ASSIGNMENT 2

General Lab Info:

Name: Poornima Manjunath

Assignment no: 2

Due Date: 3/9/2018

Objective:

The aim of the assignment is to perform the image processing using FFT and to do tasks with the frequency domain processing.

In the first part, we are performing 2 D FFT on the image which is centered and obtaining its spectrum. Also, we are re -obtaining the image by doing inverse 2D FFT.

In the second part, we are producing a smoothed, low-pass filtered output image by using the Butter-worth low pass filter.

Background:

The Fourier transform is called the frequency domain representation of the original signal. In DSP we convert a signal into its frequency components, so that we can have a better analysis of that signal. Fourier Transform (FT) is used to convert a signal into its corresponding frequency domain. Initially people used DFT (Discrete Fourier Transform). Later on FFT (Fast Fourier Transform) was created.
If N samples are present, DFT takes N^2 operations while FFT takes only N*log(N) operations. Hence FFT is much faster than DFT. FFT is a simpler and faster method of implementing DFT. This is very useful when the value of N is large.

In this assignment, in part1 we are converting the image information into its frequency components by performing FFT and later the 2D FFT is inverted to obtain the image back.

The second part deals with performing low-pass filtering in the frequency domain.

Filtering:
The low-pass filter should cause edges and other rapid changes in the image to be blurred, while smooth surfaces should be left more or less unaffected. The reason for doing the filtering in the frequency domain is generally because it is computationally faster to perform two 2D Fourier transforms and a filter multiply than to perform a convolution in the image (spatial) domain. This is particularly so as the filter size increases.

Algorithm:

Part1:

ASSIGNMENT 2

1. Read the image file and put it into a 2D array of dimension 256X256.

2. Center the image to the origin by multiplying each pixel by (-1)^(x+y) where x and y are its co-ordinates.

3. Take each row of pixels, pad 0's alternatively, make it a 512 and perform 1D FFT using four1.

4. Put the result of four1 i.e 1D FFT to another array row-wise. After performing 1D FFT on each row of the initial image array, a 256X512 array will be obtained.

5.Take column-wise real and imaginary elements. i.e the first column element will be real part,2nd column element will be imaginary part and build a 512 elements array. Perform four1 function on it.
Put the output of this four1() function in another array column-wise separating the real and imaginary components.

6. By repeating the step 5 on each 2 columns of the 1D FFT array, we will obtain 2D FFT array which is of dimension 256X512. It will be comprised of both real and imaginary components.

7. Take the magnitude of real and imaginary part of each pixel of the 2D FFT array (square root of the sum of real part^2 and imaginary part ^2) mag= abs(sqrt(real*real+imag*imag))
, and putting it to another array of 256X256. This will have the magnitudes of each pixel. This is the spectrum in the unnormalised form.

8. Take each element of the spectrum and normalize it using the formula

Normalized magnitude = 255*((magnitude-min)/(max-min))

9. Put the normalized magnitude into another array of 256X256. Put this array to an image file after casting it to unsigned char.

10. Now the image can be viewed as a centered spectrum (a tiny dot in the center when zoomed out)

11. Take the 2D FFT array again, take column-wise real and imaginary parts and put them to an array of 512 elements. call four1() on it with the -1 sign. i.e it will be inverse fft.

12. Put the output of four1() function with -1 sign, i.e 1st inverse fft, into another array column-wise separating the real and imaginary parts, by repeating this with all the 512 columns we will obtain a 256X512 array which is the output of 1st 1D inverse FFT.

13. Take row wise 512 elements. Put it to a temporary array. Call four1() with -1 sign. This will be the second inverse FFT. Put it to an output array of 2D inverse FFT. Repeat this step with each row, to obtain the 2D inverse FFT.

14. Separate the real and imaginary components just like doing to obtain the spectrum, take the magnitude of each element combining the real and imaginary parts .ie
magnitude = abs(sqrt(real*real+imag*imag))

15. Put this magnitude into a 256X256 array which comprises of un-normalized magnitudes.

ASSIGNMENT 2

Take the normalized values using the normalization formula
Normalized magnitude = 255*((magnitude-min)/(max-min))

16. By putting the normalized magnitudes in the unsigned char form into an array of 256X256 and writing this array to an output file, we will obtain the original image back.

<u>Part2:</u>

1. Read the image file. Put it to a 256X256 array.
2. Center the image by multiplying each pixel by (-1)^(x+y) where x and y are its co-ordinates.
3. Obtain the 2D FFT as mentioned in the part1.
Calculate the power P(u,v) of the original image by taking the summation of squared magnitudes.

4. Separate the real and imaginary components and put them into a real array of 256X256 say R(u,v) and imaginary array I(u,v) of 256X256.

5. Now, take a function of filter in the frequency domain. i.e an array say H(u,v) of 256X256.
H(u,v) = 1/(1+((D(u,v)/D0)^2*n))

where D0 is a constant which determines the amount of blur.
U and v are the co-ordinates. The “n” is the order of the filter.

D(u,v) has to be D(u-N/2,v-N/2) in order to center the Butter-worth filter.

D(u-N/2,v-N/2) = sqrt((u-N/2)^2+(v-N/2)^2)
where N= 256.

6. By calculating the filter function values for each pair of (u,v),we will get H(u,v)

7. Multiply every element of R(u,v) with corresponding element of H(u,v) .Let the resultant array be say R'(u,v)
Similarly multiply every element of I(u,v) with corresponding element of H(u,v) .Let the resultant array be say I'(u,v)

8. Put the resultant real array elements R'(u,v) and the imaginary array I'(u,v) alternatively in a resultant array of 256X512. Let this array be G(u,v).

9. Calculate the power P'(u,v) of the filtered image by taking the summation of squared magnitudes.

Now we can see the difference between the power i.e a function of energy of the original image and the filtered image.

10. Take G(u,v), take the magnitude i.e sqrt(real*real +imag*imag) . Put the magnitudes to an array of 256X256.

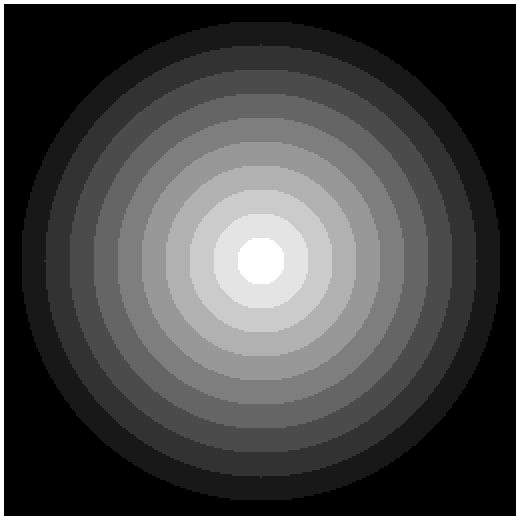
11. Normalize the magnitude using the normalization formula and put them to array of 256X256 by casting to unsigned char.

ASSIGNMENT 2

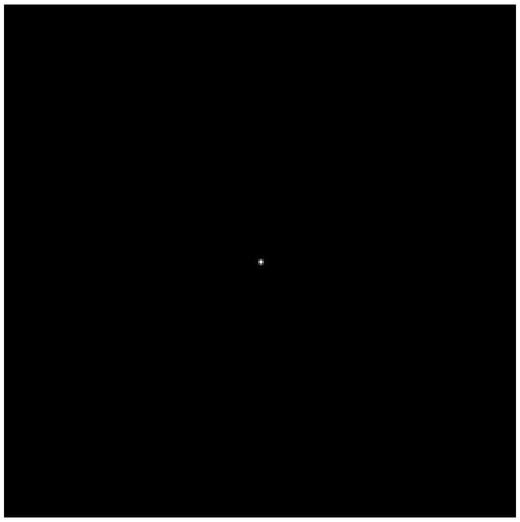
12. By writing the normalized array of unsigned char format to an image file, we can obtained a filtered image file.

Results:

1. original cake image

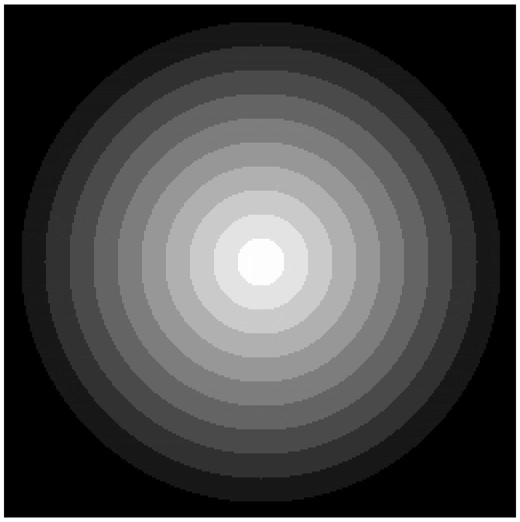


2. Centered spectrum of cake image

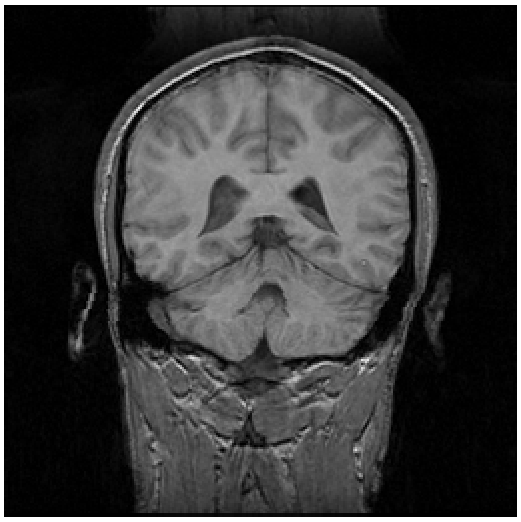


ASSIGNMENT 2

3. Re-obtained cake image

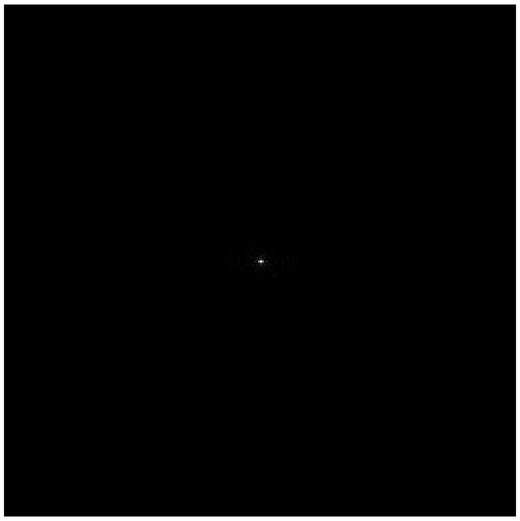


4. Original MRI image

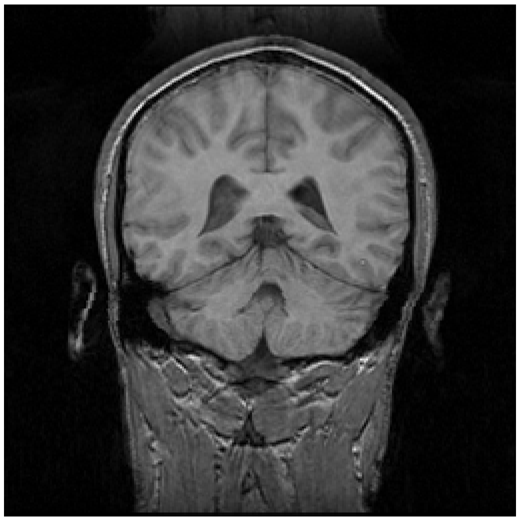


ASSIGNMENT 2

5.Spectrum of mri which is centered



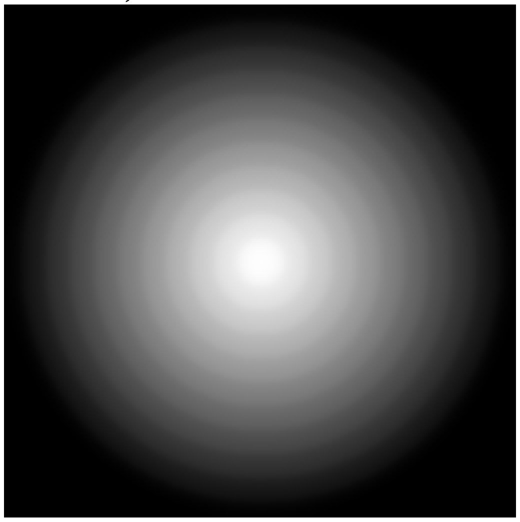
6. Re-obtained image after 2D IFFT



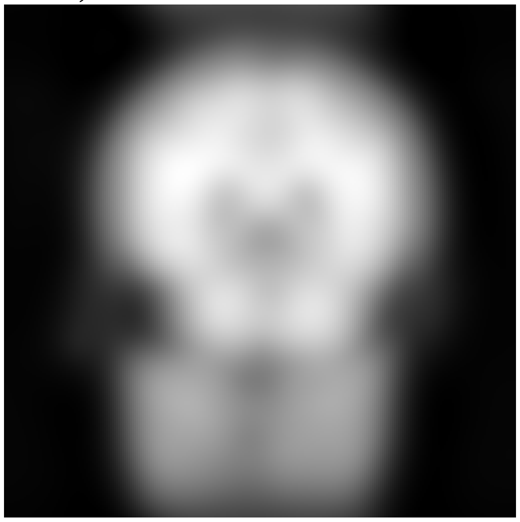
7. Low Pass filtered cake image D0 = 5 ,n=1

ASSIGNMENT 2

8. Low Pass filtered cake image D0 = 10 ,n=1

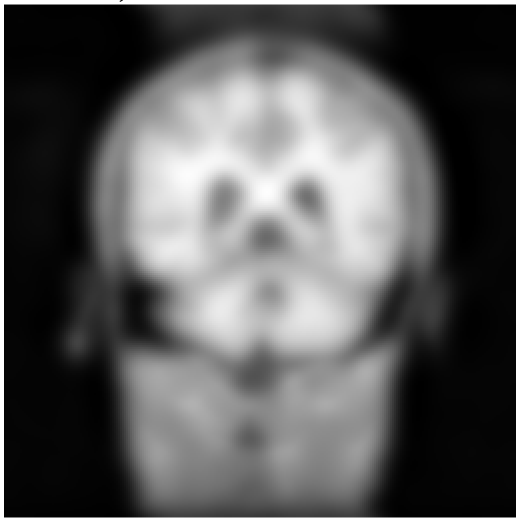


9.Low Pass filtered mri image D0 =5 ,n=2

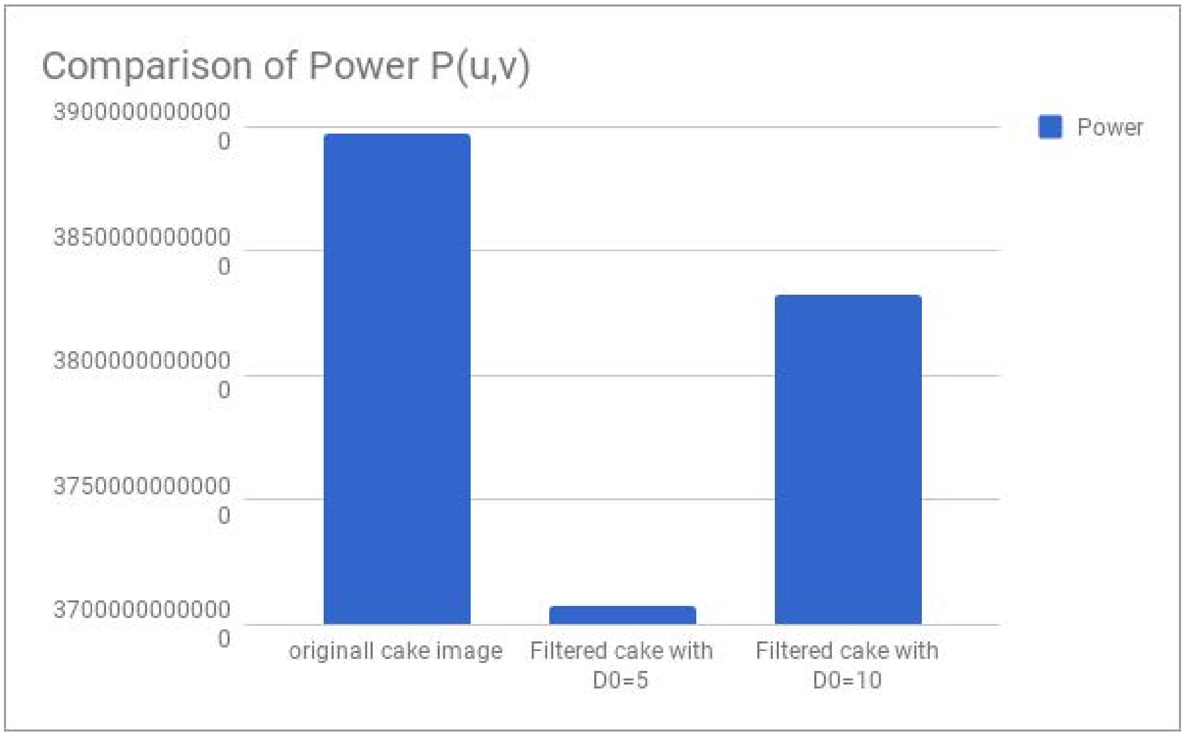


ASSIGNMENT 2

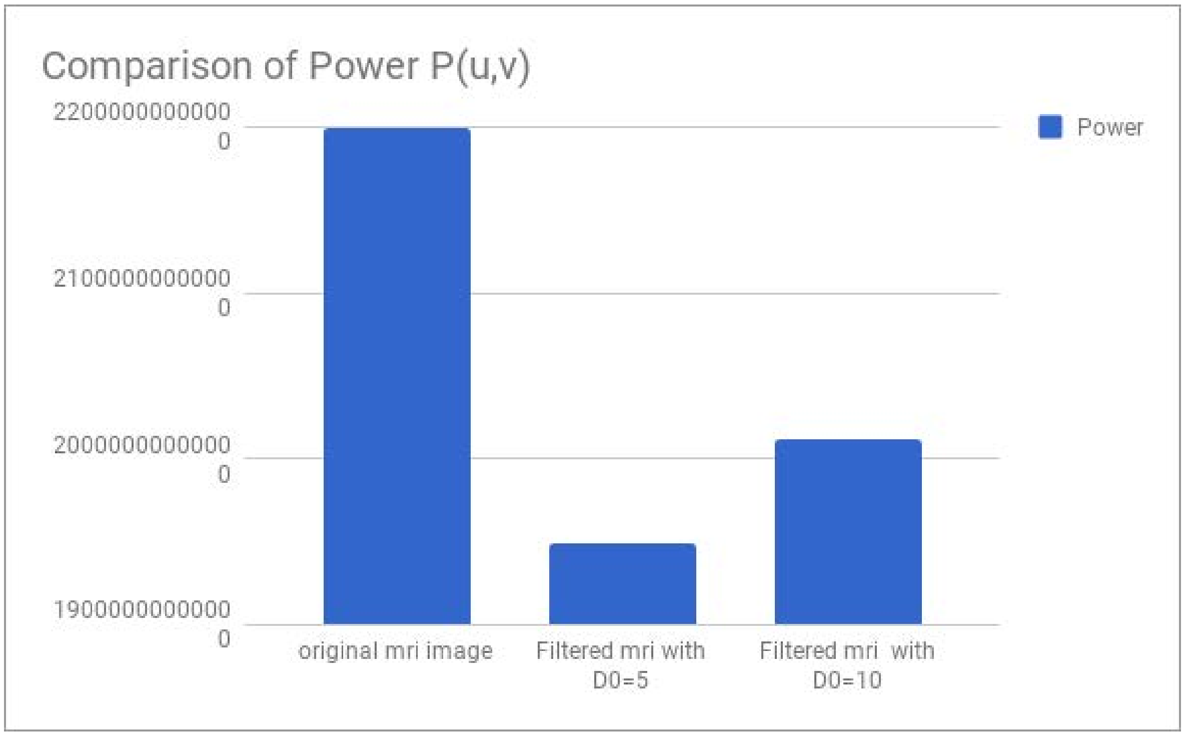
10. Low Pass filtered mri image D0 =10 ,n=2



Graphs:



ASSIGNMENT 2



Observations:

Part 1:

By multiplying each pixel with (-1)^(x+y) and doing 2 D FFT on the image, we will obtain a frequency spectrum which is shifted to place the zero frequency at the center. And by doing inverse 2D FFT, we can re-obtain the original image.

Part 2:

We observe that Butter-worth low-pass filters

1. create a blurred (or smoothed) image
2.attenuate the high frequencies and leave the low frequencies of the Fourier transform relatively unchanged

The Butter-worth filter function is given by

$$H(u,v) = \frac{1}{1+\left[D(u,v)/D_0\right]^{2n}}$$

Also, we observe that with the decreasing D0, the image is getting more blur and the Energy function P'(u,v) is much less than that of the original image P(u,v).

ASSIGNMENT 2

Conclusion:

Part1:

Fourier Transform decomposes an image into its real and imaginary components which is a representation of the image in the frequency domain. If the input signal is an image then the number of frequencies in the frequency domain is equal to the number of pixels in the image or spatial domain. The inverse transform re-transforms the frequencies to the image in the spatial domain.

$$F(x) = \sum_{n=0}^{N-1} f(n) e^{-j2\pi(x\frac{n}{N})}$$

$$f(n) = \frac{1}{N}\sum_{n=0}^{N-1} F(x) e^{j2\pi(x\frac{n}{N})}$$

Since the spatial domain contains a discrete signal, the frequency domain is periodic. In other words, the frequency domain arrays are duplicated an infinite number of times to the left, right, top and bottom. For instance, imagine a tile wall, with each tile being the N×N magnitude. Row N/2 and column N/2 break the frequency spectrum into four quadrants. For the real part and the magnitude, the upper-right quadrant is a mirror image of the lower-left, while the upper-left is a mirror image of the lower-right. This symmetry also holds for the imaginary part and the phase, except that the values of the mirrored pixels are opposite in sign. Frequency spectrum with the spectrum shifted to place zero frequency is also an N×N section of this tile wall, but it straddles four tiles; the center of the image being where the four tiles touch, except it has been shifted N/2 pixels horizontally (either left or right) and N/2 pixels vertically (either up or down) in the periodic frequency spectrum. When the spectrum is displayed with zero frequency at the center, the line from each pair of points is drawn to the DC value at the center of the image, i.e., [N/2, N/2]. This organization is simpler to understand and work with, since all the lines are drawn to the same point. In general, the frequency spectra of discrete images are displayed with zero frequency at the center whenever people will view the data

Part2 :

Frequency filtering is based on the Fourier Transform. The operator usually takes an image and a filter function in the Fourier domain. This image is then multiplied with the filter function in a pixel-by-pixel fashion:

$G(u,v) = F(u,v)\ H(u,v)$

where F(u,v)is the input image in the Fourier domain, H((u,v) the filter function and G(u,v) is the filtered image. To obtain the resulting image in the spatial domain, G(u,v) has to be re-transformed using the inverse Fourier Transform.

Since the multiplication in the Fourier space is identical to convolution in the spatial domain, all frequency filters can in theory be implemented as a spatial filter. But multiplication is an easier task compared to convolution.

We will lose some portion of the power when we do low pass filtering. The smaller the D0, more is the blur and more is the power lost.

Codes

Part1 : To perform the 2-D FFT and obtain the spectrum and do 2-D IFFT and obtain the image back.

```
/*******
 * Program to perform  FFT  of the Image
 * For Computer Vision by Poornima Manjunath
 */
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr
void four1(float *data, int nn, int isign)
{
	int n,mmax,m,i,jstep,j;
	double wtemp,wr,wpr,wpi,wi,theta;
	float tempr,tempi;
	n=nn << 1;
	i=1;
	for (j=1;j<n;j+=2) {
		if (i > j) {
			SWAP(data[i-1],data[j-1]);
			SWAP(data[i],data[j]);
		}
		m=n >> 1;
		while (m >= 2 && i > m) {
			i -= m;
			m >>= 1;
		}
		i += m;
	}
	mmax=2;
	while (n > mmax) {
		jstep=2*mmax;
		theta=4*asin(1)/(isign*mmax);
```

```
            wtemp=sin(0.5*theta);
            wpr = -2.0*wtemp*wtemp;
            wpi=sin(theta);
            wr=1.0;
            wi=0.0;
            for (m=1;m<mmax;m+=2) {
                for (j=m;j<=n;j+=jstep) {
                    i=j+mmax;
                    tempr=wr*data[i-1]-wi*data[i];
                    tempi=wr*data[i]+wi*data[i-1];
                    data[i-1]=data[j-1]-tempr;
                    data[i]=data[j]-tempi;
                    data[j-1] += tempr;
                    data[j] += tempi;
                }
                wr=(wtemp=wr)*wpr-wi*wpi+wr;
                wi=wi*wpr+wtemp*wpi+wi;
            }
            mmax=jstep;
        }
}
float calc_magnitude(float x,float y){
      float t;
      t=(fabs)(sqrt((x*x)+(y*y)));
return t;
}
float calculate_min(float image[256][256])
{
      float min = image[0][0];
    int i,j;
    for(i=0;i<256;i++)
      {
            for(j=0;j<256;j++)
            {
                  if(min>image[i][j])
                  {
                        min = image[i][j];
                  }
            }
      }
      return min;
}
float calculate_max(float image[256][256])
{
      float max = image[0][0];
    int i,j;
    for(i=0;i<256;i++)
      {
```

```
                for(j=0;j<256;j++)
                {
                        if(max<image[i][j])
                        {
                                max = image[i][j];
                        }
                }
        }
        return max;
}

float normalise(float magnitude,float min,float max)
{
        float pixel;
     float intermed;


     intermed = ((magnitude-min)/(max-min));
     pixel = (255*intermed);

         return pixel;
}
int main(int argc, char *argv[]){
  unsigned sizeX=256; //image width
  unsigned sizeY=256; //image height
  unsigned char image[sizeX][sizeY]; //image array
   int  new_image[sizeX][sizeY]; //centered image array
  unsigned levels;
  int val;
  unsigned char norm_magnitude[256][256];
  //read image file
  FILE *fp;
  fp = fopen("cake","r");
  int i,j,k;
  if(fp==0) {
   printf("Error in reading cake image\n");
   return 1;
   }
  if(3!=fscanf(fp, "P5 %d %d %d ", &sizeX, &sizeY, &levels)) return 1;

  fread(image,sizeof(unsigned char),sizeX*sizeY,fp);
  fclose(fp);
  //multiply every pixel by (-1)^(x+y) to center the image
  for(i=0;i<256;i++)
  {
        for(j=0;j<256;j++)
     {
          val = pow(-1,i+j);
```

```
                new_image[i][j] = val*image[i][j];
        }
 }
//create an array for 512 complex numbers putting Imaginary numbers as 0;
float temp[512];
float transform1[256][512];  //2d array F(x,v)

for(i=0;i<256;i++)
{
        k =0;
        for(j=0;j<256;j++)
        {
                temp[k] = new_image[i][j];
                k++;
        temp[k]=0;
        k++;
        }
        four1(temp,256,1); //call the fourier transform on the temp

      for(j=0;j<512;j++)    //put the array to the F(x,v)
        {

        transform1[i][j] = temp[j];
        }

 }
 //take columnwise 2 elements and create a 512 array to perform fft one more time

 float temp2[512];
 float transform2[256][512]; //2D array F(u,v)
 unsigned int n =0;
 unsigned int j_init = 0;
 while(j_init<512)
 {
        k=0;
        for(i=0;i<256;i++)
        {


                j = j_init;
                temp2[k] = transform1[i][j];
                j++;
                k++;
                temp2[k] = transform1[i][j];
                k++;
       }
        four1(temp2,256,1); //call the fourier transform on the temp2 array

        k=0;
```

ASSIGNMENT 2

```
    //put the output of the array columnwise with real and imaginary parts
       for(i=0;i<256;i++)
       {

                j = j_init;
                transform2[i][j]= temp2[k];
                j++;
                k++;
                transform2[i][j]= temp2[k];
                k++;
     }
  j_init+=2;
}

     //writing the magnitude  to the spectrum array
     double x,y;
     float spectrum[256][256];
     unsigned char pixel_value;
        j_init = 0;
     int p = 0;
        while(j_init<512)
     {

                for(i=0;i<256;i++)
                {
                         j = j_init;
                x = transform2[i][j];
                y = transform2[i][j+1];
                         spectrum[i][p] =calc_magnitude(x,y);
                }
                p++;

                j_init+=2;
        }
        //printf("i= %d\t p= %d\n",i,p);
        //for(i=0;i<256;i++)
    //{
        //      for(j=0;j<256;j++)
        //      {
        //               printf("%ld\n",spectrum[i][j]);
        //      }
        //}
     //normalising the magnitude and putting into a normalised array with unsigned char type
        float min  = calculate_min(spectrum);
        float max = calculate_max(spectrum);
        //printf("min = %f, max = %f\n",min,max);
    for(i=0;i<256;i++)
        {
         for(j=0;j<256;j++)
         {
                 norm_magnitude[i][j] = (unsigned char) normalise(spectrum[i][j],min,max);
       }
        }


     //write spectrum  to  image file
     FILE *iFile = fopen("spectrum_cake.pgm","w");
     if(iFile==0) return 1; //error handling
     fprintf(iFile, "P5 %d %d %d ",sizeX,sizeY, 255);//write header
     fwrite(norm_magnitude,sizeof(unsigned char),sizeX*sizeY,iFile);//write binary image
     fclose(iFile);
```

//to perform inverse fft take columnwise real and imag elements and put them in a temp array of 512 elements

```
             float temp3[512];
          float inv_transform1[256][512];
          j_init = 0;
             while(j_init<512)
             {
                     k=0;
                     for(i=0;i<256;i++)
                     {

                               j = j_init;
                               temp3[k] = transform2[i][j];
                               j++;
                               k++;
                               temp3[k] = transform2[i][j];
                               k++;
            }
                     four1(temp3,256,-1); //call the inverse fourier transform on the temp3 array
            k=0;
            //put the output of the array columnwise with real and imaginary parts
                     for(i=0;i<256;i++)
                     {

                               j = j_init;
                               inv_transform1[i][j]= temp3[k];
                               j++;
                               k++;
                               inv_transform1[i][j]= temp3[k];
                               k++;
            }
                     j_init+=2;
            }
```

//create an array for 512 complex numbers putting row wise elements of the resultant array of 1st inverse transform operation;

ASSIGNMENT 2

```
        float temp4[512];
        float inv_transform2[256][512];

        for(i=0;i<256;i++)
        {

                for(j=0;j<512;j++)
                {
                        temp4[j] = inv_transform1[i][j];



                }
        four1(temp4,256,-1); //call the inverse fourier transform on the temp4

      for(j=0;j<512;j++)
        {
        inv_transform2[i][j] = temp4[j];
        }

    }
          //writing the magnitude in the noramlised form to the another array just like spectrum

      float inv_spectrum[256][256];
      unsigned char inv_norm_magnitude[256][256];

         j_init = 0;
      p = 0;
         while(j_init<512)
      {

                for(i=0;i<256;i++)
                {
                        j = j_init;
                x = inv_transform2[i][j];
                y = inv_transform2[i][j+1];
                        inv_spectrum[i][p] =calc_magnitude(x,y);
                }
                p++;

                j_init+=2;
        }

      min  = calculate_min(inv_spectrum);
      max = calculate_max(inv_spectrum);
//      printf("min = %f, max = %f\n",min,max);

      for(i=0;i<256;i++)
        {
            for(j=0;j<256;j++)
            {
                inv_norm_magnitude[i][j] = (unsigned char) normalise(inv_spectrum[i][j],min,max);

           }
           }

        //write this normalised array  to output image file
        FILE *oFile = fopen("reobtained_cake.pgm","w");
        if(oFile==0) return 1; //error handling
        fprintf(oFile, "P5 %d %d %d ",sizeX,sizeY, 255);//write header
        fwrite(inv_norm_magnitude,sizeof(unsigned char),sizeX*sizeY,oFile);//write binary image
        fclose(oFile);
  return 0;
}
```

Code 2:

To perform low-pass filtering of the images

```
/*******
 * Program to perform Low-Pass filtering  of the Image
 * For Computer Vision by Poornima Manjunath
 */
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr
void four1(float *data, int nn, int isign)
{
        int n,mmax,m,i,jstep,j;
        double wtemp,wr,wpr,wpi,wi,theta;
        float tempr,tempi;
        n=nn << 1;
        i=1;
        for (j=1;j<n;j+=2) {
                if (i > j) {
                        SWAP(data[i-1],data[j-1]);
                        SWAP(data[i],data[j]);
                }
                m=n >> 1;
                while (m >= 2 && i > m) {
                        i -= m;
                        m >>= 1;
                }
                i += m;
        }
        mmax=2;
        while (n > mmax) {
                jstep=2*mmax;
                theta=4*asin(1)/(isign*mmax);
                wtemp=sin(0.5*theta);
                wpr = -2.0*wtemp*wtemp;
                wpi=sin(theta);
                wr=1.0;
                wi=0.0;
                for (m=1;m<mmax;m+=2) {
                        for (j=m;j<=n;j+=jstep) {
                                i=j+mmax;
                                tempr=wr*data[i-1]-wi*data[i];
                                tempi=wr*data[i]+wi*data[i-1];
                                data[i-1]=data[j-1]-tempr;
                                data[i]=data[j]-tempi;
                                data[j-1] += tempr;
                                data[j] += tempi;
                        }
                        wr=(wtemp=wr)*wpr-wi*wpi+wr;
                        wi=wi*wpr+wtemp*wpi+wi;
                }
                mmax=jstep;
        }
}
float calc_magnitude(float x,float y){
            float t;
            t=(fabs)(sqrt((x*x)+(y*y)));
return t;
}
float calculate_min(float image[256][256])
{
            float min = image[0][0];
        int i,j;
        for(i=0;i<256;i++)
            {
                        for(j=0;j<256;j++)
                        {
                                    if(min>image[i][j])
                                    {
                                                min = image[i][j];
                                    }
                        }
            }
            return min;
}
float calculate_max(float image[256][256])
```

ASSIGNMENT 2

```
{
            float max = image[0][0];
        int i,j;
        for(i=0;i<256;i++)
            {
                        for(j=0;j<256;j++)
                        {
                                    if(max<image[i][j])
                                    {
                                                max = image[i][j];
                                    }
                        }
            }
            return max;
}

float normalise(float magnitude,float min,float max)
{
            float pixel;
        float intermed;


        intermed = ((magnitude-min)/(max-min));
        pixel = (255*intermed);

             return pixel;
}
float calc_filter_value(int i,int j)
{
            float val1,val2,val3,val4,val5;
        float D0 = 5;
        int N= 256;
        int n = 1;
        float filter_val;
        val1 = (i-128)*(i-128);
        val2 = (j-128)*(j-128);
        val3 = sqrt(val1+val2);

        val4 = val3/D0;
        val5 = pow(val4,2*n);

        filter_val = 1/(1+val5);

        return filter_val;
}
int main(int argc, char *argv[]){
  unsigned sizeX=256; //image width
  unsigned sizeY=256; //image height
```

ASSIGNMENT 2

```
unsigned char image[sizeX][sizeY]; //image array
int  new_image[sizeX][sizeY]; //centered image array
unsigned levels;
int val;
unsigned char norm_magnitude[256][256];
//read image file
FILE *fp;
fp = fopen("cake","r");
int i,j,k;
if(fp==0) {
 printf("Error in reading cake image\n");
 return 1;
 }
if(3!=fscanf(fp, "P5 %d %d %d ", &sizeX, &sizeY, &levels))
{
     printf("xxx");
        return 1;
}

fread(image,sizeof(unsigned char),sizeX*sizeY,fp);
fclose(fp);
//multiply every pixel by (-1)^(x+y) to center the image
for(i=0;i<256;i++)
{
        for(j=0;j<256;j++)
     {
             val = pow(-1,i+j);
                   new_image[i][j] = val*image[i][j];
      }
 }
//create an array for 512 complex numbers putting Imaginary numbers as 0;
float temp[512];
float transform1[256][512];  //2d array F(x,v)

for(i=0;i<256;i++)
{
        k =0;
        for(j=0;j<256;j++)
        {
                temp[k] = new_image[i][j];
                k++;
        temp[k]=0;
        k++;
        }
        four1(temp,256,1); //call the fourier transform on the temp

     for(j=0;j<512;j++)    //put the array to the F(x,v)
        {
```

```
        transform1[i][j] = temp[j];
        }

 }
 //take columnwise 2 elements and create a 512 array to perform fft one more time

 float temp2[512];
 float transform2[256][512]; //2D array F(u,v)
 unsigned int n =0;
 unsigned int j_init = 0;
 while(j_init<512)
 {
        k=0;
        for(i=0;i<256;i++)
        {

                j = j_init;
                temp2[k] = transform1[i][j];
                j++;
                k++;
                temp2[k] = transform1[i][j];
                k++;
      }
        four1(temp2,256,1); //call the fourier transform on the temp2 array

        k=0;
     //put the output of the array columnwise with real and imaginary parts
        for(i=0;i<256;i++)
        {

                j = j_init;
                transform2[i][j]= temp2[k];
                j++;
                k++;
                transform2[i][j]= temp2[k];
                k++;
      }
   j_init+=2;
}

 //calculate the energy before filtering

      double R,I;
      double Energy_org = 0;

         j_init = 0;
```

```
    while(j_init<512)
   {

             for(i=0;i<256;i++)
             {
                       j = j_init;
             R = transform2[i][j];
             I = transform2[i][j+1];
                       Energy_org= Energy_org+((R*R)+(I*I));
             }


             j_init+=2;
     }

//2D FFT F(u,v) has been obtained ;Now the filter part
float real_array[256][256];    //seperating the real and imaginary parts
float imag_array[256][256];

float H[256][256];

for(i=0;i<256;i++)
{
    k =0;
       for(j=0;j<512;j=j+2)
    {
             real_array[i][k] = transform2[i][j];

          k++;
     }
}

for(i=0;i<256;i++)
{
    k =0;
       for(j=1;j<512;j=j+2)
    {
             imag_array[i][k] = transform2[i][j];
          k++;
     }
}

for(i=0;i<256;i++)
{
       for(j=0;j<256;j++)
    {

             H[i][j] = calc_filter_value(i,j);
```

```
        }
 }

 float temp_real[256][256];    //multiplication result of real part and the filter value
 float temp_imag[256][256];    //multiplication result of imag part and the filter value
 float filter_fft[256][512];  // filtered  o/p array consisting of both real and imaginary parts
 for(i=0;i<256;i++)              //multiply real type by the Filter function
 {
        for(j=0;j<256;j++)
     {

                  temp_real[i][j] =  real_array[i][j]*H[i][j];
        }
 }

 for(i=0;i<256;i++)          //multiply imaginary part by the Filter function
 {
        for(j=0;j<256;j++)
     {

                  temp_imag[i][j] =  imag_array[i][j]*H[i][j];
        }
 }
//put both the real and imaginary components  into another single  array
for(i=0;i<256;i++)
{
    k =0;
    for(j=0;j<512;j++)
    {
                  filter_fft[i][k] = temp_real[i][j];
                  k++;
                  filter_fft[i][k] = temp_imag[i][j];
                  k++;
        }
}
    //calculate the Energy after filtering
     double Energy_filtered = 0;

         j_init = 0;

         while(j_init<512)
      {

                  for(i=0;i<256;i++)
                  {
                           j = j_init;
                  R = filter_fft[i][j];
                  I = filter_fft[i][j+1];
```

```
                        Energy_filtered=Energy_filtered+((R*R)+(I*I));
                }

                j_init+=2;
        }
//writing the magnitude  to the spectrum array
     double x,y;
     float spectrum[256][256];
     unsigned char pixel_value;
        j_init = 0;
     int p = 0;
        while(j_init<512)
     {

                for(i=0;i<256;i++)
                {
                        j = j_init;
                 x = filter_fft[i][j];
                 y = filter_fft[i][j+1];
                        spectrum[i][p] =calc_magnitude(x,y);
                }
                p++;

                j_init+=2;
        }
        //printf("i= %d\t p= %d\n",i,p);
        //for(i=0;i<256;i++)
     //{
        //      for(j=0;j<256;j++)
        //      {
        //              printf("%ld\n",spectrum[i][j]);
        //      }
        //}
         float min  = calculate_min(spectrum);
         float max = calculate_max(spectrum);

     //put the magnitude in the normalised form to an output array which is of unsigned char
     for(i=0;i<256;i++)
        {
          for(j=0;j<256;j++)
          {
                 norm_magnitude[i][j] = (unsigned char) normalise(spectrum[i][j],min,max);
        }
        }
        //write spectrum  to  image file
      FILE *iFile = fopen("spectrum_filter_cake.pgm","w");

         if(iFile==0)
        {
                printf("error in opening the image 1\n");
                return 1; //error handling
         }
       fprintf(iFile, "P5 %d %d %d ",sizeX,sizeY, 255);//write header
       fwrite(norm_magnitude,sizeof(unsigned char),sizeX*sizeY,iFile);//write binary image
       fclose(iFile);
          //to perform inverse fft take columnwise real and imag elements and put them in a temp array of
512 elements
          float temp3[512];
       float inv_transform1[256][512];
       j_init = 0;
          while(j_init<512)
          {
                k=0;
                for(i=0;i<256;i++)
                {

                        j = j_init;
                        temp3[k] = filter_fft[i][j];
                        j++;
                        k++;
                        temp3[k] = filter_fft[i][j];
                        k++;
        }
                four1(temp3,256,-1); //call the inverse fourier transform on the temp3 array
        k=0;
        //put the output of the array columnwise with real and imaginary parts
                for(i=0;i<256;i++)
                {

                        j = j_init;
                        inv_transform1[i][j]= temp3[k];
                        j++;
                        k++;
                        inv_transform1[i][j]= temp3[k];
                        k++;
        }
                j_init+=2;
        }
    //create an array for 512 complex numbers putting row wise elements of the resultant array of 1st
inverse transform operation;
        float temp4[512];
        float inv_transform2[256][512];

        for(i=0;i<256;i++)
        {
                for(j=0;j<512;j++)
                {
                        temp4[j] = inv_transform1[i][j];

                }
        four1(temp4,256,-1); //call the inverse fourier transform on the temp4

      for(j=0;j<512;j++)
        {
        inv_transform2[i][j] = temp4[j];
        }

     }
      //writing the magnitude in the noramlised form to the another array just like spectrum

       float inv_spectrum[256][256];
       unsigned char inv_norm_magnitude[256][256];

          j_init = 0;
       p = 0;
          while(j_init<512)
       {

                for(i=0;i<256;i++)
                {
                        j = j_init;
                x = inv_transform2[i][j];
                y = inv_transform2[i][j+1];
                        inv_spectrum[i][p] =calc_magnitude(x,y);
                }
                p++;

                j_init+=2;
        }

      min  = calculate_min(inv_spectrum);
      max = calculate_max(inv_spectrum);
//      printf("min = %f, max = %f\n",min,max);

      for(i=0;i<256;i++)
        {
          for(j=0;j<256;j++)
          {
                inv_norm_magnitude[i][j] = (unsigned char) normalise(inv_spectrum[i][j],min,max);

        }
        }
       //write this normalised array  to output image file
       FILE *oFile = fopen("filtered_cake.pgm","w");
       if(oFile==0)
         {
                printf("Error in opening the image2\n");

                return 1; //error handling
         }
       fprintf(oFile, "P5 %d %d %d ",sizeX,sizeY, 255);//write header
       fwrite(inv_norm_magnitude,sizeof(unsigned char),sizeX*sizeY,oFile);//write binary image
       fclose(oFile);
      printf("The P(u,v) i.e the power of original image is %f and the power of the image after filtering
is %f\n",Energy_org,Energy_filtered);
      double diff = Energy_org - Energy_filtered;
      printf("The difference in the power P(u,v) is %f\n",diff);
   return 0;
}
```

ASSIGNMENT 2