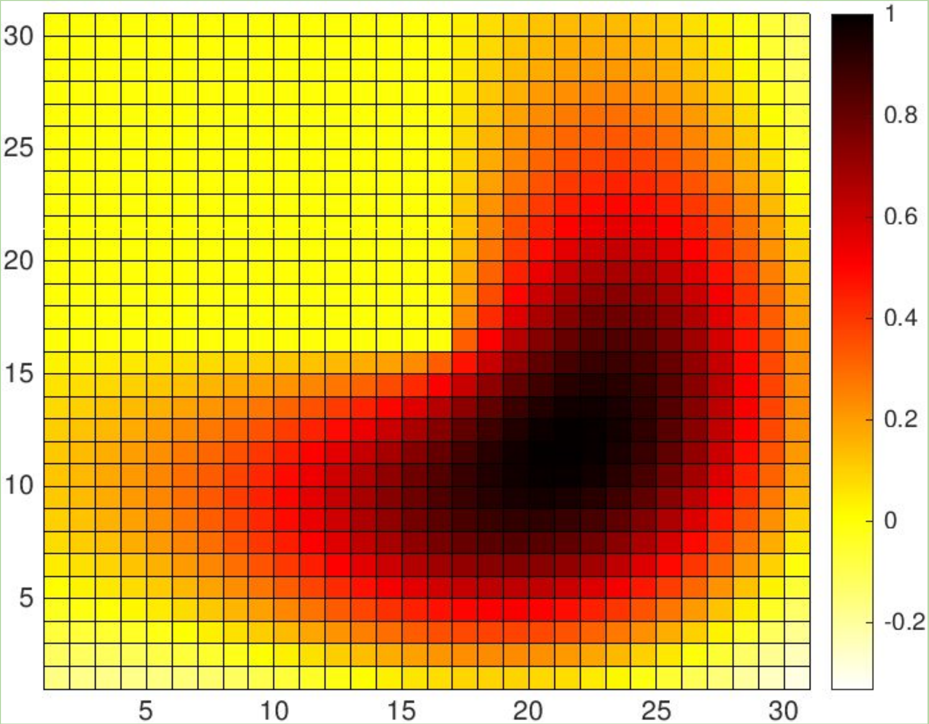
`colormap(white)`



Color Scale and ‘colormap’

We can manipulate the color map manually (using RGB values) or create our own colormaps

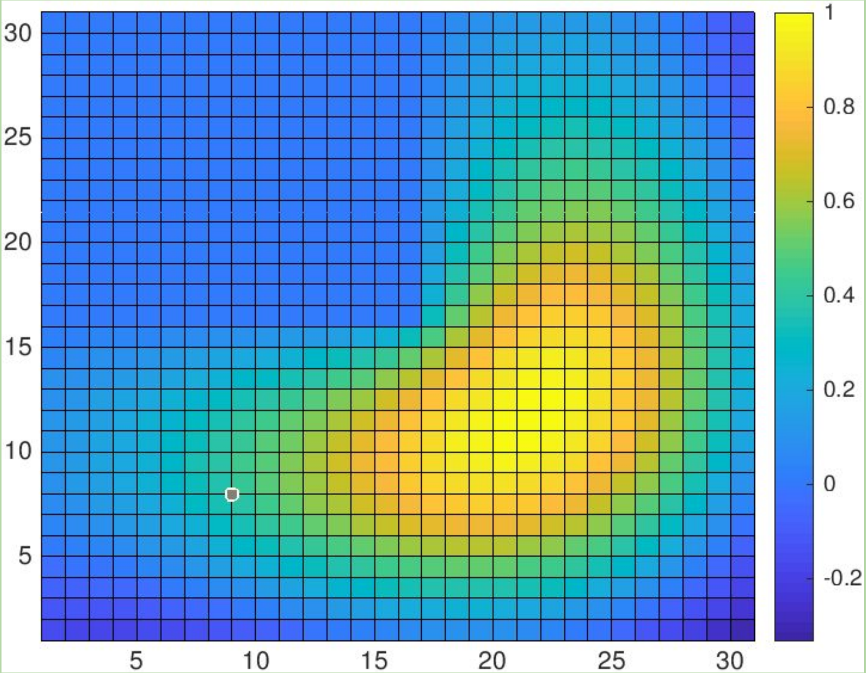
```
>> c = hot(255);     % use 255 levels in colormap
>> c = flipud(c);    % invert colormap
>> colormap(c);      % set colormap to inverted hot
```



Color Scale and Dynamic Range

By default, MATLAB will set the color scale to fit to the **dynamic range** of the data. Sometimes, the scales aren't what we would like.

```
cLim = [-0.3338, 1.0000]
```



Color Scale and Dynamic Range

Changing the dynamic range can be accomplished by manipulating the data (ok) or changing the color scale (better)

```
colorbar                          % turn colorbar on
cLim = get(gca,'cLim');    % get current scale
set(gca,'cLim',[cLim(2)-1.0 cLim(2)]) % colormap limits [min max]
```

```
cLim = [0, 1]
```