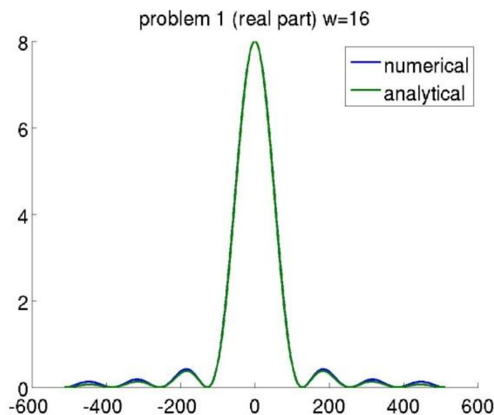
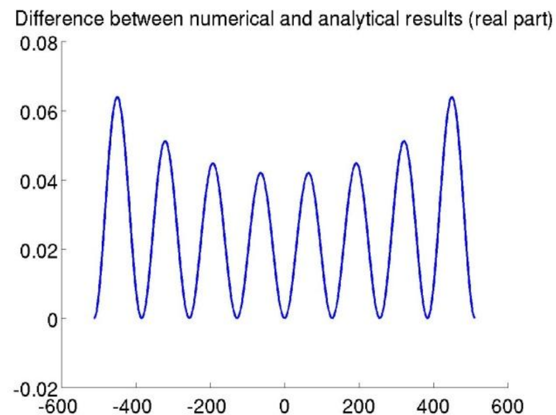
EE 257 Homework 5 Solutions by Ann Chen

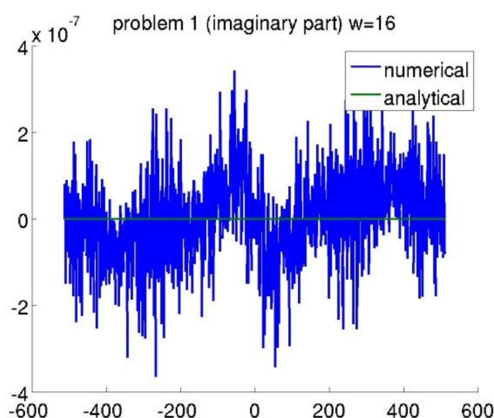
Note: In this solution, we use MATLAB only for plotting.

Problem 1

Here, we only provide the results of the triangle function with width 16. The plots with different widths are similar to the plots below.







Notice that the DFT of the discrete signal is only an approximation of the Fourier transform (the integral is approximated as a sum) of the (sampled) continuous signal and you should see some small symmetric mismatch between the numerical and analytical solution, especially in the highest frequencies.

The triangle function in the time domain should be symmetric. Here is how we may define a triangle function of width 16 (Fortran 90):

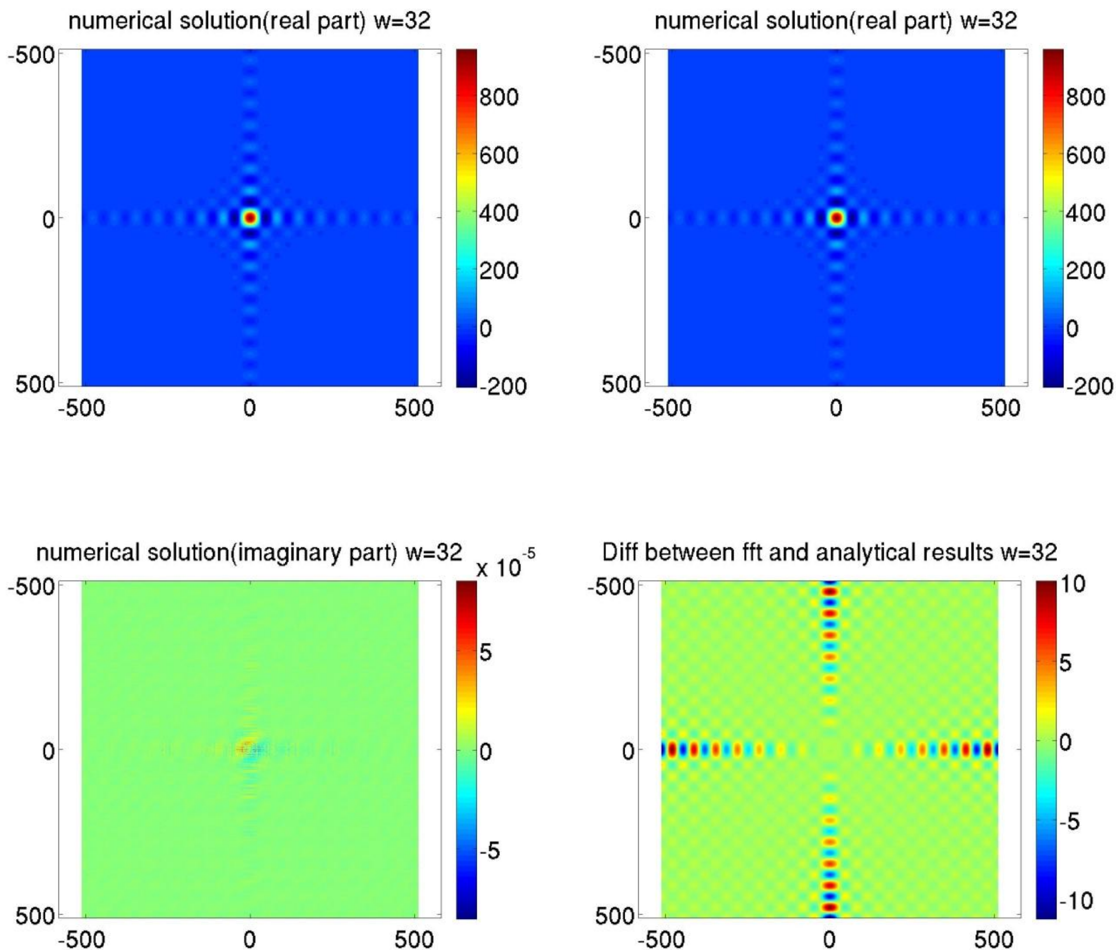
```
!triangle function
  N=1024
  w=16
  a=real(w/2)!scaling parameter
  cindx=N/2+1!Center index
  allocate(s(N))
  do i=1,N
     if(i<cindx+a.AND.i>cindx-a)then
        s(i)=cmplx((a-abs(i-cindx))/a,0)
     else
        s(i)=cmplx(0,0)
```

```
    endif
  end do
```

Problem 2

Again, we observe symmetric mismatch between the numerical and analytical solution, especially in the highest frequencies.

2D rect function:



The rect function is symmetric and contains jumps. As a result, it is tricky to define a discrete rect signal with even width. Here is how we may define a 2D rect function of width 32 (Fortran 90):

```
! 2D rect function
  N=1024 !fft length in both dimension
  w=32
  a=w/2!scaling parameter
  cindx=N/2+1!Center index
  allocate(s(N,N))
```

```
do i=1,N
   do j=1,N
      if(i<cindx+a.AND.i>cindx-a.AND.j<cindx+a.AND.j>cindx-a)then
         s(i,j)=cmplx(1,0)
      else
         s(i,j)=cmplx(0,0)
      endif
   enddo
end do
```

Note that by using the above definition, we have w-1 non zeros in one dimension. We should compare the FFT results with the analytical solution of width (w-1) for a perfect match.

There are other ways to define a 2D rect function. However, the difference between the FFT and analytical solution should look similar to the figure above if there is no mismatch between the numerical and analytical solution.

2D triangle function:

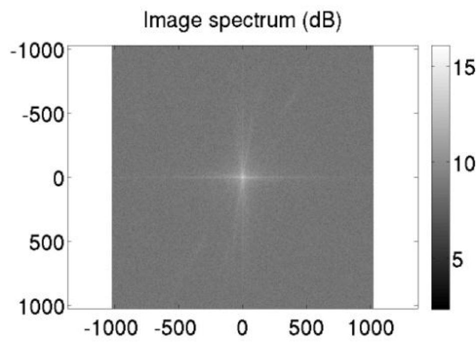
A 2D triangle function with width w is defined as:

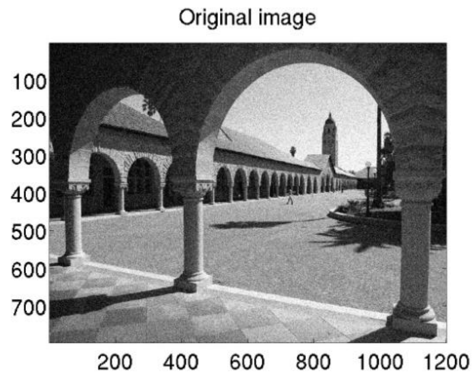
$$2Dtria(x,y)=1Dtria(x)* 1Dtria(y)$$

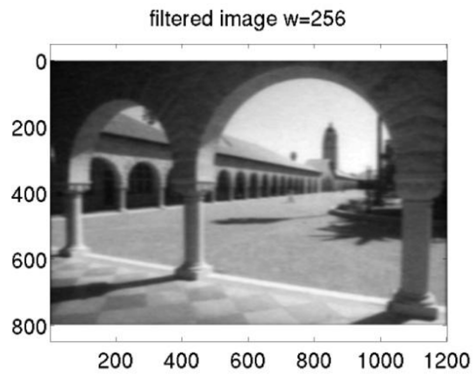
Here, * denotes 'multiply'; 1Dtria(x) and 1Dtria(y) are two 1D triangle functions with width w (please refer to problem 1 for the definition of a 1D triangle function).

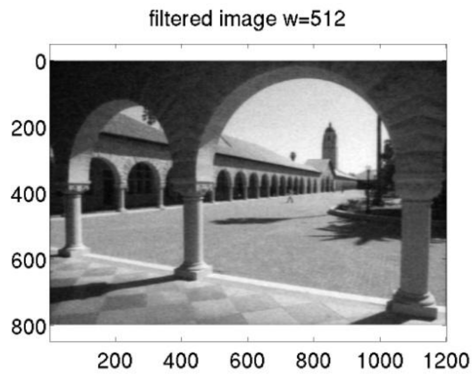
As the triangle function does not contain jumps, it is less tricky to match the numerical and analytical solution. . We should compare the FFT results with the analytical solution of the same width (w) for a perfect match.

Problem 3



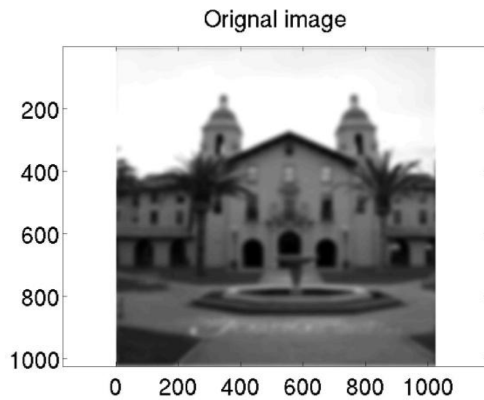


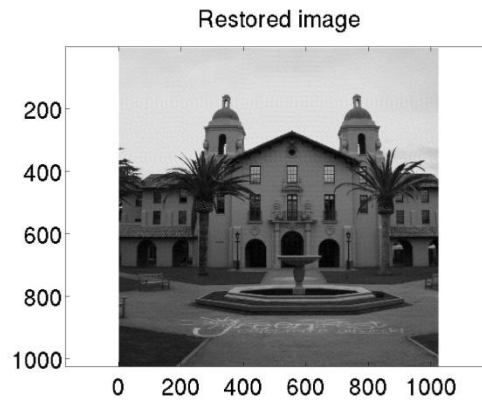


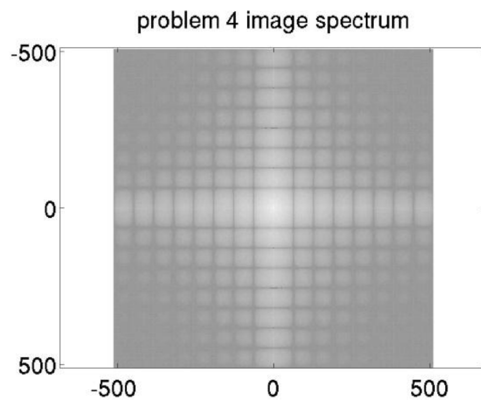


We filter our image using a 2D triangle function in **frequency domain**. As we can see, w=256 leads to a blurry image and w=512 seems to be a good choice. The filtered images are less contaminated by high frequency noises comparing to the original image.

Problem 4







Bandwidth calculation:

We observe that there are 128 samples in the main lobe.

We know $\Delta f=1/1024$,

BW= 128/1024 =1/8 (cycles/sample point)

Inverse triangle filter:

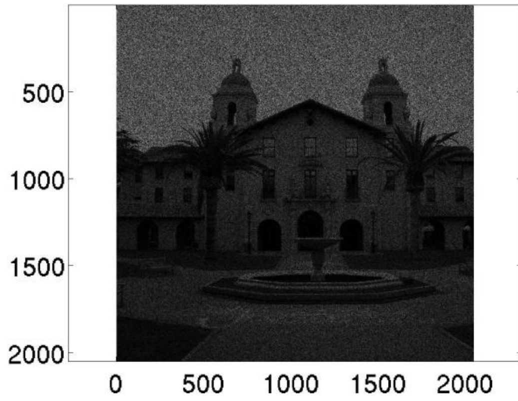
(1) Define a 2D triangle function in time domain

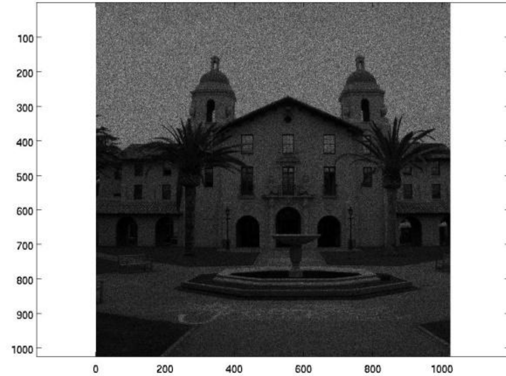
(2) FFT the 2D triangle function and the image

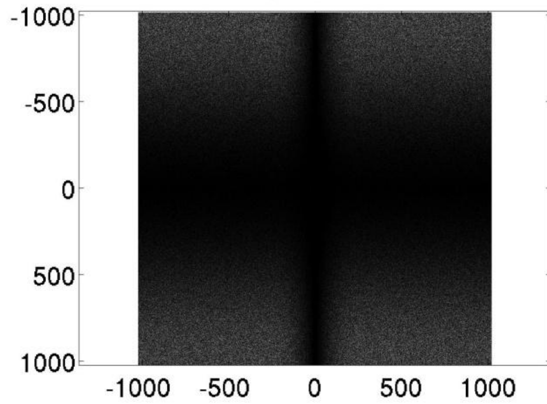
(3) Apply the inverse filter as:

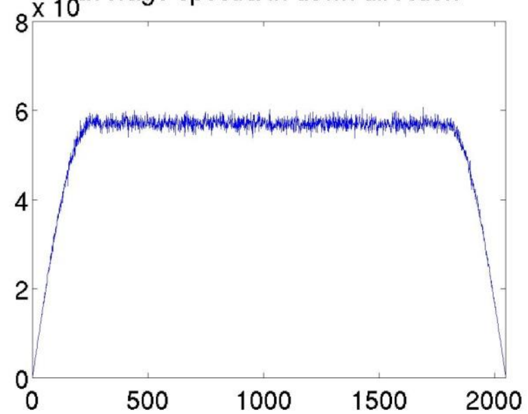
```
do i=1,n
     do j=1,n
        if(abs(tri_fft(i,j))<0.0001)then
           ! Do not filter if triangle function is close to zero
           im_fft(i,j)=im_fft(i,j)
        else
           im_fft(i,j)=im_fft(i,j)/tri_fft(i,j)
        endif
     enddo
  enddo
```

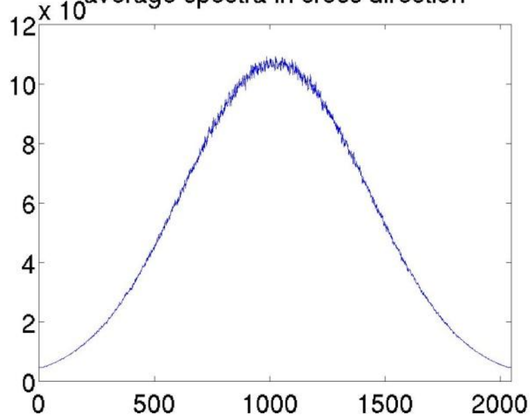
Problem 5











Note: when you average a few pixels with complex number, you should average the absolute value of each pixel (power) rather than average real and imaginary part separately (phase).

Here are the parallel FFT codes in Fortran90 (you can parallel your C++ code in a similar way):

```
!2D fft
  t0=secnds(0.0)
  !$omp parallel do private(i)shared(im,n)
  do i=1,n
     call fft(im(i,:),n,-1)!-1 is the forward fft
  enddo
  !$omp end paralleldo
  !$omp parallel do private(j)shared(im,n)
  do j=1,n
     call fft(im(:,j),n,-1)!-1 is the forward fft
  enddo
  !$omp end paralleldo
  t1=secnds(t0)
  print *,'2D fft uses:',t1,'sec'
```

The non parallel code takes 1.7 sec on an 8 core machine.

The parallel code takes 0.4 sec on the same machine.