# Intro Math Problem Solving
November 7

Contour and Surface Plots

A Temperature Field Contour

Slicing the Data

A Model of Cooling

Animating Time Data

# Final Project

I have posted information about the final project in the Canvas directory final_project. It looks something like a homework set, except somewhat longer and harder.

The final project should be turned in by midnight, 12 December, our last class meeting day.

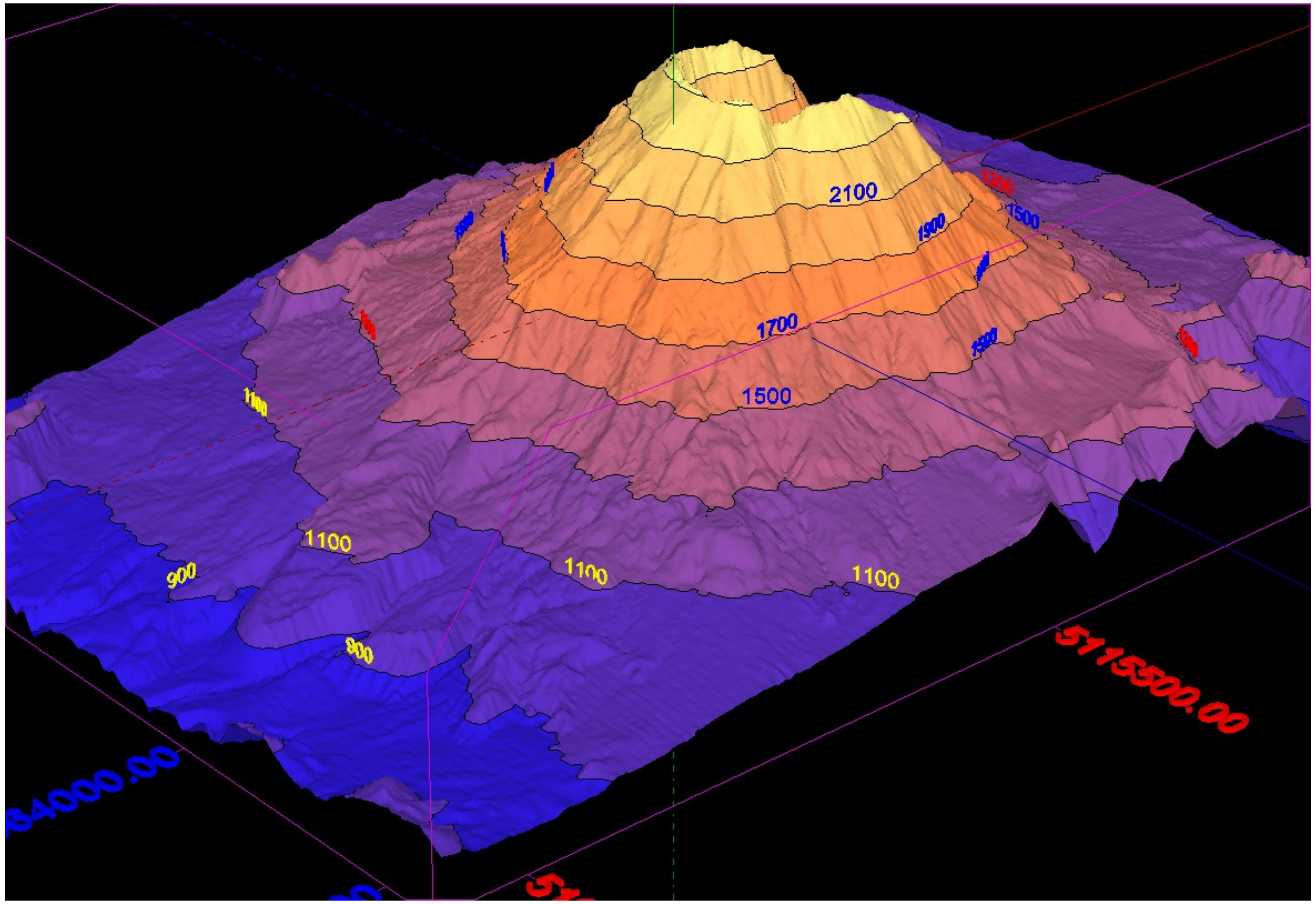
If you prefer to do a personalized final project, you must tell me so before November 16th (last class before Thanksgiving break). Suggested topics are in the projects directory.

## Reference

Chapter 7, Section 2 and 3 of our textbook "Insight Through Computing" discuss the way way a temperature problem can be set up on a grid of points, and displayed with a contour plot.

.

# Contour and Surface Plots



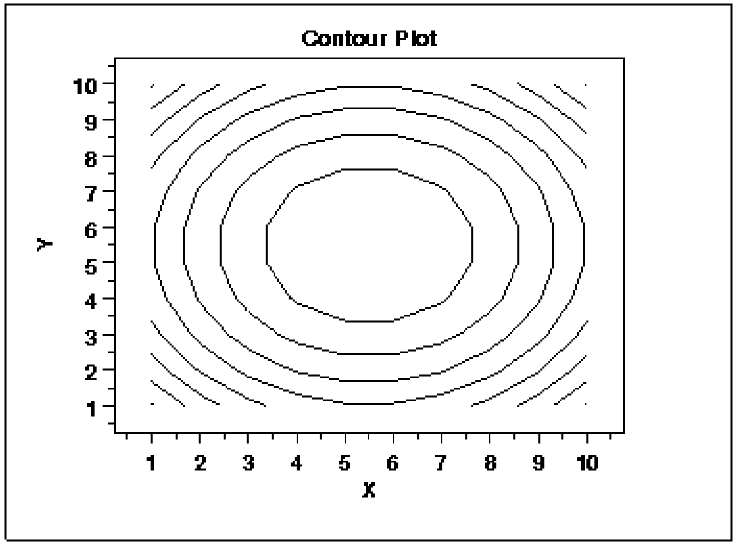
## Contours: 3D data on 2D Paper

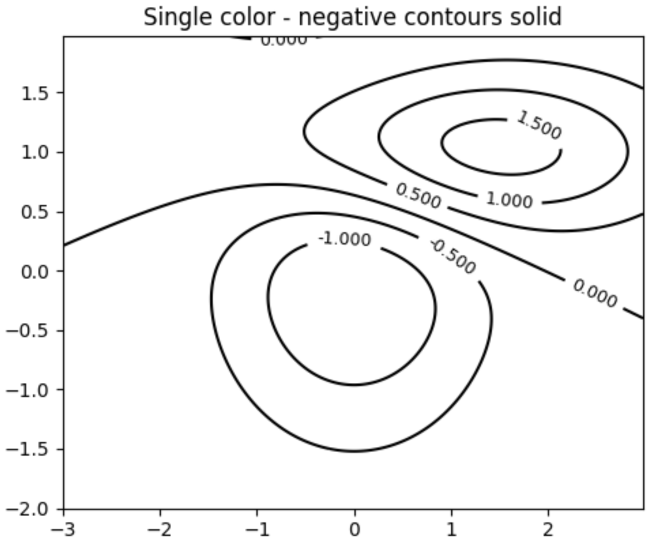
The first contour maps were used to record the variations in elevation in mountainous areas.  Starting from a standard “flat” map, careful measurements of height were made over the regions.  All points at a height of 100 feet were connected by a curve, then 200 feet, 300 feet, and so on.

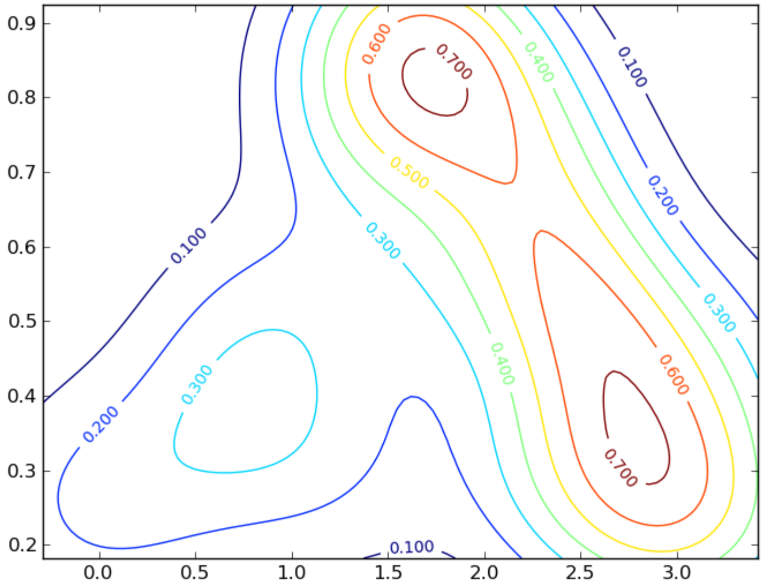
The resulting map gave enough information to determine how to cross a mountain, or follow a stream bed, or find high points.

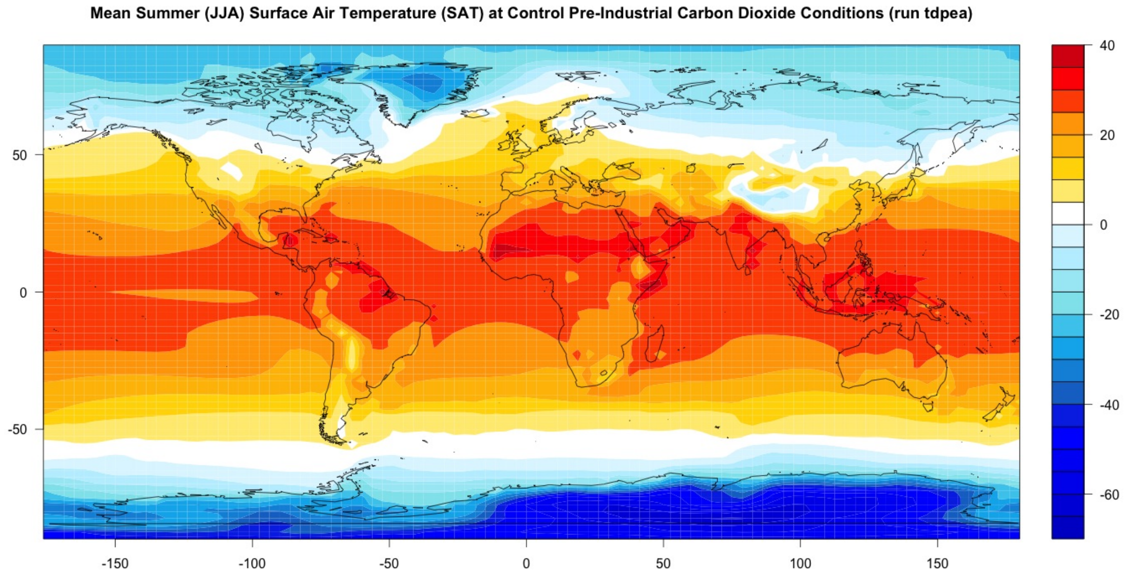
If we think of the underlying “flat map” as using (X,Y) coordinates, then the contour map is giving us a way to record a Z coordinate of height.

# No Labels, Labels, Colors, Filled








Mean Summer (JJA) Surface Air Temperature (SAT) at Control Pre-Industrial Carbon Dioxide Conditions (run tdpea)

# Contours for any old function

The MATLAB plot() command helps us visualize simple functions of the form y=f(x), but we need a way to visualize functions of two variables:

$$z = f(x,y)$$

Think of $z$ as an elevation, that depends on the coordinates $x$ and $y$. We can draw contours of the "hills" and "valleys" of this function.

Our eyes will be able to understand this information and register any patterns that emerge.

# MATLAB Contours

MATLAB includes functions contour(), contour3(), contourf(), and surf() for these plots, along with useful functions like colorbar() and clabel().

If the data is on an x, y grid, we prepare by setting:

```
x = linspace ( xlo, xhi, n );  % <- vector of x’s
y = linspace ( ylo, yhi, m ); % <- vector of y’s
[ X, Y ] = meshgrid ( x, y ); % <-  X and Y matrixes
Z = f(X,Y);     % <- Create matrix of Z values.
```

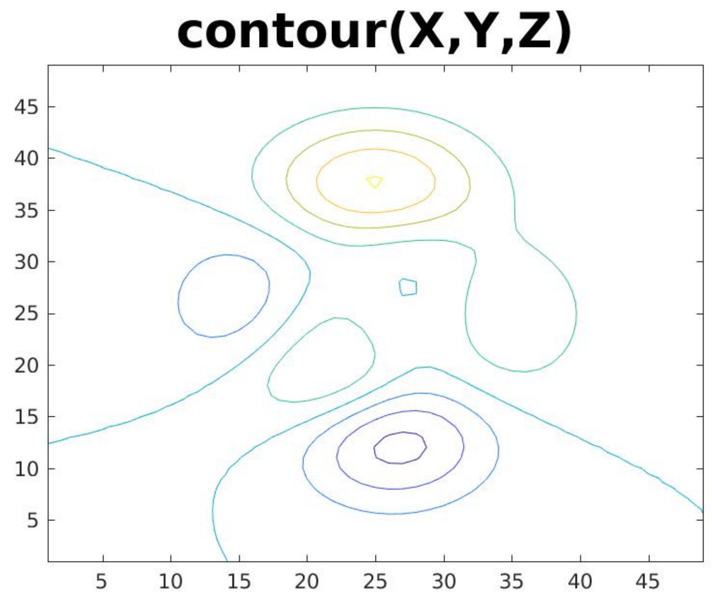
# Example Data Setup

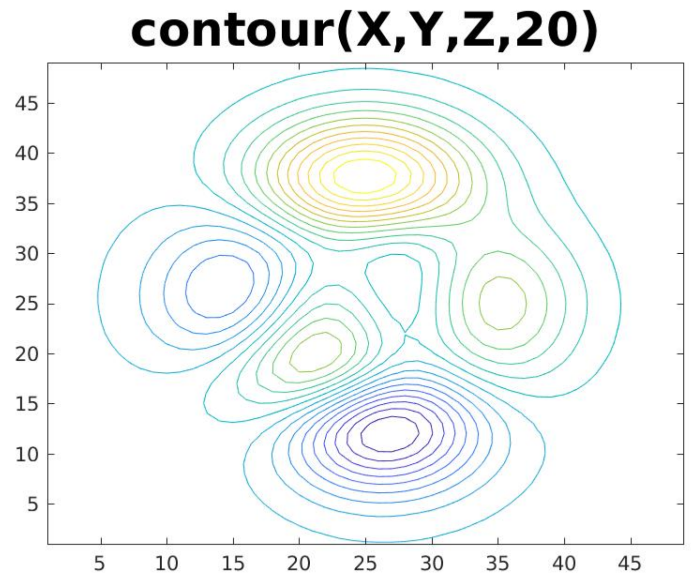
```
m = 49;
n = 49;
x = 1:n;
y = linspace ( 1, m, m );
[ X, Y ] = meshgrid ( x, y );
Z = peaks ( X, Y );   % <- Built in MATLAB function
```

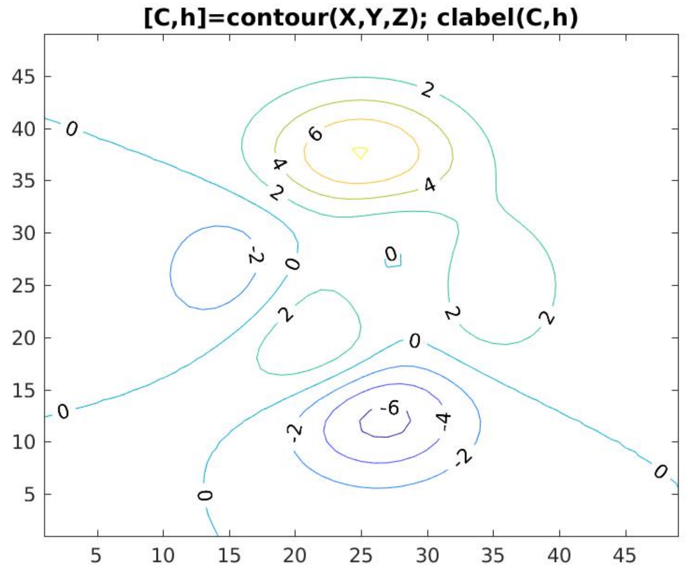
# Example plots 1, 2, 3, 4

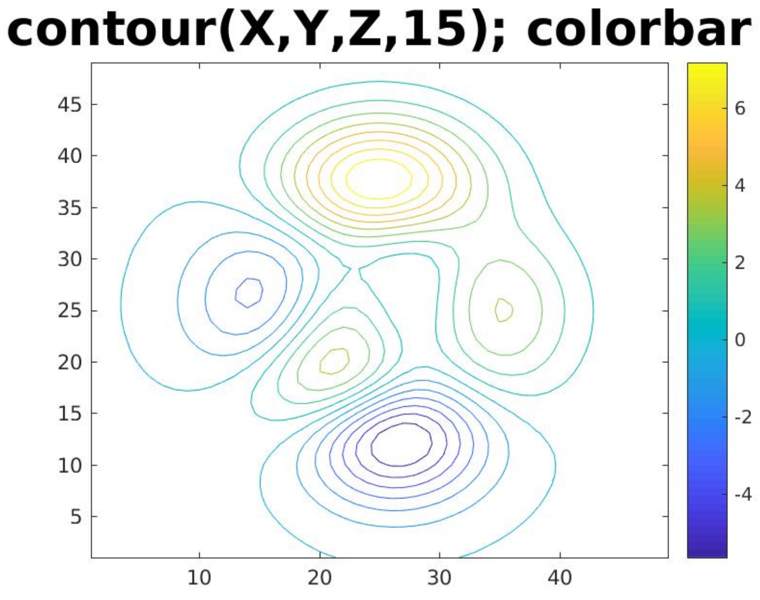
```
% Make a basic contour plot.

contour ( X, Y, Z )
title ( 'contour(X,Y,Z)','FontSize', 24 )

%  Request 20 contour levels.

contour ( X, Y, Z, 20 )
title ( 'contour(X,Y,Z,20)','FontSize', 24 )

%  Request labels for contour lines.

[ C, h ] = contour ( X, Y, Z );
clabel ( C, h )
title ( '[C,h]=contour(X,Y,Z); clabel(C,h)','FontSize', 12 )

%  Include a color bar.

contour ( X, Y, Z, 15 )
colorbar()
title ( 'contour(X,Y,Z,15); colorbar','FontSize', 24 )
```

# Examples 1, 2, 3, 4



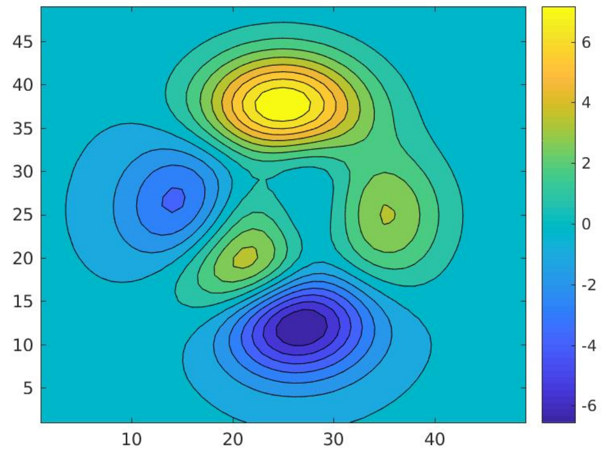




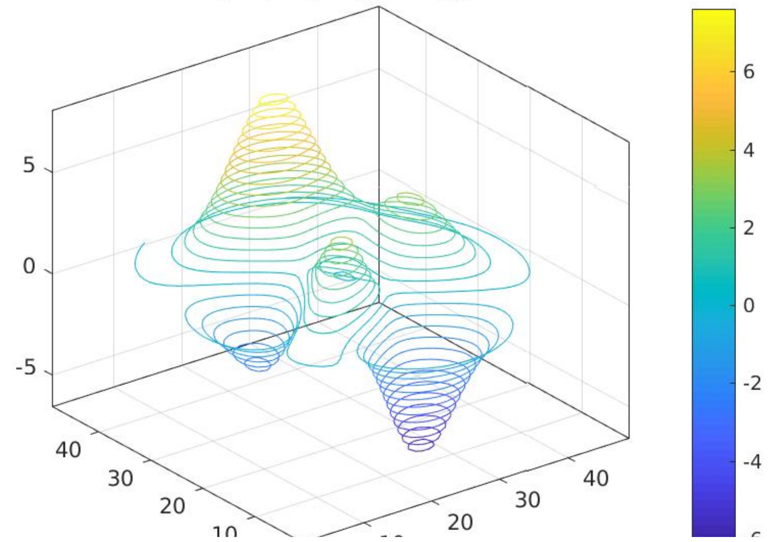


# Example plots 5, 6, 7, 8

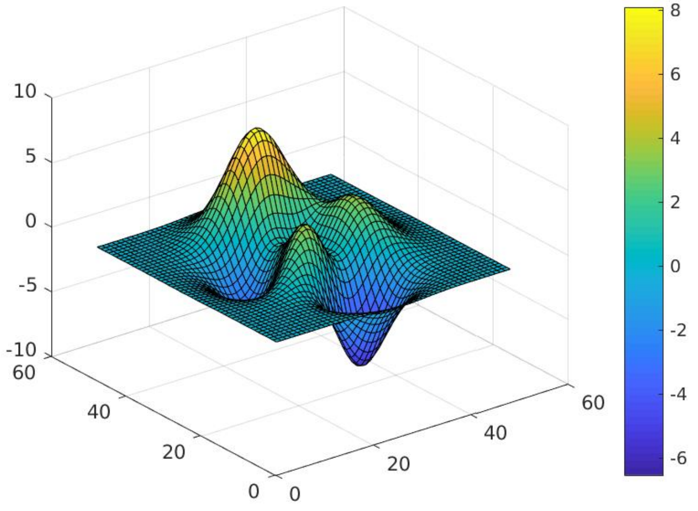
```
contourf(X,Y,Z,15); colorbar
```



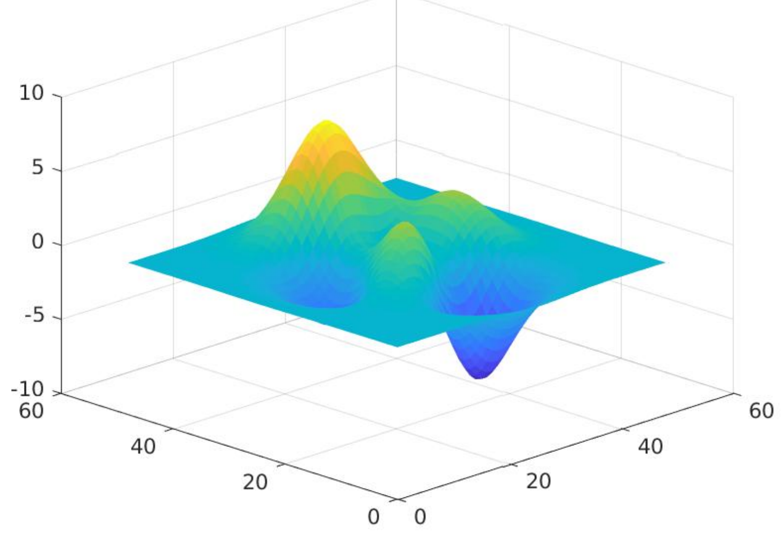
```
contour3(X,Y,Z,30); colorbar
```



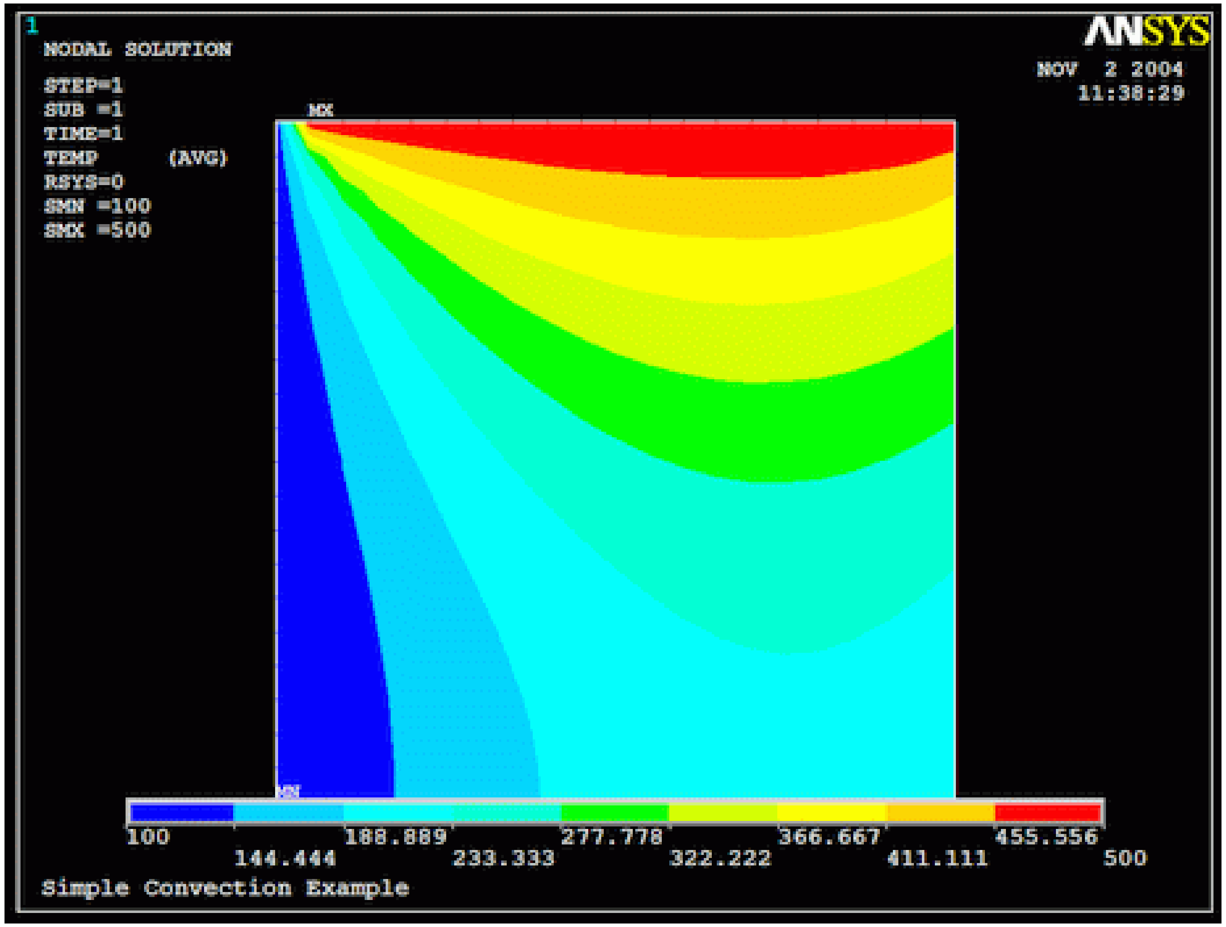
```
surf(X,Y,Z); colorbar
```



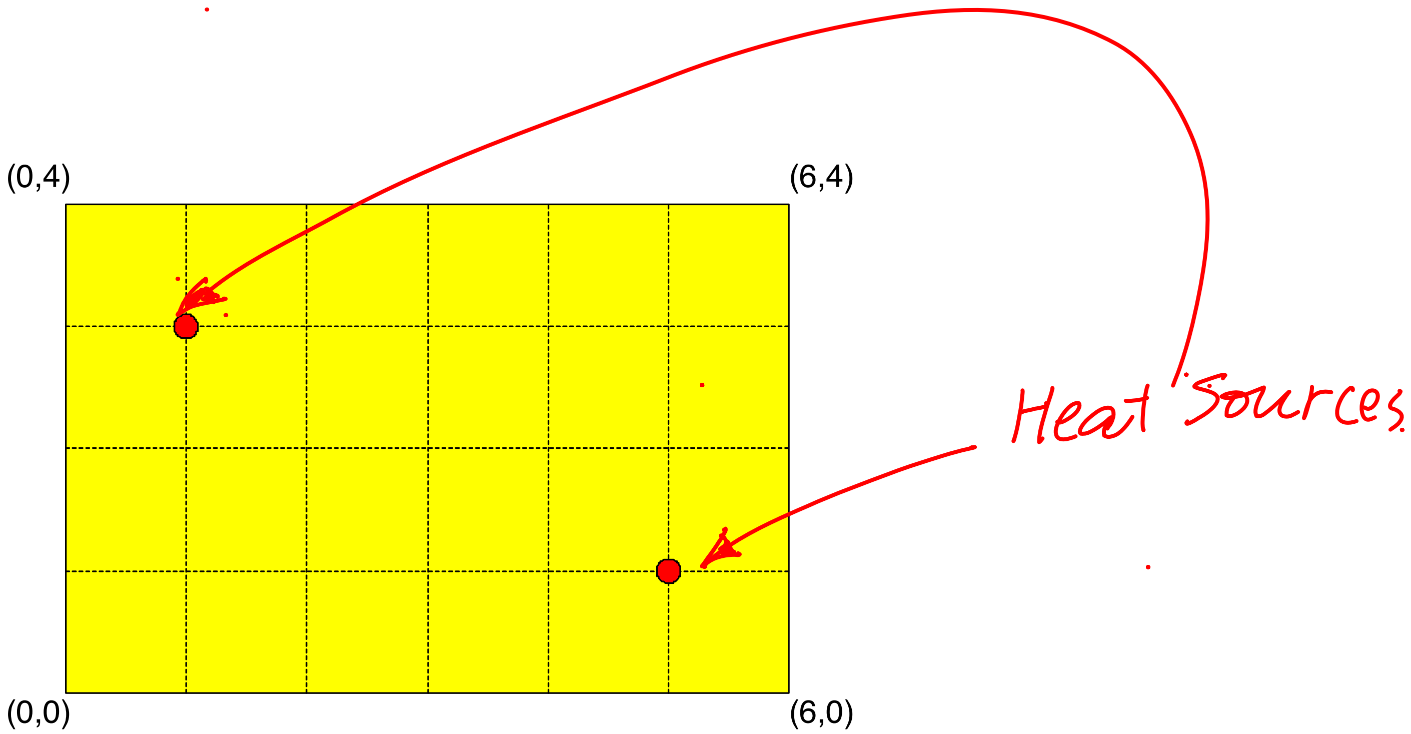
```
surf(X,Y,Z,'EdgeColor','None')
```



# A Temperature Field Contour



# Display a field of temperatures T(X,Y)



## A Temperature Field

Suppose that in the rectangle [0,6] x [0,4], we placed two heat sources at (1,3) and (5,1), and let the heat spread across the region over time.

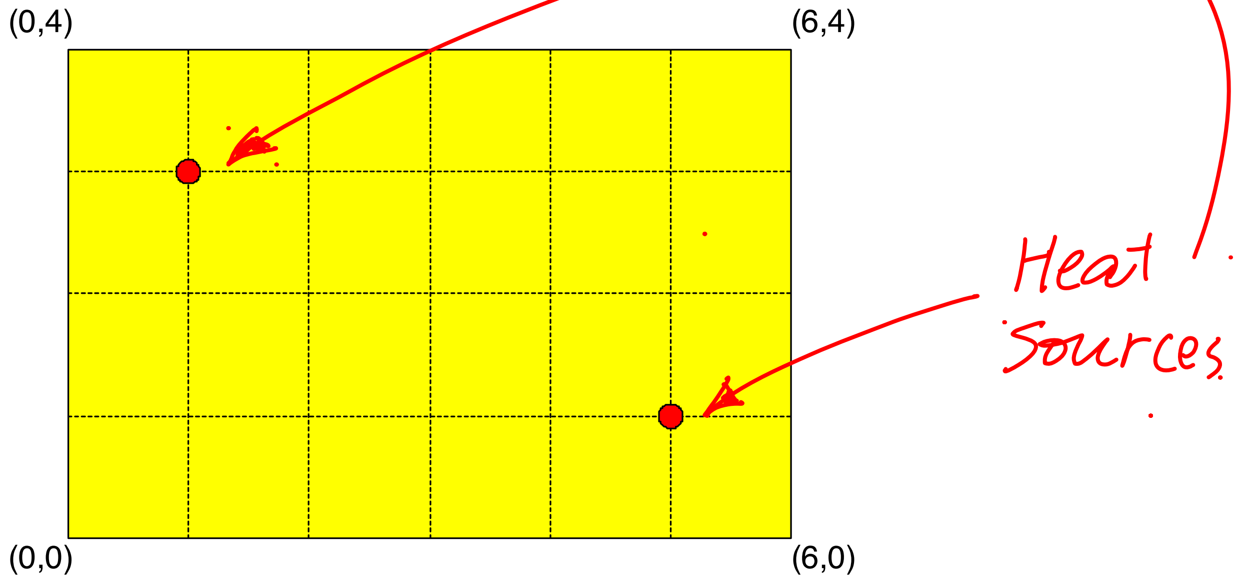
Suppose that the resulting temperature distribution has the form

T(x,y) = 100 e^(-0.4*(     (x-1)^2+0.7*(y-3)^2 ) )

+    80 e^(-0.2*( 2*(x-5)^2+1.5*(y-1)^2) )

Can we display this temperature field?

## Visualize temperature distribution—contour plot



$$T(x,y) = 100e^{-.4((x-1)^2+.7(y-3)^2)} + 80e^{-.2(2(x-5)^2+1.5(y-1)^2)}$$

## Contour Procedure

Choose n values for vector of values x over [0,6];

Choose m values for vector of values y over [0,4];

Create arrays X and Y over [0,6] x [0,4].

Evaluate function T(X,Y) to get "Z" array.

Call contour() or contour3() or contourf() or surf().

Optional calls to colorbar() or clabel().

# Choosing N and M

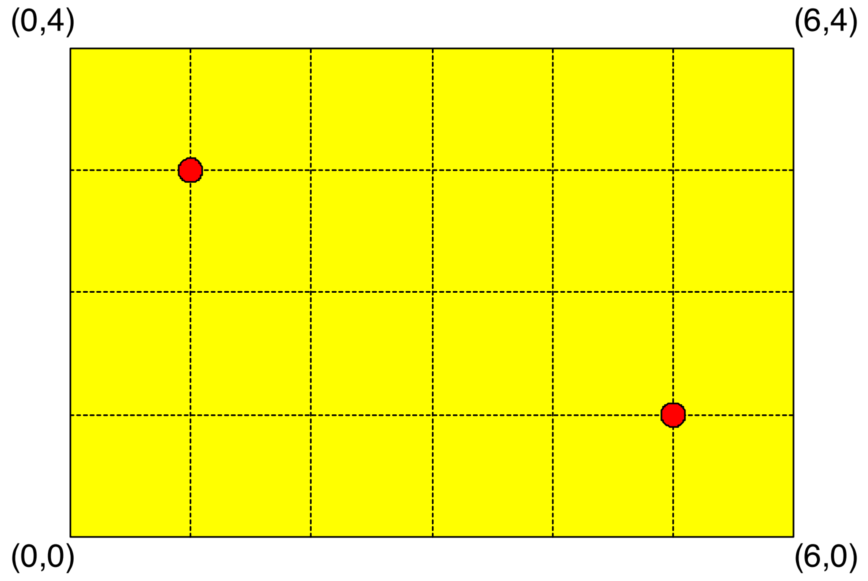
The region is 6 wide by 4 tall.

To define a grid of squares, we would use

n = multiple of 6 + 1; (width)

m = multiple of 4 + 1; (height)

Remember why we have to add 1!



## Set grid: X and Y

```
% x and y are vectors, simple lists.
%
xvec = linspace ( 0.0, 6.0, n );
yvec = linspace ( 0.0, 4.0, m );
%
%  X and Y are tables or matrices!
%
[ X, Y ] = meshgrid ( xvec, yvec );
```

# temperature_grid.m

```
function [ X, Y ] = temperature_grid ( k )

%% temperature_grid defines arrays X and Y for the temperature problem.
%
%  K controls the fineness of the grid.
%
  xvec = linspace ( 0.0, 6.0, k*6+1 );
  yvec = linspace ( 0.0, 4.0, k*4+1 );

  [ X, Y ] = meshgrid ( xvec, yvec );

  return
end
```

# [X,Y] = MESHGRID(x,y)?

```
y=4 | 0,4  1,4  2,4  3,4  4,4  5,4  6,4
y=3 | 0,3  1,3  2,3  3,3  4,3  5,3  6,3
y=2 | 0,2  1,2  2,2  3,2  4,2  5,2  6,2
y=1 | 0,1  1,1  2,1  3,1  4,1  5,1  6,1
y=0 | 0,0  1,0  2,0  3,0  4,0  5,0  6,0
    +--^----^----^----^----^----^----^--
      x=0  x=1  x=2  x=3  x=4  x=5  x=6
```

# [X,Y] = MESHGRID(x,y)?

```
          0    1    2    3    4    5    6

          0    1    2    3    4    5    6

X =       0    1    2    3    4    5    6

          0    1    2    3    4    5    6

          0    1    2    3    4    5    6


          4    4    4    4    4    4    4

          3    3    3    3    3    3    3

Y =       2    2    2    2    2    2    2

          1    1    1    1    1    1    1

          0    0    0    0    0    0    0
```

## temperature.m

Formula:

```
T(x,y) = 100 e^(-0.4*(     (x-1)^2+0.7*(y-3)^2 ) )
          +    80 e^(-0.2*( 2*(x-5)^2+1.5*(y-1)^2) )
```

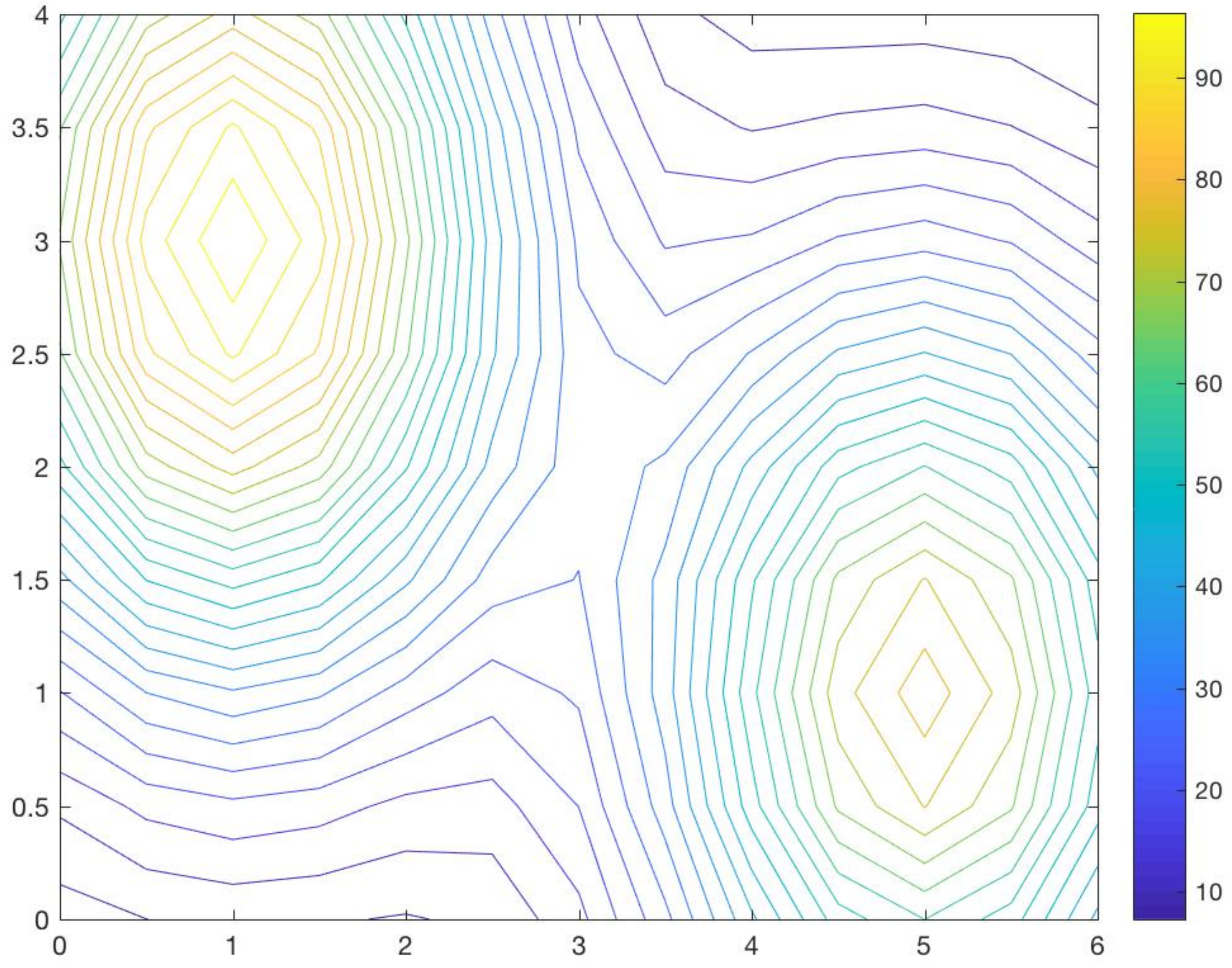
Function:

```
function T = temperature ( X, Y )
% X, Y, and T can be arrays, so use dot operators!
  arg1 = -0.4 * (         ( X-1.0) .^ 2 + 0.7 * (Y-3.0) .^ 2 );
  arg2 = -0.2 * ( 2.0 * (X-5.0) .^ 2 + 1.5 * (Y-1.0) .^ 2 );
  T = 100.0 * exp ( arg1 ) + 80.0 * exp ( arg2 );
  return
end
```

## Simple Contour Plot

```
k = 2;
[X,Y] = temperature_grid ( k );
T = temperature ( X, Y );
contour ( X, Y, T, 25 );   % <- 25 contour lines
colorbar ( );
```

# temperature contour plot



# Specifying the Contour Levels

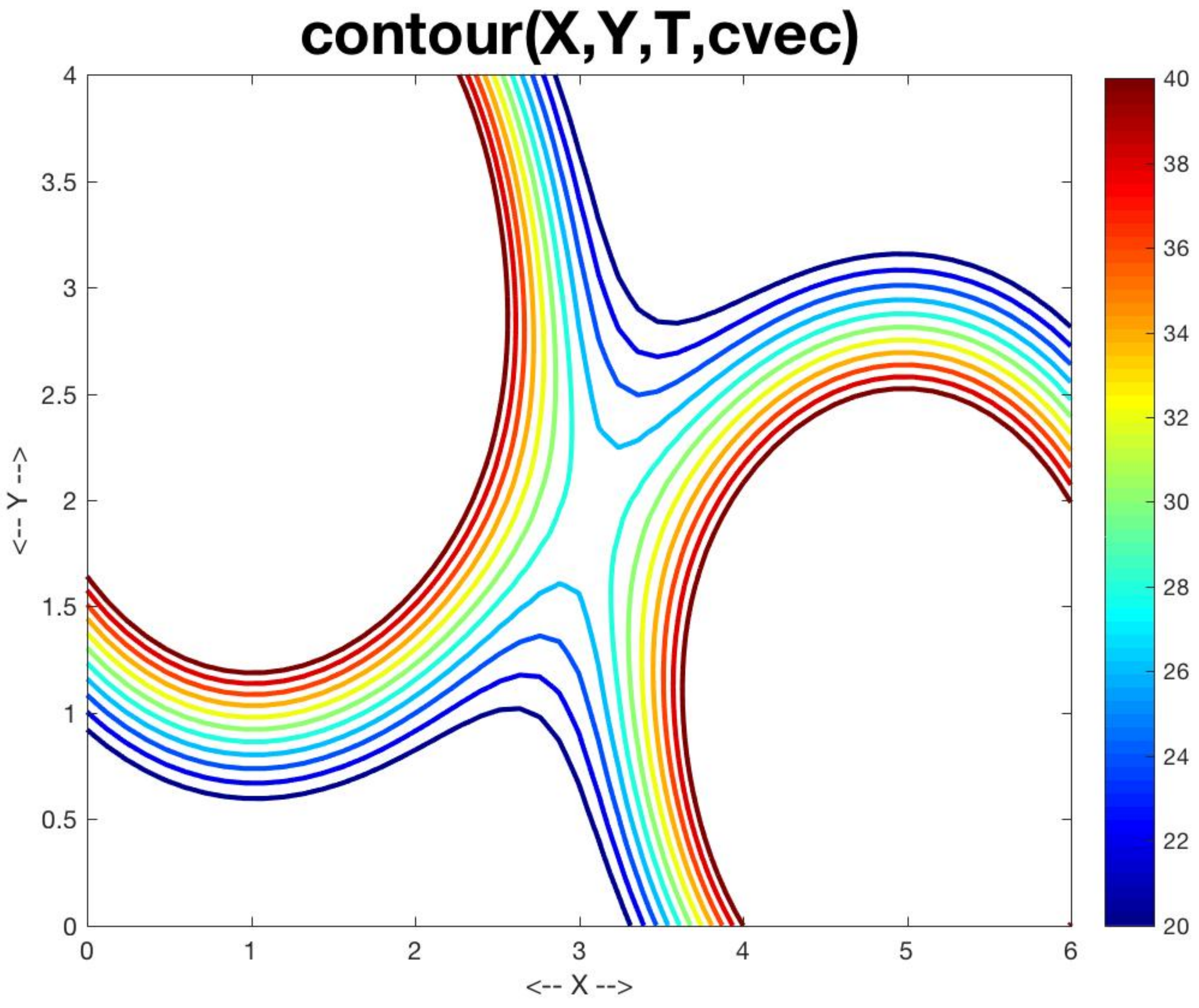
contour(X,Y,Z) will use 10 contour levels, equally spaced between Zmin and Zmax.

contour(X,Y,Z,25) will use 25 such levels.

You can pick your own vector of contour levels:

```
cvec = linspace ( 20.0, 40.0, 11 );
contour(X,Y,Z,cvec);
```

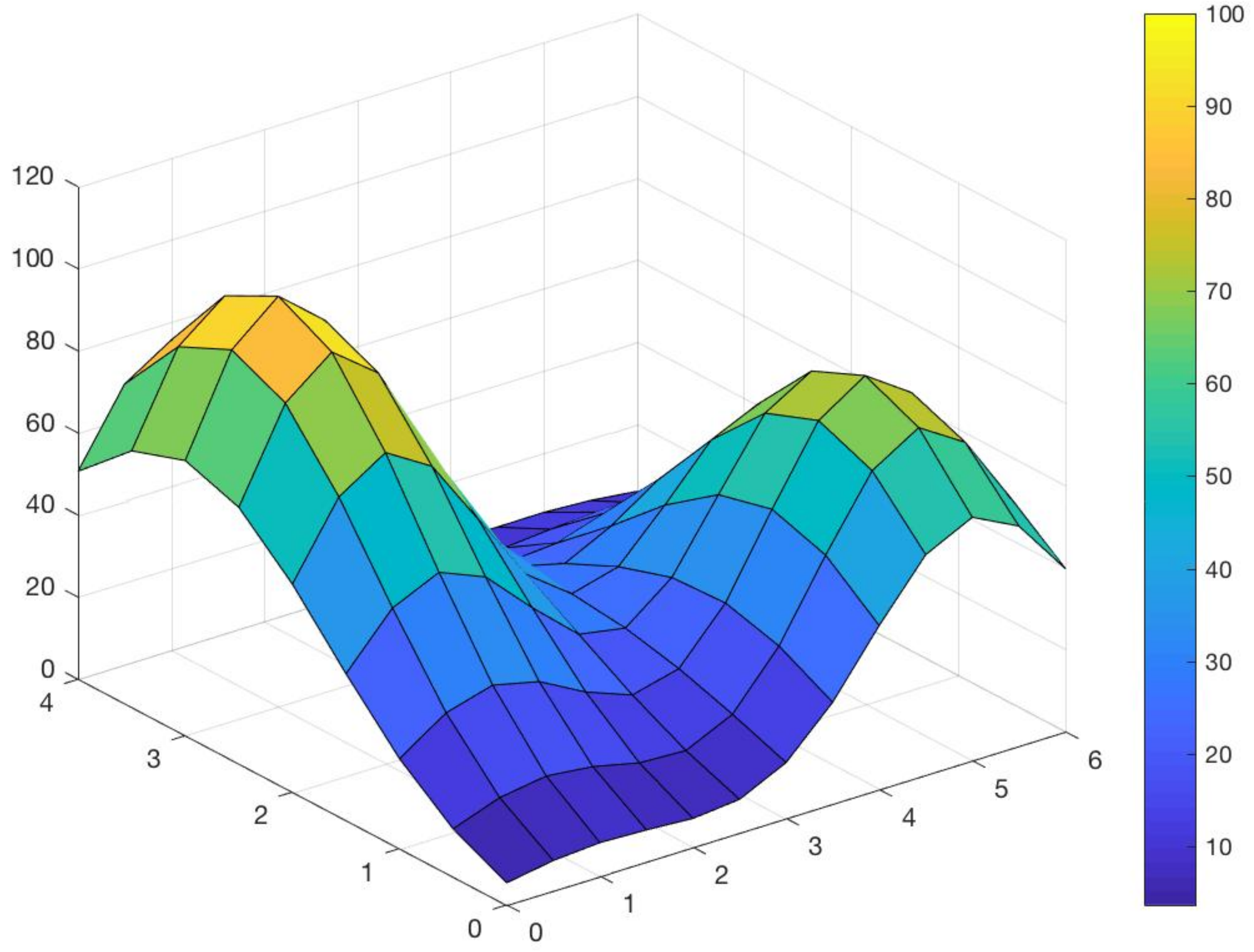
# 20 <= Contours <= 40



## Simple Surface Plot

```
k = 2;
[X,Y] = temperature_grid ( k );
T = temperature ( X, Y );
surf ( X, Y, T );
colorbar ( );
```

# temperature surface plot



## Slicing the Data

# Slicing the Data

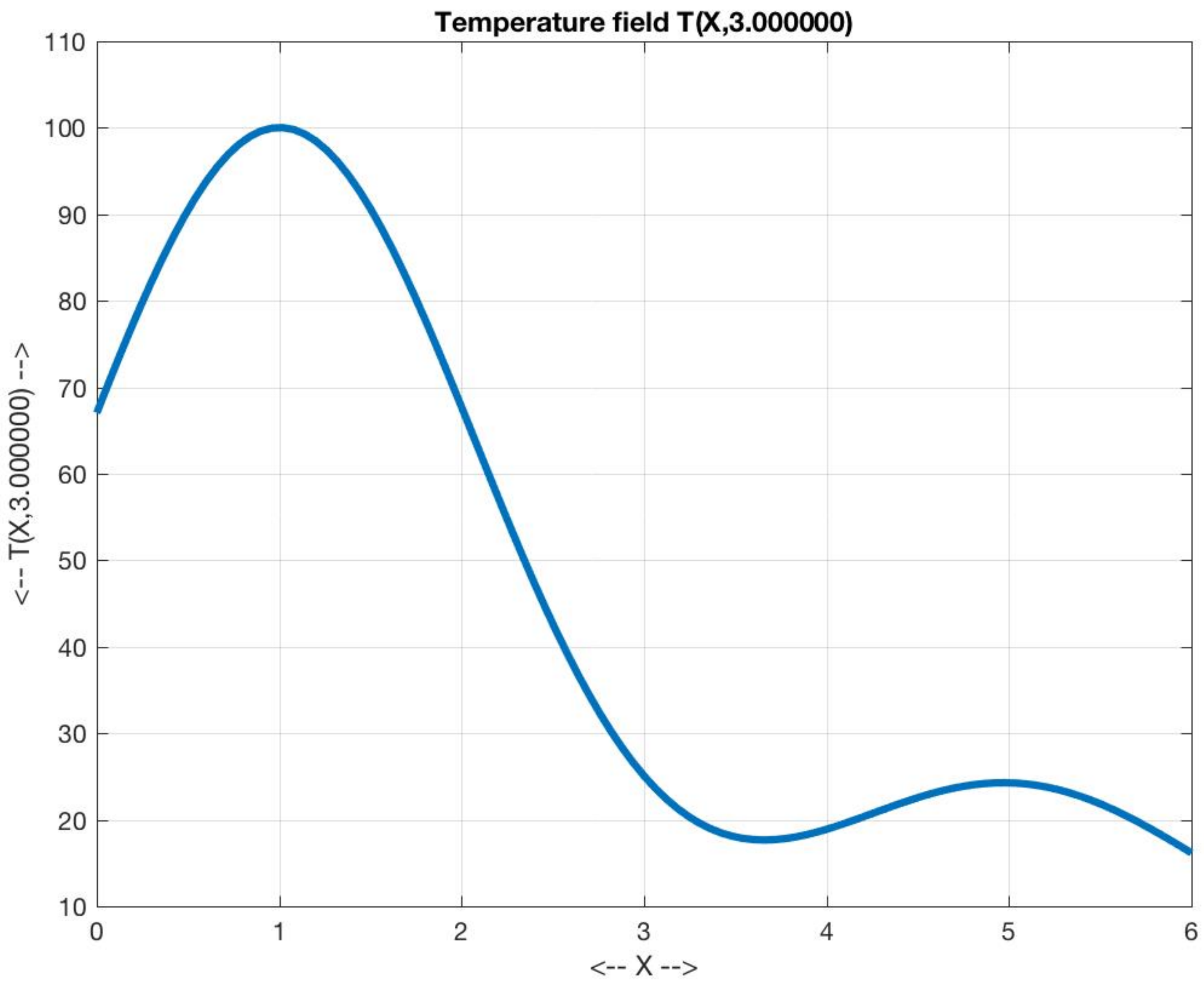
The contour plot suggests that the temperature field has peak values at (1,3) and (5,1).

We can make "slices" of the data to see this more closely:

Let x go from 0 to 6, while y = 3, and plot.

```
xvec = linspace ( 0.0, 6.0, 101 );   <- xvec is a row vector
yvec = 3.0 * ones ( 1, 101 );        <- make yvec a row vector = 3
zvec = temperature ( xvec, yvec );
plot ( xvec, zvec );
```

# Plotting T(X,3)

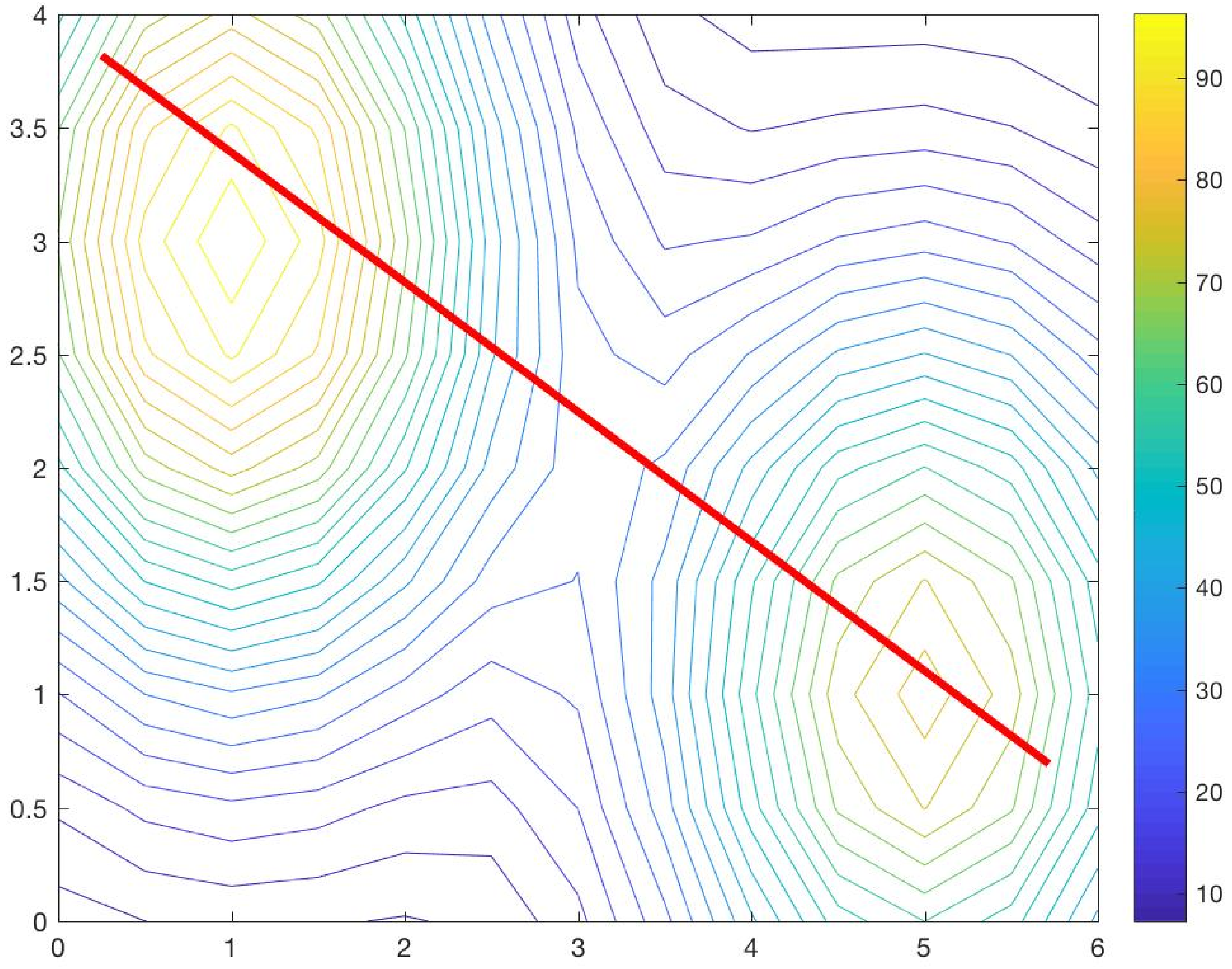


# Sample T(X,Y) along a Line

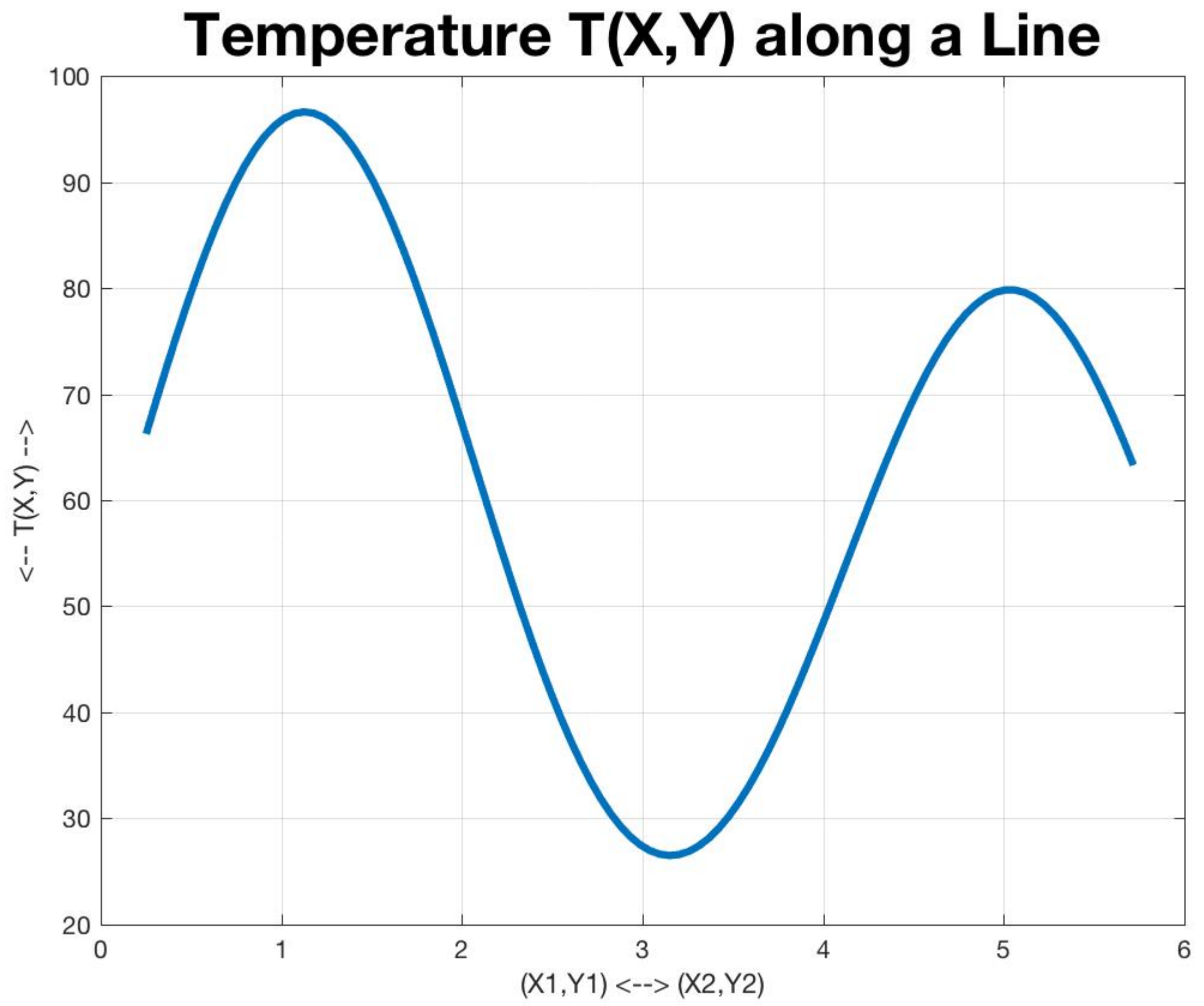
We can interactively select a slicing line  temperature_contour ( );

```
figure ( 1 )
k = 2;
[X,Y] = temperature_grid ( k );
T = temperature ( X, Y );
temperature_contour ( X, Y, T )

[ x1x2, y1y2 ] = ginput ( 2 );

figure ( 2 )
xvec = linspace ( x1x2(1), x1x2(2), 101 );
yvec = linspace ( y1y2(1), y1y2(2), 101 );
zvec = temperature ( xvec, yvec );
plot ( xvec, zvec, 'Linewidth', 3 );:
```

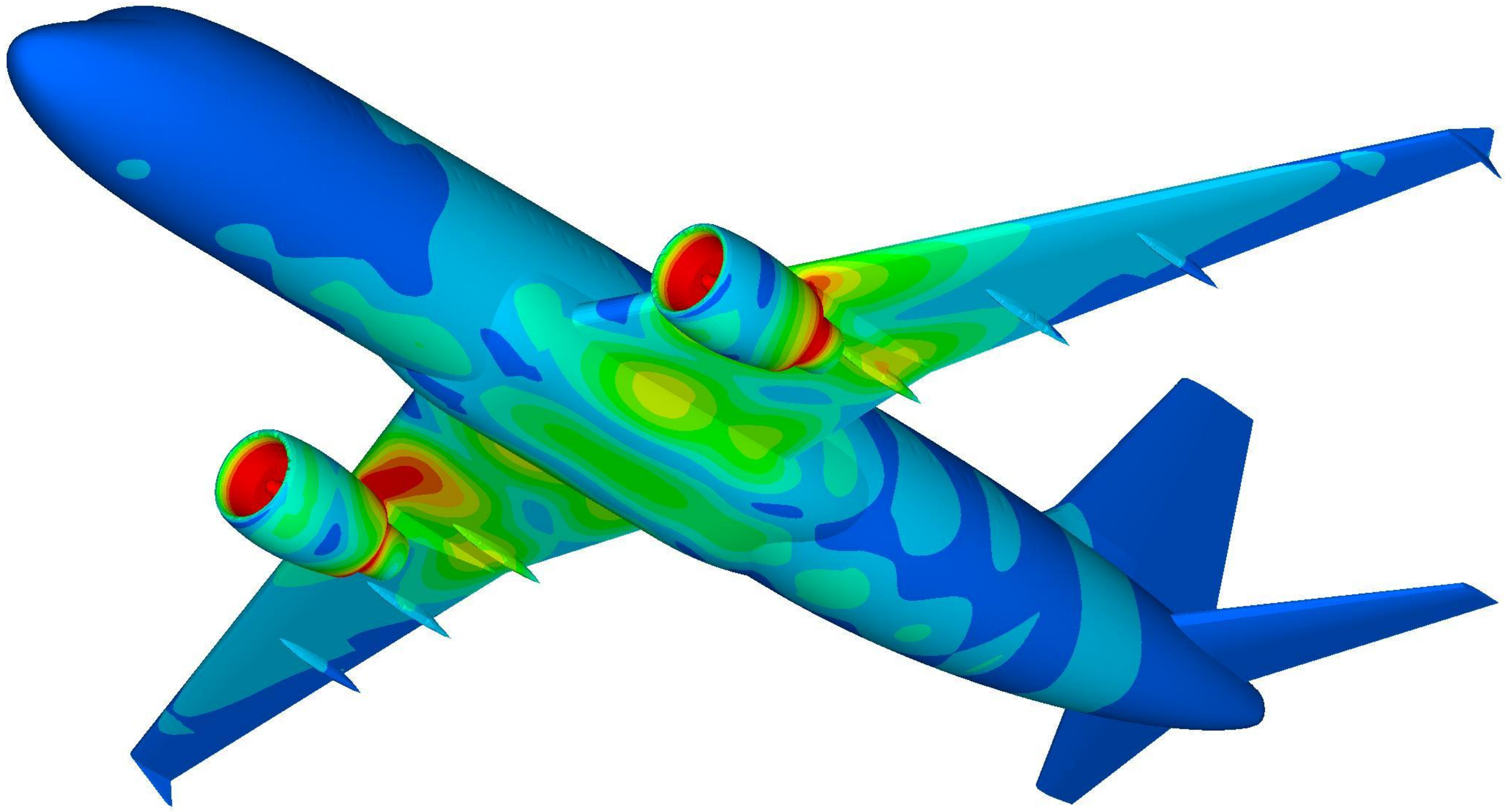
# Our Slicing Line



# Temperature along the Line



## A Model of Cooling



## How Temperature Settles Down

Our 6x4 rectangle had a specified temperature distribution given by the function T(X,Y). This distribution included some two "hot spots".

In our experience, hot spots tend to spread and settle down.

Can we come up with a MATLAB model of temperature that starts with our given distribution T(X,Y), and then behaves like real physical objects for which hot spots cool down?

## The Boundary Problem

Physical objects are in contact with the outside world. The boundary of our rectangle must be touching other objects with a given temperature, and this contact will affect the results.

To simplify the problem, we will assume that the boundary of the rectangle is bathed in ice water, and so has a temperature of 0 degrees Celsius.

Our grid will not involve boundary nodes, where the temperature will always be 0, and interior nodes where we expect cooling to occur.

Let's define this new temperature field.

# temperature2.m

```
function T = temperature2 ( X, Y )

  arg1 = -0.4  * (       (X-1.0) .^ 2 + 0.7 * (Y-3.0) .^ 2 );
  arg2 = -0.2 * ( 2.0 * (X-5.0) .^ 2 + 1.5 * (Y-1.0) .^ 2 );
  T = 100.0 * exp ( arg1 ) + 80.0 * exp ( arg2 );

  T(1,:) = 0.0;          % <- T(1,1:n) = 0;
  T(end,:) = 0.0;        % <- T(m,1:n) = 0;
  T(:,1) = 0.0;          % <- T(1:m,1) = 0;
  T(:,end) = 0.0;        % <- T(1:m,n) = 0

  return
end
```

## T(:,end) = 0?

We can set an entire MxN matrix T to 0 by writing

```
T = 0;        or       T = zeros(m,n);
```

or even using a FOR loop.

But to set row i to zero, we can write

```
T(i,1:n) = 0;       or       T(i,:) = 0
```

To set column j to zero, we can write

```
T(1:m,j) = 0;       or      T(:,j) = 0
```

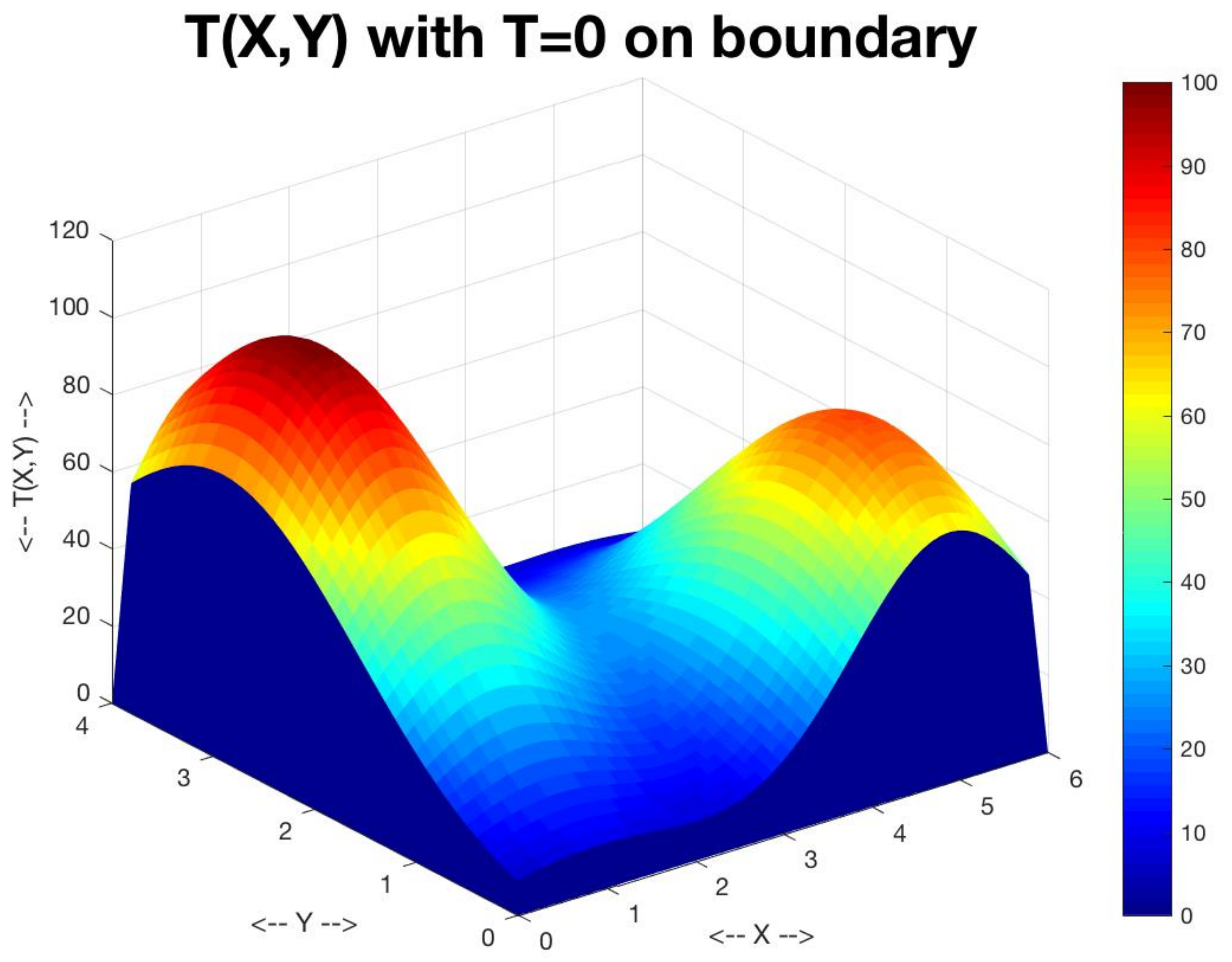
To set the last row or last column to zero:

```
T(end,:) = 0;      T(:,end) = 0;
```

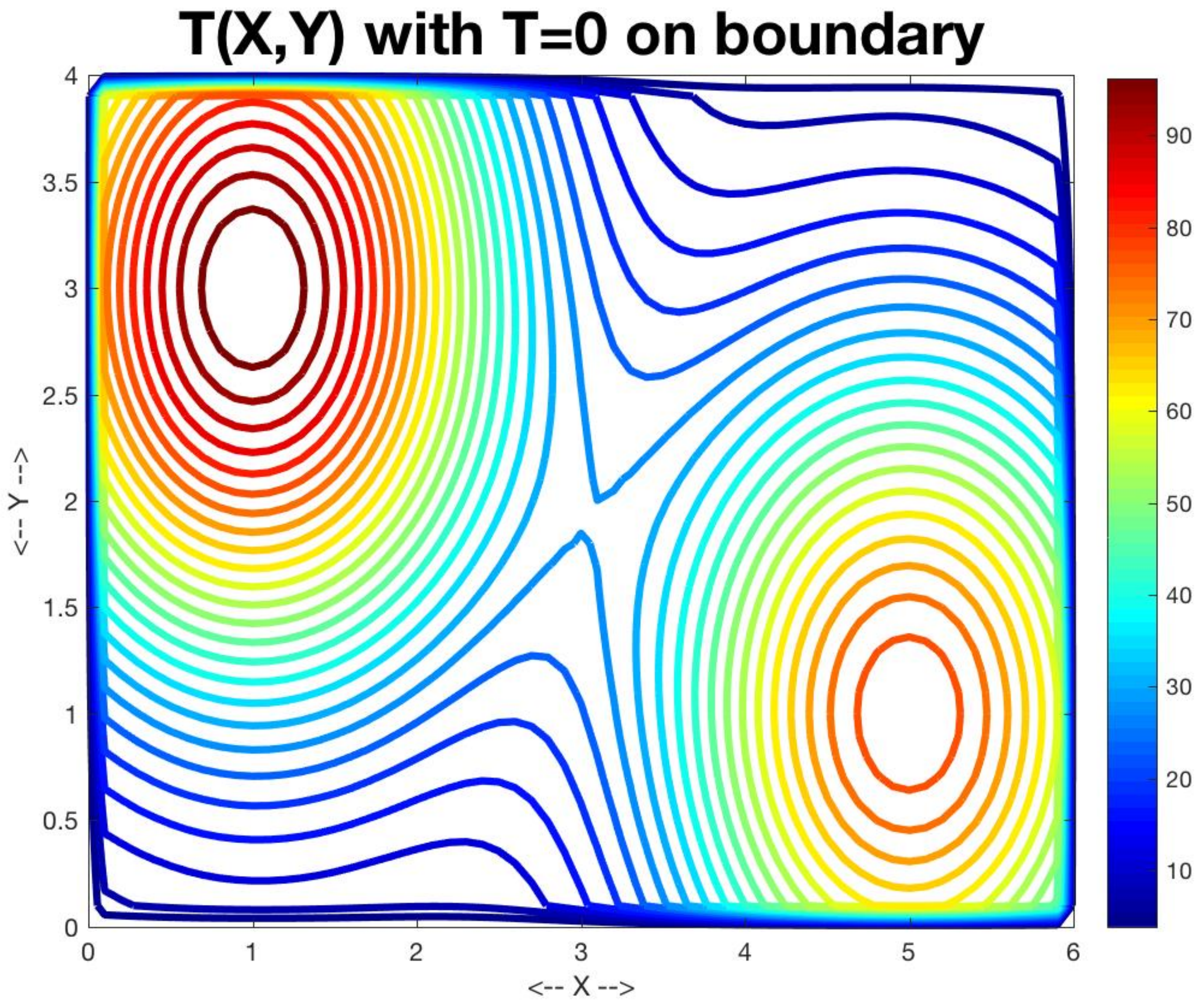
":" means "*everything in this dimension*";

"end" means "*last row" or "last column*";

# Surface Plot with Zero Boundary



# Contour Plot with Zero Boundary



## A Cooling Model

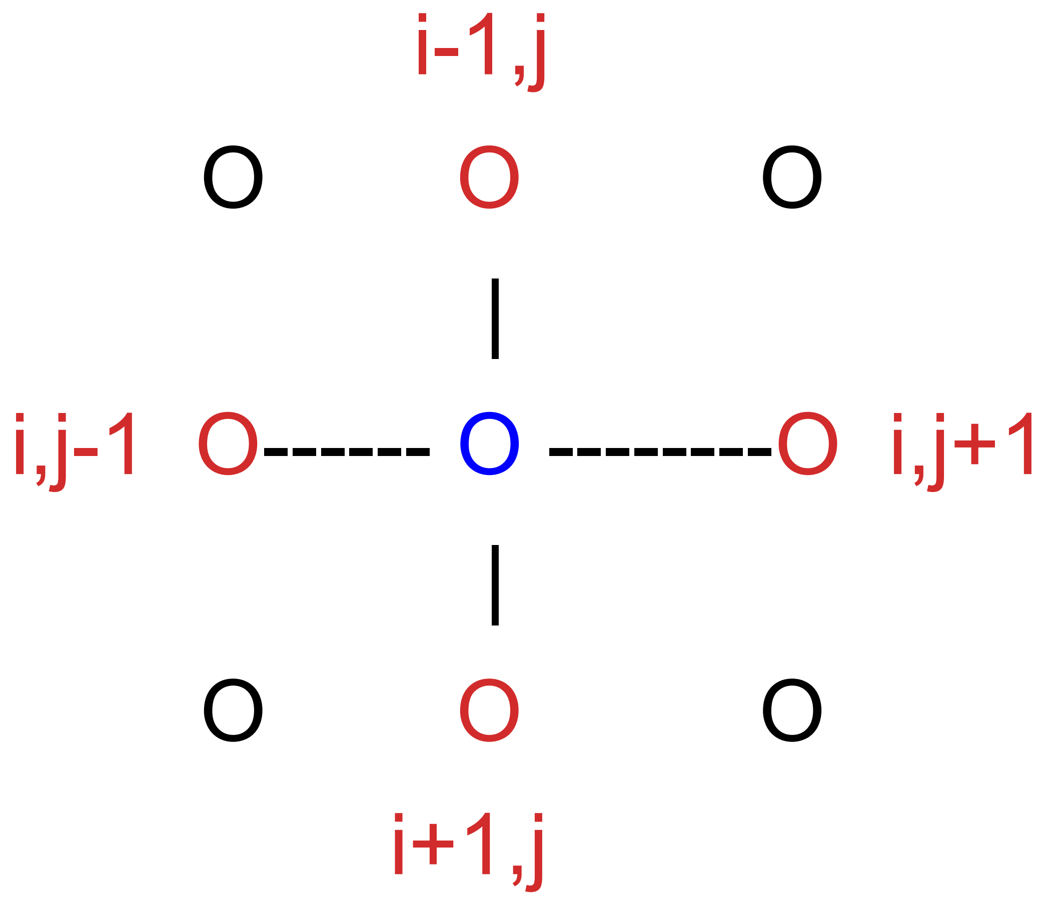
We simulate the cooling process by setting the temperature to the initial field, and calling that time step 0.

Then we model the temperature at time steps = 1, 2, 3, ... (without specifying the units of time.)

Our model of cooling uses the following rule:

*To find the temperature at any interior node, average the temperatures of the four neighbors at the previous time.*

## New Blue = Average Old Reds



## Cooling with a FOR Loop

```
Told = T;
for i = 2 : m – 1
  for j = 2 : n – 1
    T(i,j) = 0.25 * (  Told(i-1,j) ...  (west)
                     + Told(i+1,j) ... (east)
                     + Told(i,j-1) ...  (south)
                     + Told(i,j+1) );  (north)
  end
end
```

## Cooling with Colons

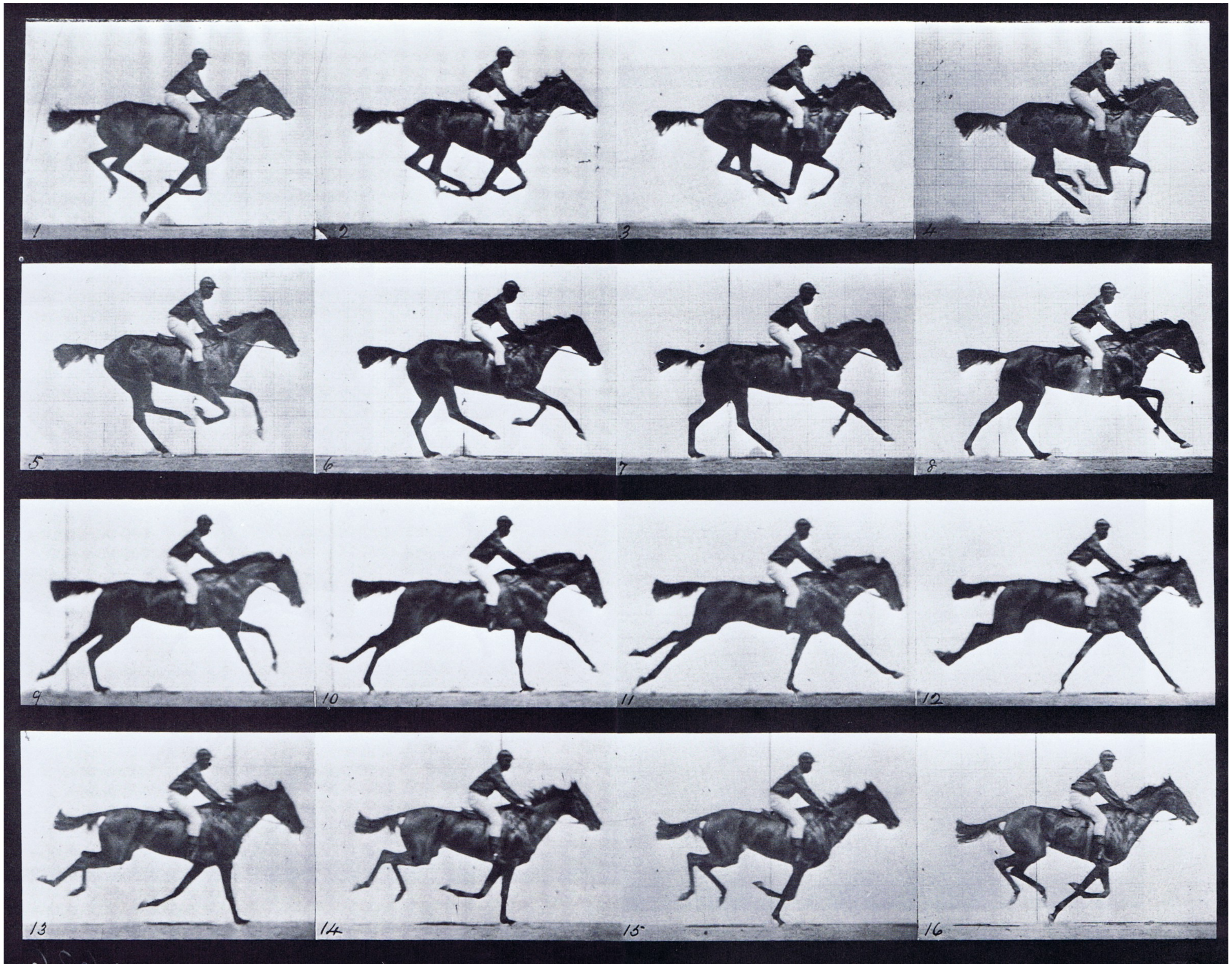
```
function T = cooling ( T )

  Told = T;
  [ m, n ] = size ( T );

  T(2:m-1,2:n-1) = 0.25 * (  Told(1:m-2,2:n-1) ..  (west).
                           + Told(3:m,2:n-1) ...    (east)
                           + Told(2:m-1,1:n-2) ...(north)
                           + Told(2:m-1,3:n) );    (south)
  return
end
```

# What Do We Do With Our Data?

If we run our program, we can see the maximum temperature decreases, but that doesn’t tell us very much!

```
Step  0, 0 <= T <= 100.04
Step  1, 0 <= T <= 97.9448
Step  2, 0 <= T <= 95.9389
Step  3, 0 <= T <= 94.0168
Step  4, 0 <= T <= 91.6163
Step  5, 0 <= T <= 88.7641
Step  6, 0 <= T <= 85.6495
Step  7, 0 <= T <= 83.0865
Step  8, 0 <= T <= 80.9457
Step  9, 0 <= T <= 78.7251
Step 10, 0 <= T <= 76.4688
```

# Animating Time Data



## Animate a Plot Sequence

A table of data overloads our brain, but our eye can spot patterns if we convert the data to a plot.

In the same way, having a sequence of plots can overwhelm our brain, but if we can add animation, then the skills we use to live in the world can help us see patterns in the moving data.

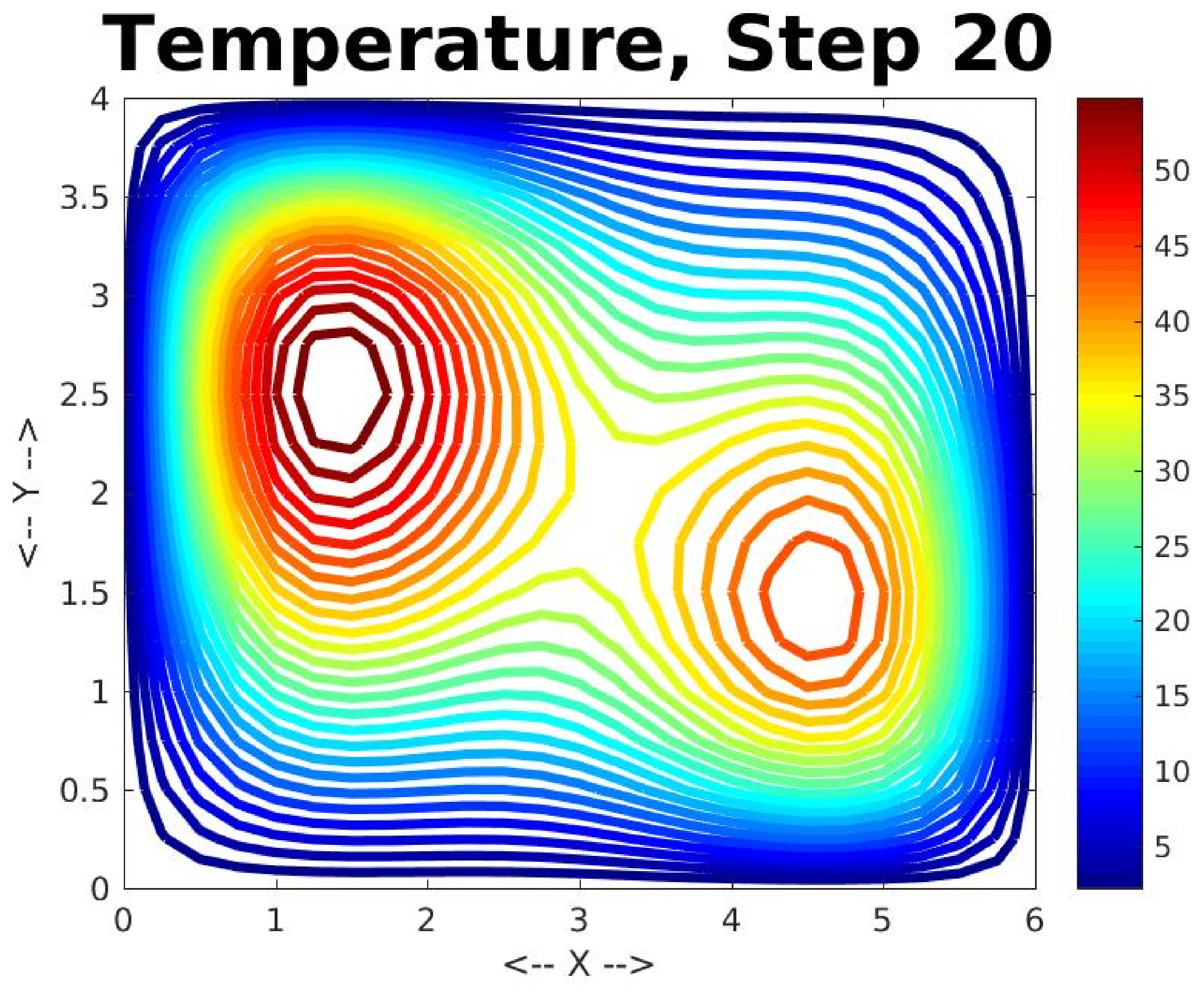
We will look at a simple technique for animating a sequence of contour or surface plots associated with our cooling problem.

Our basis tools are a for loop and the pause command.

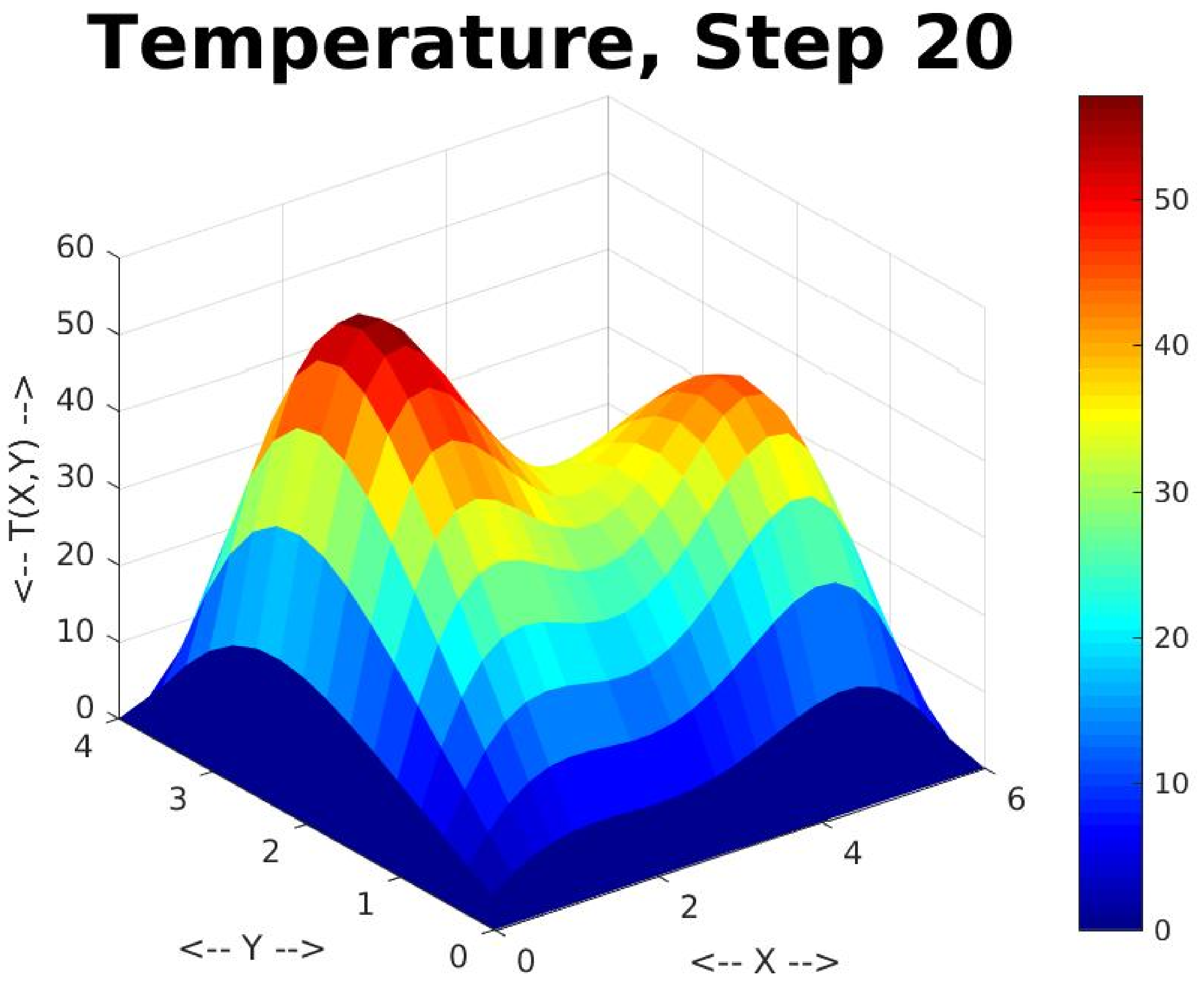
## Display a Contour Sequence

```
for step = 0 : 10
  if ( step == 0 )
    [ X, Y ] = temperature_grid ( 4 );
   T = temperature2 ( X, Y );
  else
    T = cooling ( T );
  end
  temperature_contour ( X, Y, T );
  pause ( 1 );          <- Wait 1 second, before moving on.
end
```

# Step 20 in Contour Sequence



# Step 20 in Surf Sequence



## Scaling Problem

As the contour and surface animations proceed, we notice that the scale on the colorbar keeps changing. Although the temperature is decreasing, the scale is updated so that the current highest temperature shows as red.

In the surface plot, the vertical scale keeps adjusting so that the “peaks” are kept high.

Since the temperature is cooling, we may instead want to use a fixed scale for all the images.

Then we will see things turn “blue” or go flat.

# Define and Use a Fixed Scale

To use a fixed scale, we compute the initial temperature field T, and record the minimum and maximum values at this starting time:

```
tmin = min ( min ( T ) );
tmax = max ( max ( T ) );
```

Then we call caxis ( [tmin,tmax] ), which tells the colorbar to reference these original values, rather than using the minimum and maximum observed at the current step.

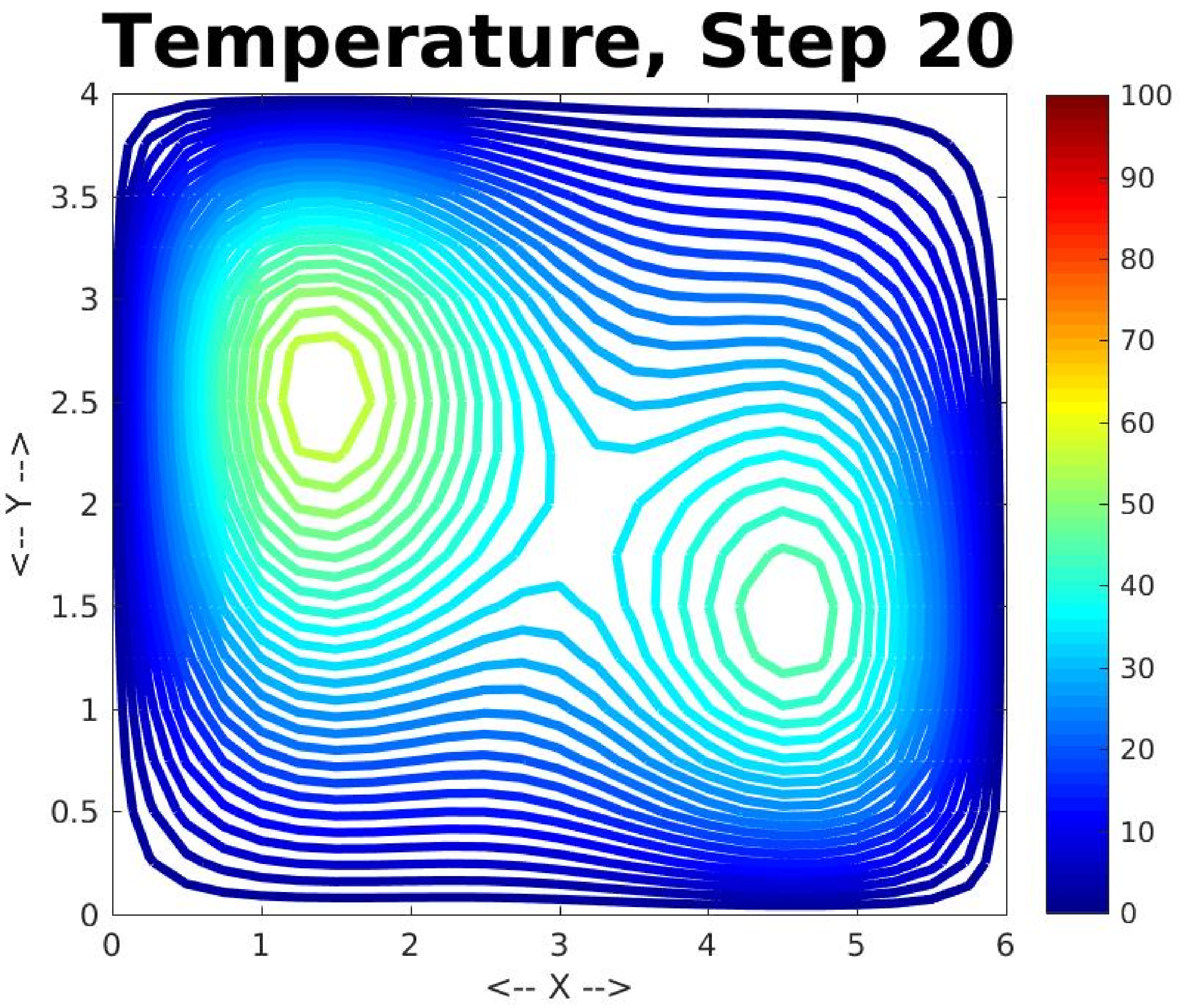
# Displaying Scaled Contours

```
for step = 0 : stepmax

  if ( step == 0 )
    [ X, Y ] = temperature_grid ( 4 );
    T = temperature2 ( X, Y );
    tmin = min ( min ( T ) );
    tmax = max ( max ( T ) );
  else
    T = cooling ( T );
  end

  title_string = sprintf ( 'Temperature, Step %d', step );
  temperature_contour_scale ( X, Y, T, tmin, tmax, title_string );
  pause ( 1 );

end
```

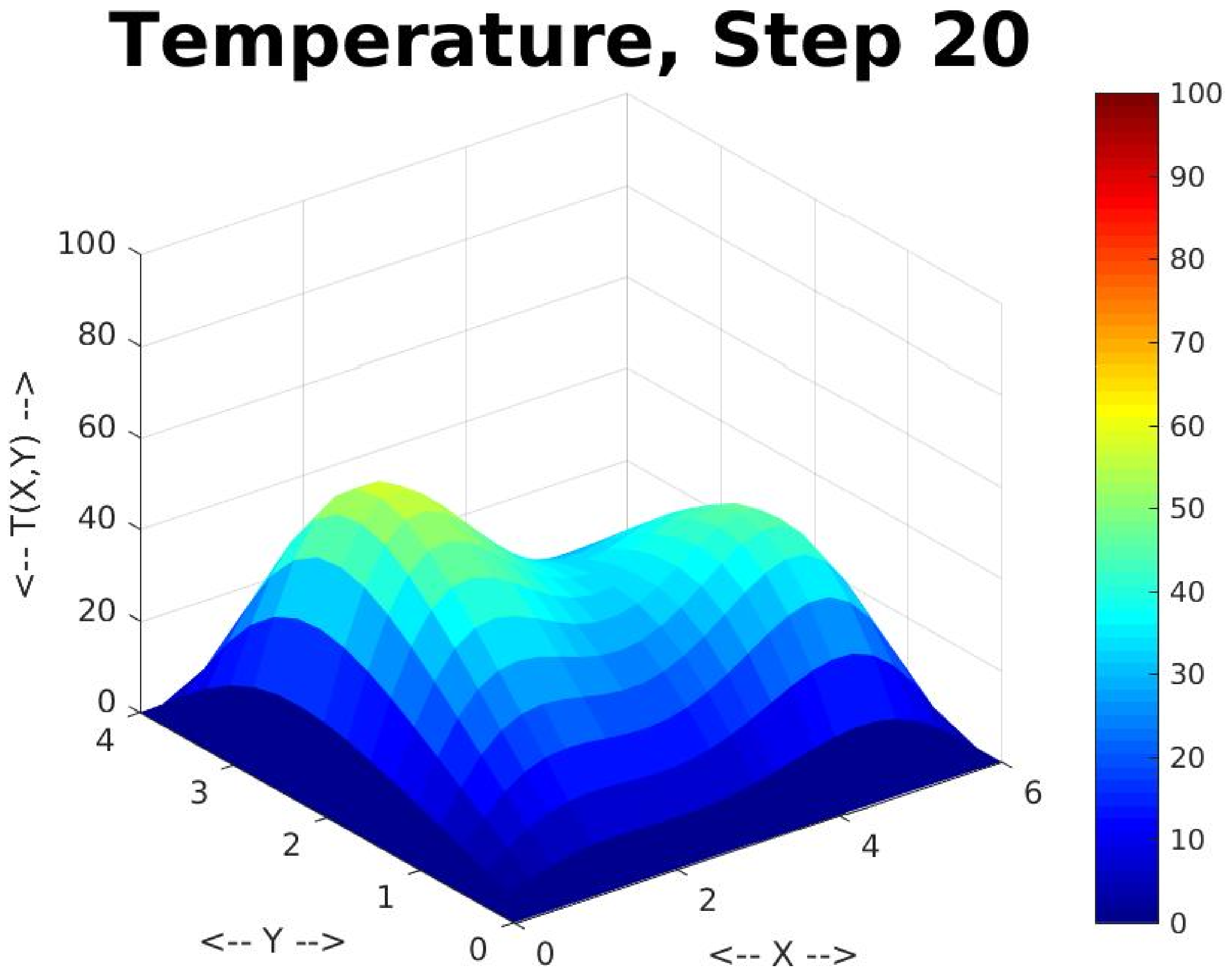
# temperature_contour_scale.m

```
function temperature_contour_scale ( X, Y, T, tmin, tmax, title_string )

  contour ( X, Y, T, 25, 'LineWidth', 3 );

  colorbar ( );
  colormap ( jet );
  caxis ( [ tmin, tmax ] );       % <- Color bar scale based on TMIN, TMAX

  title ( title_string, 'Fontsize', 24 );
  return
end
```

# Step 20, Scaled Contours



# Step 20, Scaled Surface


Temperature, Step 20
<-- T(X,Y) -->
<-- Y -->
<-- X -->