Solutions For Homework #1

Problem 1: [20 pts]

(a) The file *image1* consists of 96320 1-byte elements. In this part of the problem, we need to find two integers x and y whose product $xy = 96320$. The numbers x and y represent the number of rows and columns in the image respectively. The linelength, then, would just be the number of columns y.

```
% the following is code to find the factors
% of datalen

denominator = 1; factors = [];
while(1)
   numerator = datalen/denominator;

   % store results of division only if it is
   % a whole number
   if( round(numerator)-numerator == 0)
      factors = [factors; numerator denominator];
   end

   denominator = denominator + 1;
   if(denominator > datalen); break; end;
 end
```

The code above produces about 27 distinct pairs of integers, each of which a possible candidate pair for (x, y). We find, after some trial and error, that the correct pair is $x = 172, y = 560$. Hence, the linelength is 560.

Alternatively, we can get an idea of the linelength by inspecting a plot of the first 1000 or so elements in the data array. This is shown in Figure 1. Note the periodic nature of the data. This is due to the presence of several bytes of header information at the start of each line. By noting the locations of the header in this 1D sequence, we can get an idea of the linelength.

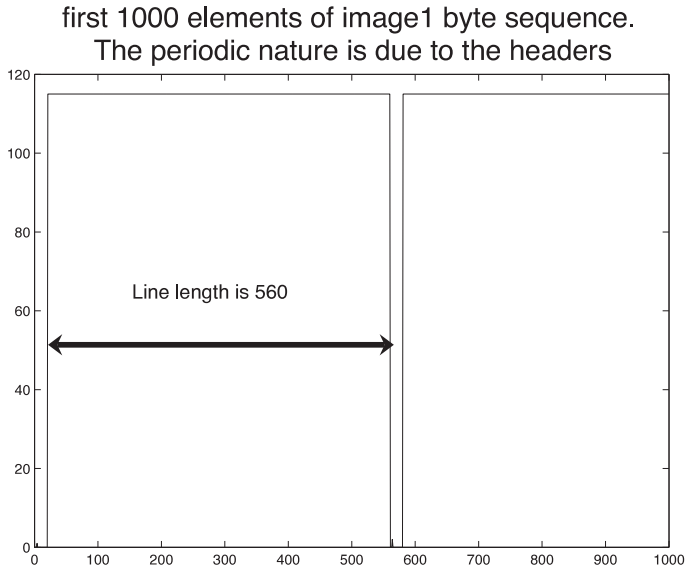

Figure 1: Plot of first 1000 elements of *image1*. Note the periodic nature of the data. The inferred linelength is 560

(b) The number of header bytes is 20

(c) The header section of the image contains a counter for each image line. This counter is located at element number 4 of each line. By extracting the counter for each line, and checking the progression of counter values, we can find the line that is missing. It is line number 114. The image is shown below: The display of the image was set such that black was assigned the value 0 and white 255.

Problem 2: [20 pts]

(a) The histogram of the image given in Problem 1c is shown in Figure 3

The mean μ and standard deviation of *image1* is $\mu = 69.05, \sigma = 26.61$.

(b) The desired mean is $\mu_1 = 128$ while the desired standard deviation is $\sigma_1 = 80$. The linear transformation that outputs an image with the desired mean and standard deviation is

$$\text{transformed image} = \sigma_1 \frac{(\text{image} - \mu)}{\sigma} + \mu_1 \tag{1}$$

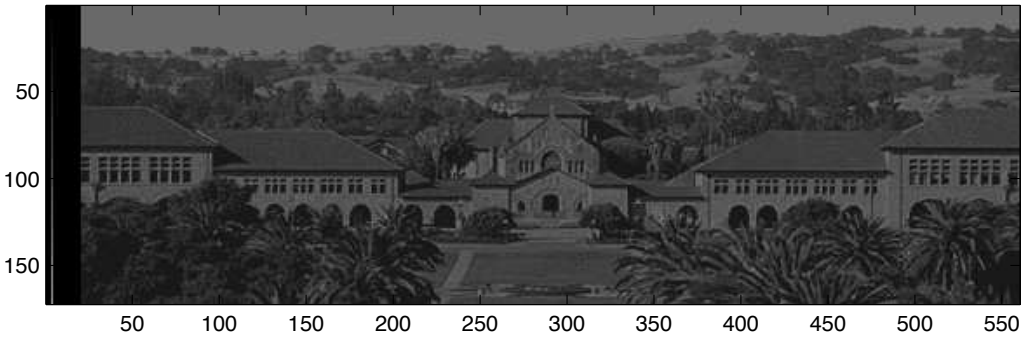

Figure 2:

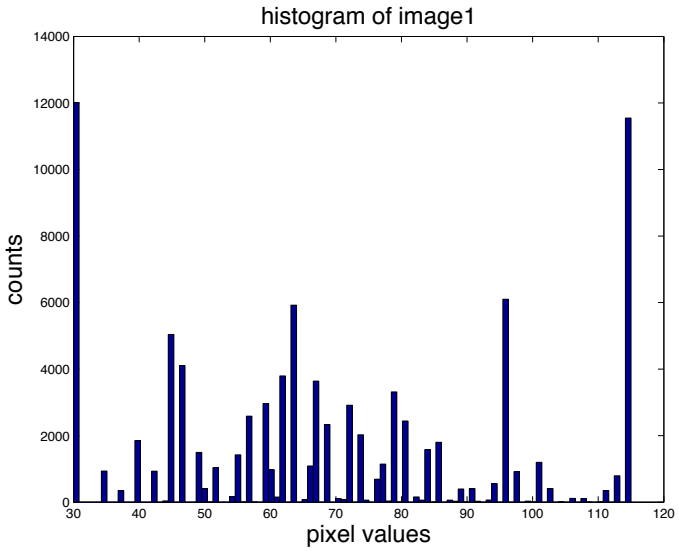

Figure 3:

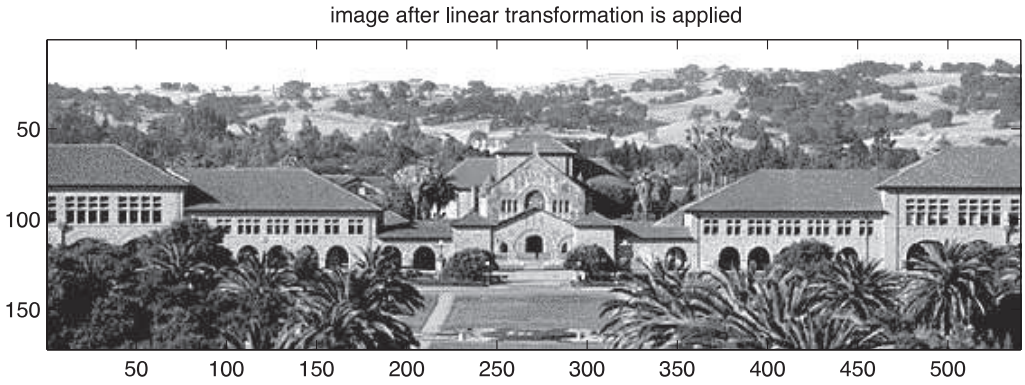

Figure 4:

(c) The transformed image is shown in Figure 4. As in Problem 1, the grayscale display was set such that black was assigned the value 0 and white 255. Also, pixel values that are greater than 255 were clipped to 255, while pixel values less than zero were set to 0.

(d) The original image has low contrast because most of the pixel values are confined to a narrow region, of width approximately 26, centered at about 70. Since white was assigned the value 255 and black was assigned 0, this explains why the original image looks dark (70 is relatively low). In comparison, the image after the linear transformation is applied displays much better contrast. This is because the pixel values make better use of the 0-255 range.

Problem 3: [20 pts]

As observed in Problem 2, the original image *image1* possesses a low contrast ratio, making it difficult to observe details. We saw that the linear transformation dramatically improved the contrast as well as the overall brightness level (the mean of pixel values). Consequently, our criteria for choosing the "best" p in the transformation

$$\text{out}(i,j) = \text{in}(i,j)^p \tag{2}$$

is based on the range of pixel values after transformation. We seek, mainly, to stretch the input distribution in order to improve the contrast ratio. This, of course, necessitates that $p > 1$. Figure 5 plots the (sorted) distribution of pixel values for

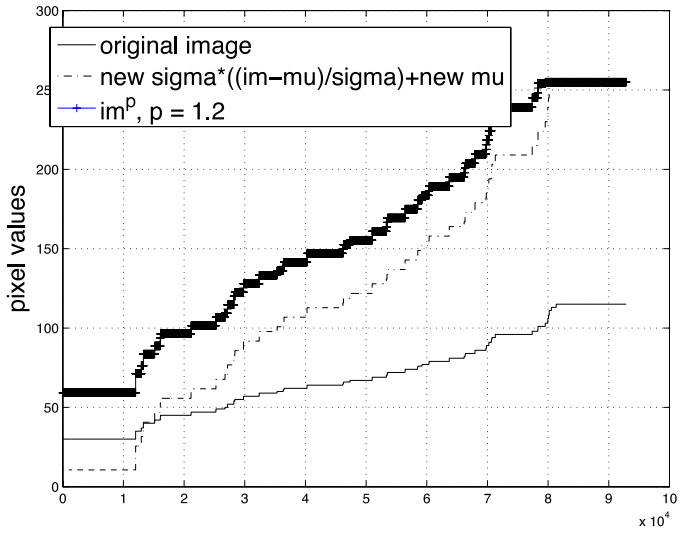

Figure 5:

the original, linearly-transformed and nonlinearly-transformed (with $p = 1.2$) images. We note that the nonlinear transformation, with $p = 1.2$ succeeds in stretching the range of pixel values. In comparison with the linearly transformed image, however, the nonlinearly-transformed image possesses a somewhat higher mean. The nonlinearly-transformed image is shown in Figure 6

Problem 4:[20 pts]

(a) The MATLAB code we use to generate the 10x10 array of pseudorandom numbers is given at the back, as is the code for smoothing the array. Figure 7 shows the contour map of this 2D array of numbers, with contours spaced evenly at 10, 20, 30, 40 and 50

(b) We observe 3 tops, at pixel locations (10,3), (4,5) and (7,8)

(c) We see that the contour at level 40, towards the left of the image, crosses itself indicating the presence of a saddle point. There appears to be a local minimum, towards the center of the image, between the two highest regions

Problem 5:[20 pts]

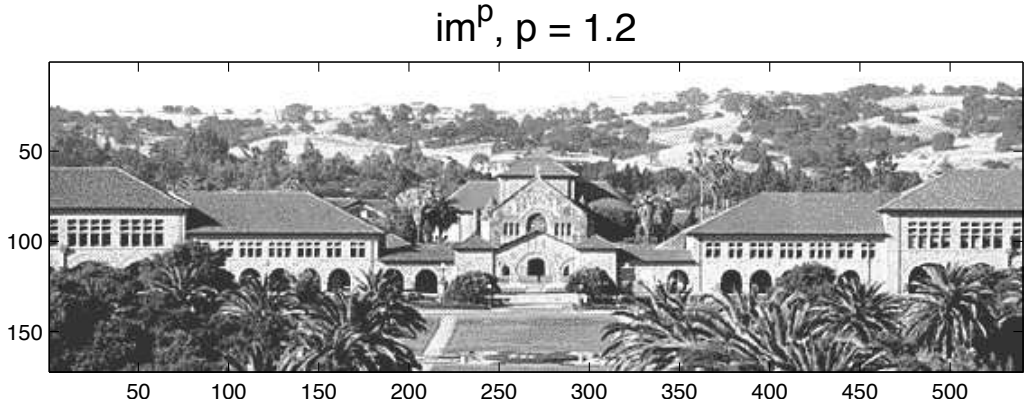

Figure 6:

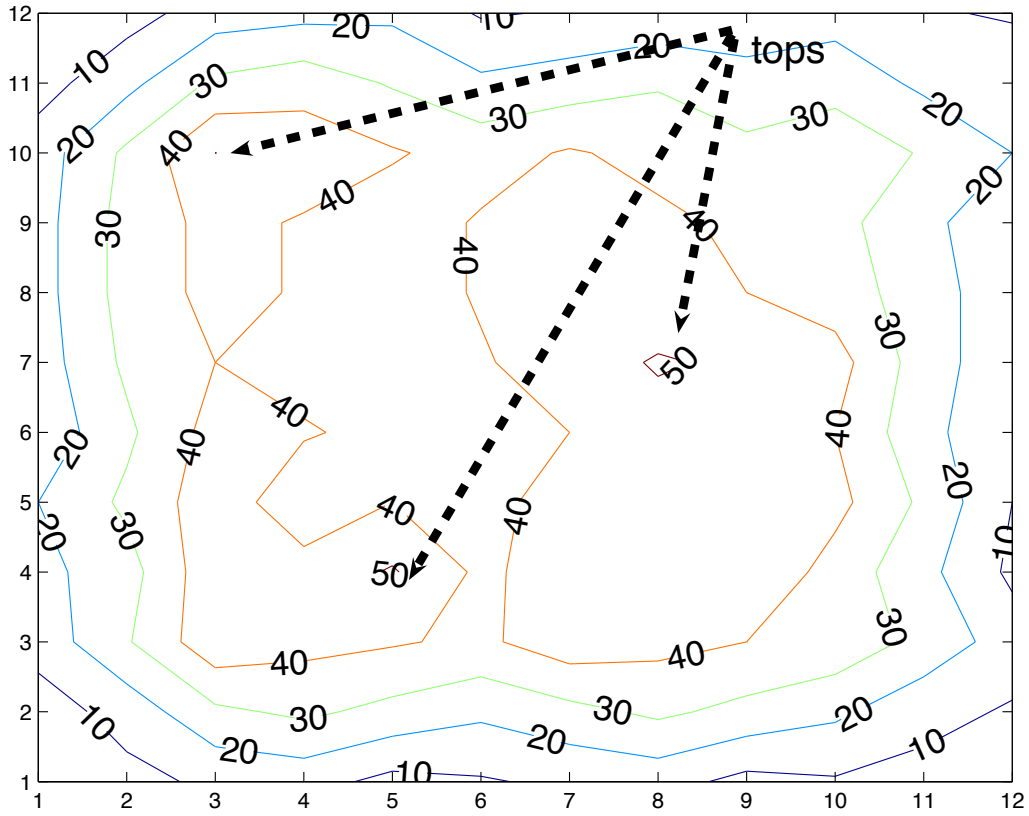

Figure 7:

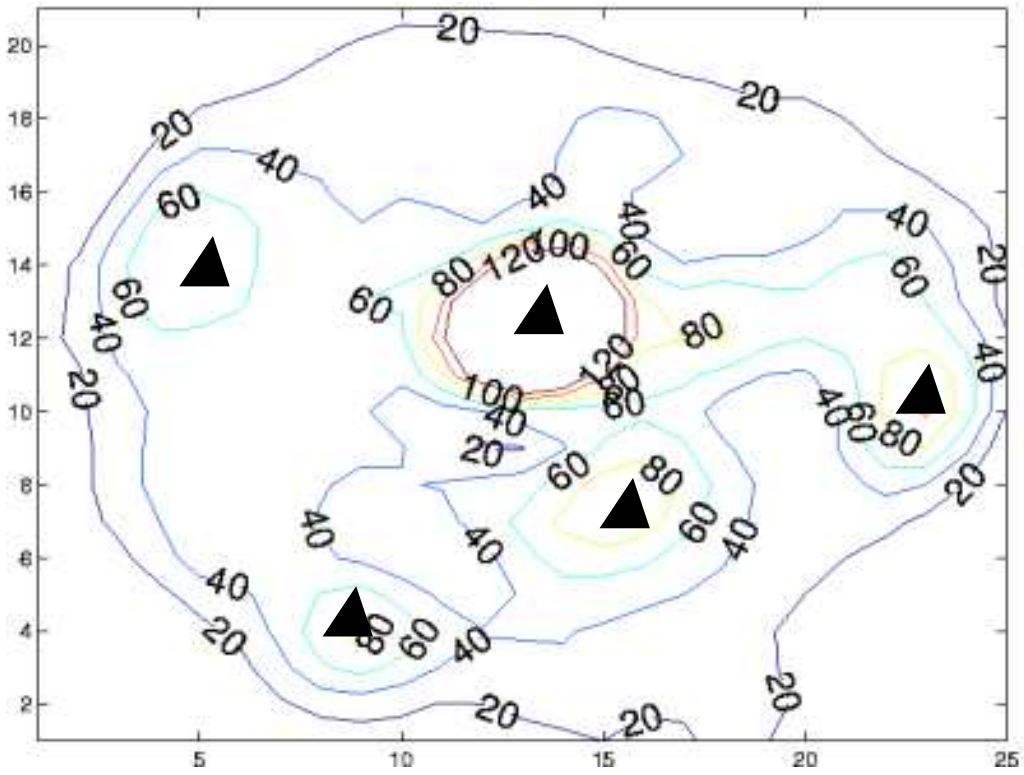

Figure 8:

(a) Figure 8 shows a contour map of sun microwave temperature at contour level spacings 20,40,60...

(b) The 5 tops are indicated by black triangles in Figure 8.

(c) The concave region is indicated by the shaded area in Figure 9

(d) The average Earth-Sun distance, d is 149,597,892 km, while the average Sun radius, r is 695,000 km. Figure 10 shows a diagram we use to compute arc spacing between elements. The number of elements across the 2D array of numbers given in the Bracewell book is 25. Hence, the microwave temperature is sampled every $2 \times r/25 = 55,600$ km across the disk of the Sun. Thus, the approximate angular spacing in radians of one element is $55600/d = 3.71 \times 10^{-4}$ radians or 0.0213 degrees. As there are 60 minutes in a degree of arc, the arc spacing in minutes of one element is

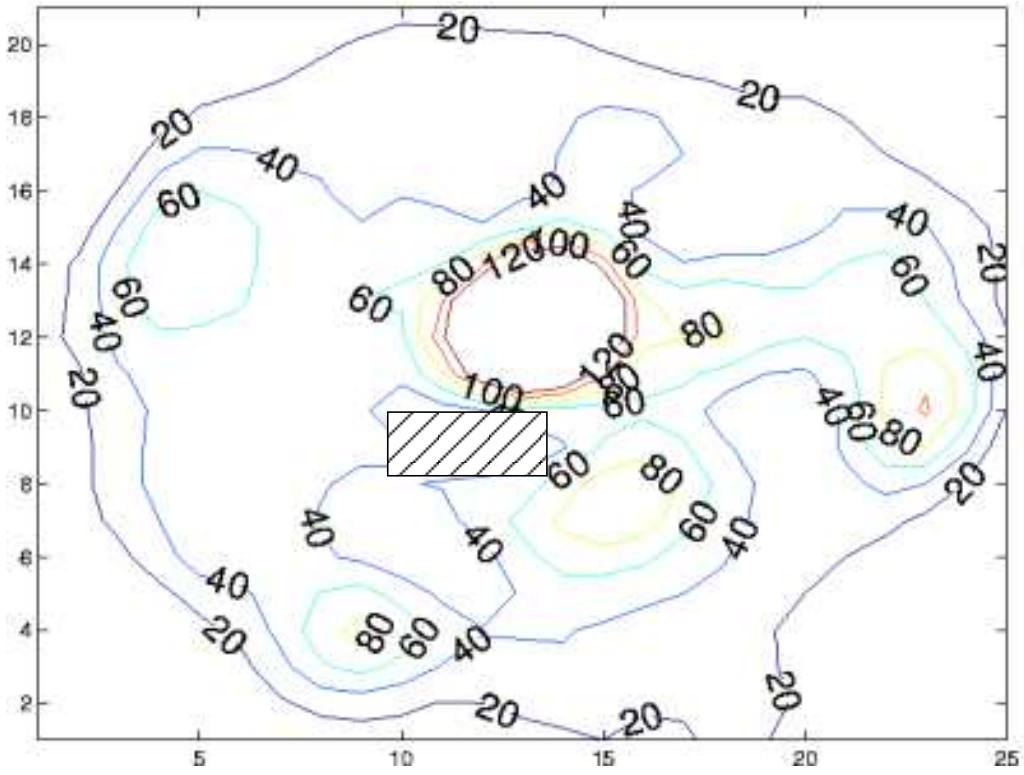

Figure 9:

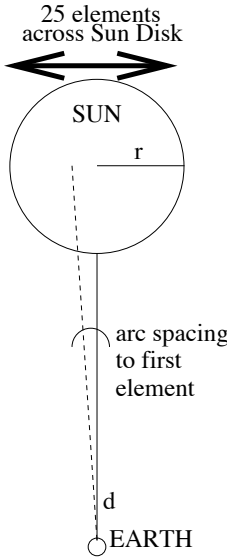

Figure 10: Diagram showing Earth-Sun geometry for computing spacing in minutes of arc of each element of the 2D array of Solar microwave temperature measurements

$$0.0213 \times 60 = 1.27$$

Problem 6:[20 pts]

To form a map of Earth's topography from the 1-dimensional array of altimeter measurements, it is essential to calculate the latitude and longitude (ϕ, θ) of every element in the *delays* file. Having the latitude and longitude coordinates of every measurement in the file will allow us to find their corresponding locations in the final 360x720 map.

We assume that the satellite is in a polar orbit. The satellite velocity is given by

$$v = \sqrt{\frac{\mu}{r}} = 3.0708 \text{ km/s} \tag{3}$$

where $\mu = 3.986004 \times 10^14$ m^{3}s^{-2}, the geocentric gravitational constant, $r = 42270426.73$ meters, the orbit radius. The orbit period is then

$$T = \frac{2\pi r}{v} = 86,490 \text{ seconds} \approx 1 \text{ day} \tag{4}$$

We know that the sampling period between consecutive measurements is 20 s. In 20 seconds, the change in latitude $\Delta\phi$ experienced by the moving satellite is

$$\Delta\phi = \frac{v \times 20 \text{ seconds}}{r} = 0.00145 \text{ radians} \tag{5}$$

Furthermore, the change in longitude $\Delta\theta$ experienced by the moving satellite can be computed from the rotational period of the Earth. Since the Earth takes 24 hours $\times$ 3600 secs per hour $= 86,400$ secs rotate 360°, then

$$\Delta\theta = \frac{2\pi \times 20 \text{ seconds}}{86,400} = 0.001454 \text{ radians} \tag{6}$$

Now that we have computed the change in latitude and longitude, we can calculate the latitude and longitude coordinate for element number k as follows

$$\begin{aligned} \phi(k) &= \phi(k-1) + \Delta\phi \\ \theta(k) &= \theta(k-1) + \Delta\theta \end{aligned}$$

However, we need to take care in "wrapping" the values of ϕ and θ appropriately so that the latter is always between -90° and 90° and the former between -180° and 180°. In particular,

$$\begin{aligned} \text{if } \phi(k) > 90^\circ &\Rightarrow \phi(k) = 180^\circ - \phi(k) \\ \text{if } \phi(k) < -90^\circ &\Rightarrow \phi(k) = -180^\circ - \phi(k) \end{aligned} \tag{7}$$

Also, we must bear in mind that every time the satellite flies over a Pole, the longitude of its ground track "jumps" 180°. That is,

$$\begin{aligned} \text{if } \phi(k) > 90^\circ &\Rightarrow \theta(k) = \theta(k) + 180^\circ \text{ MOD } 180^\circ \\ \text{if } \phi(k) < -90^\circ &\Rightarrow \theta(k) = \theta(k) + 180^\circ \text{ MOD } 180^\circ \end{aligned} \tag{8}$$

where MOD refers to modulus after division. The preceding equations will ensure that the calculated latitude and longitude stay between $[-90^\circ, 90^\circ]$ and $[-180^\circ, 180^\circ]$ respectively.

Equations 8 and 9 allow us to calculate the latitude and longitude coordinates of every single element in the *delays* data array. The objective now is to find the correct (i, j) indices, in the 360 x 720 image, corresponding to each element. We find the image indices based on the computed geographical coordinates. Consider

$$\begin{aligned} i &= \lceil 180 \times (\frac{\phi}{90^\circ} + 1) \rceil \\ j &= \lceil 360 \times (\frac{\phi}{180^\circ} + 1) \rceil \end{aligned} \tag{9}$$

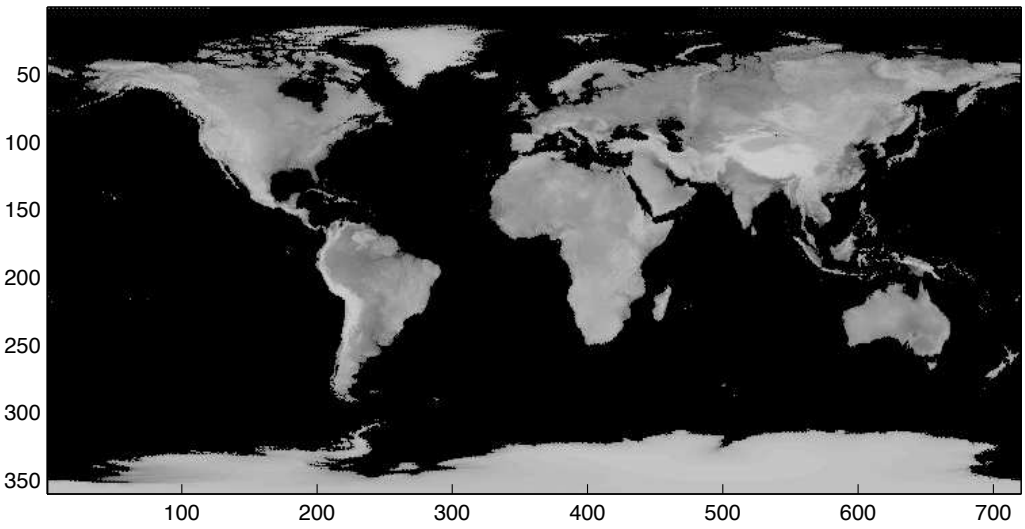

Figure 11:

The role of the ceiling operators, $\lceil . \rceil$ in Equation 10 above will cause more than one measurement in the data file to be mapped to the same image location. Alternatively, one could also average all these overlapping points.

Finally, the delay values in nanoseconds need to be converted to height values. The reason for the negative delays is due to the fact a constant value has been subtracted from the data file. This constant value corresponds to the delay of a signal traveling to and reflecting from a purely spherical Earth (devoid of topography). The factor that we multiply the final image is

$$K = \frac{-c \times 10^{-9} \text{ s}}{2} \tag{10}$$

where c is the speed of light. Figure 11 shows the resulting topography map.

- Madrid (Spain) is located approximately at $(\phi, \theta) = (40.45°, -3.72°) \Rightarrow (i, j) = (100, 353)$ and its elevation is about 634.4 meters
- The Tibetan Plateau is located approximately at $(\phi, \theta) = (32.5°, 87.5°) \Rightarrow (i, j) = (116, 535)$ and its elevation is about 5,448.6 meters
- Greenland is located approximately at $(\phi, \theta) = (75°, -45°) \Rightarrow (i, j) = (31, 271)$ and its elevation is about 2,668.8 meters

MATLAB code for Problem 6

```
% parameters
%-------------%

mu = 3.986004e14;   % gravitational constant
r = 42270426.73;  % orbit radius
re = 6378140;  % Earth radius
h = r - re;

T = 2*pi*r*sqrt(r/mu); % satellite period
v = 2*pi*r/T; % satellite velocity

Ts = 20; % sample period

% increments in latitude and longitude
delta_lat = v*Ts/r;
delta_long = Ts*2*pi/(24*3600);

% image parameters
num_rows = 360; num_cols = 720;

% load in delay file
load delays;

% create latitude and longitude array and initialize
lat = zeros(size(delays)); long = zeros(size(delays));
lat(1) = 0; long(1) = 0; ssign = -1;

% loop through delay file and assign
% lat/long coordinates to each sample
for k=2:length(delays)

  if(mod(k,100000) == 0); disp(k);end

  lat(k) = lat(k-1) + ssign*delta_lat;
  long(k) = long(k-1) + delta_long;
```

```
% 4 possible scenarios
%---------------------%

% (1) Flying South to North over North Pole
if((lat(k-1) <= pi/2) & (lat(k) > pi/2))

  % decrease latitude from 90 degrees to -90
  ssign = -1;
  lat(k) = pi/2 - (lat(k)-pi/2);

  % shift longitude by 180 degrees
  if(long(k) > 0)
    long(k) = long(k) - pi;
  else
    long(k) = long(k) + pi;
  end

end

% (2) Flying North to South over Equator
if((lat(k-1) >= 0) & (lat(k) < 0))
  ssign = -1;rowsign = 1;
end

% (3) Flying North to South over South Pole
if((lat(k-1) >= -pi/2) & (lat(k) < -pi/2))

  % increase latitude from -90 degrees to 90
  ssign = 1;
  lat(k) = -pi/2 - (pi/2 + lat(k));

  % shift longitude by 180 degrees
  if(long(k) > 0)
    long(k) = long(k) - pi;
  else
    long(k) = long(k) + pi;
  end
```

```
  end

  % (4) Flying South to North over Equator
  if((lat(k-1) <= 0) & (lat(k) > 0))
     ssign = 1;
  end


  if(long(k) >= pi)
    long(k) = long(k) - 2*pi;
    col(k) = 1;
  end

end

% compute image row and column from lat/long
row = ceil(num_rows/2*(lat/(pi/2)+1));
col = ceil(num_cols/2*(long/pi+1));

% scale factor
K = -3e8*1e-9/2;

% place delay values in proper image locations
im = zeros(num_rows, num_cols);
im(sub2ind(size(im),row,col)) = K*delays;
```

Extra Credit

We apply the kernel $[1 \; -1]$ to the image matrix from Problem 5. As given in the lecture notes, the image data is convolved with the aforementioned kernel to produce a shaded relief map. The following MATLAB code performs this

```
ker = [1 -1];
shaded_relief = conv2(a,ker);
```

Conv2 is a built-in MATLAB command that performs 2-dimensional convolution. The shaded relief map is shown in Figure 12 We also apply the same kernel to the topography map from Problem 6. The results are shown in Figure 13

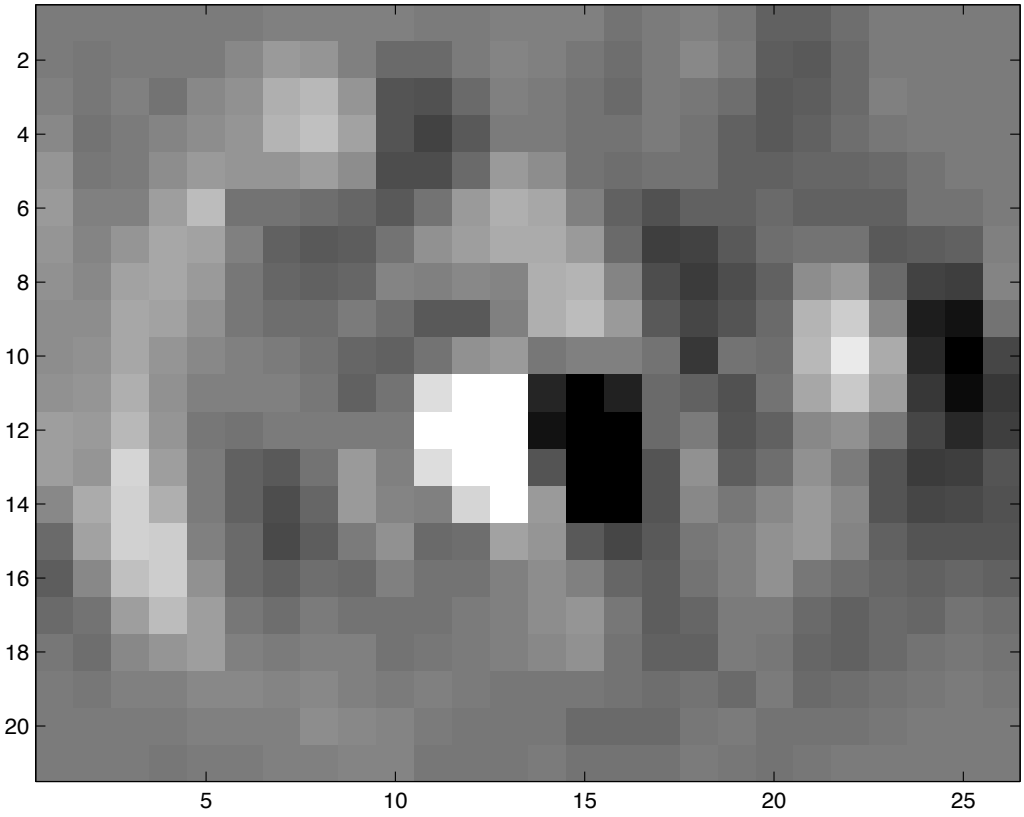

Figure 12:

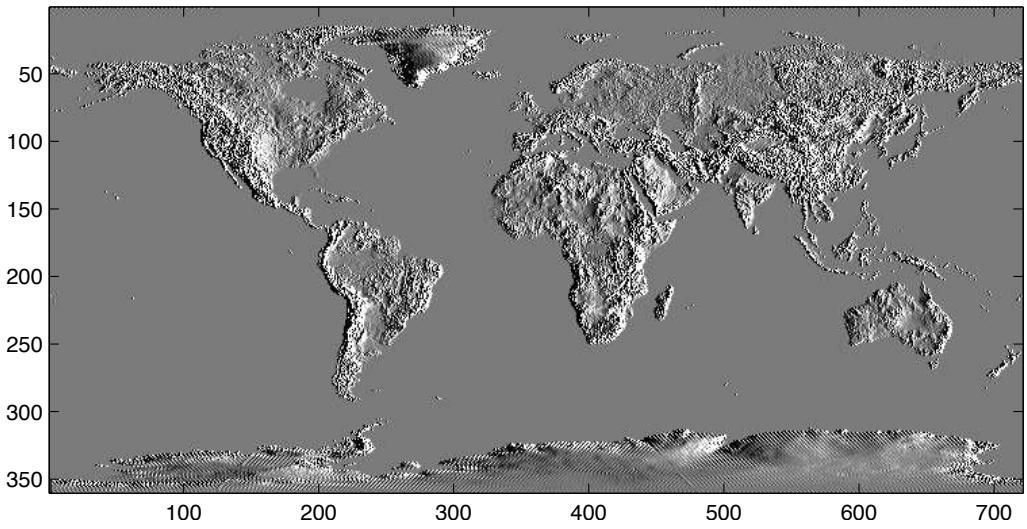

Figure 13: