Queens College, CUNY, Department of Computer Science

Numerical Methods

CSCI 361 / 761

Spring 2018

Instructor: Dr. Sateesh Mane

April 27, 2018

31 Lecture 31

Fourier Series

- This lecture contains examples of applications the Fast Fourier Transform.
- The Fast Fourier Transform (FFT) is implemented using the code in Lecture 30.
- This lecture requires knowledge of **complex numbers.**

31.1 Aliasing

- Consider the following function, which we shall use as the base case:

$$f_0(x) = 1 + 2\cos(\theta) + 3\sin(\theta) + 4\cos(2\theta) + 5\sin(2\theta) + 6\cos(3\theta) + 7\sin(3\theta)\,. \tag{31.1.1}$$

- We compute the Fast Fourier Transform (FFT) using $n = 2^3 = 8$ points.
- The FFT harmonics F_k are tabulated below:

k	$\Re\{F_k\}$	$\Im\{F_k\}$
0	8	0
1	8	-12
2	16	-20
3	24	28
4	0	0
5	24	28
6	16	20
7	8	12

- Note that the value of $F_0 = 8$ (not 1) because the FFT computes a sum:

$$F_0 = \sum_{j=0}^{n-1} f(\theta_j) = \sum_{j=0}^{7} f_j = 8\,. \tag{31.1.2}$$

- Similarly for all the other harmonics F_k.
- The function values f_j and the inverse FFT $\hat{f}_j$ values are tabulated below.
  The values are all real numbers so the imagianry parts are not tabulated.

j	f_j	$\hat{f}_h$
0	13	13
1	10.2426	10.2426
2	-7	-7
3	5.89949	5.89949
4	-3	-3
5	1.75736	1.75736
6	1	1
7	-13.8995	-13.8995

- Observe that the inverse FFT matches the function values: $\hat{f}_j = f_j$ for all j
- The $1/n$ factor in the inverse FFT takes care of the normalization details.

- Next let us add unwanted harmonics in the interval $k = 5, 6, 7, 8$ and observe the aliasing.
- Consider the following function:

$$\begin{aligned} f_1(x) = f_0(x) &+ 0.07\cos(5\theta) + 0.06\sin(5\theta) \\ &+ 0.05\cos(6\theta) + 0.04\sin(6\theta) \\ &+ 0.03\cos(7\theta) + 0.02\sin(7\theta) \\ &+ 0.01\cos(8\theta)\,. \end{aligned} \tag{31.1.3}$$

- We again compute the Fast Fourier Transform (FFT) using $n = 2^3 = 8$ points.
- The FFT harmonics F_k are tabulated below:

k	$\Re\{F_k\}$	$\Im\{F_k\}$
0	8.08	0
1	8.12	-11.92
2	16.2	-19.84
3	24.28	-27.76
4	0	0
5	24.28	27.76
6	16.2	19.84
7	8.12	11.92

- You should verify that the term in
  $\cos(5\theta)$ affects only $\Re\{F_3\}$, $\sin(5\theta)$ affects only $\Im\{F_3\}$,
  $\cos(6\theta)$ affects only $\Re\{F_2\}$, $\sin(6\theta)$ affects only $\Im\{F_2\}$,
  $\cos(7\theta)$ affects only $\Re\{F_1\}$, $\sin(7\theta)$ affects only $\Im\{F_1\}$,
  $\cos(8\theta)$ affects only $\Re\{F_0\}$.

- Next let us add unwanted harmonics in the interval $k = 9, 10, 11$ and observe the aliasing.
- Consider the following function:

$$\begin{aligned} f_2(x) = f_0(x) &+ 0.001\cos(9\theta) + 0.002\sin(9\theta) \\ &+ 0.003\cos(10\theta) + 0.004\sin(10\theta) \\ &+ 0.005\cos(11\theta) + 0.006\sin(11\theta)\,. \end{aligned} \tag{31.1.4}$$

- We again compute the Fast Fourier Transform (FFT) using $n = 2^3 = 8$ points.
- The FFT harmonics F_k are tabulated below:

k	$\Re\{F_k\}$	$\Im\{F_k\}$
0	8	0
1	8.004	-12.008
2	16.012	-20.016
3	24.02	-28.024
4	0	0
5	24.02	28.024
6	16.012	20.016
7	8.004	12.008

- You should verify that the term in
$\cos(9\theta)$ affects only $\Re\{F_1\}$, $\sin(9\theta)$ affects only $\Im\{F_1\}$,
$\cos(10\theta)$ affects only $\Re\{F_2\}$, $\sin(10\theta)$ affects only $\Im\{F_2\}$,
$\cos(11\theta)$ affects only $\Re\{F_3\}$, $\sin(11\theta)$ affects only $\Im\{F_3\}$.
- The program code is displayed below. The FFT code is given in Lecture 30.

```
void FFT_test()
{
  const double pi = 4.0*atan2(1.0,1.0);
  int num_bits = 3;
  int npts = (1 << num_bits); // n = power of 2

  std::complex<double> X[npts];
  std::complex<double> F[npts];
  std::complex<double> Finv[npts];

  // initialize
  for (int i = 0; i < npts; ++i) {
    F[i] = 0.0;
    Finv[i] = 0.0;
  }

  // initialize data
  double dt = 2.0*pi/double(npts);
  for (int j = 0; j < npts; ++j) {
    double theta = j*dt;
    X[j] = 1;
    X[j] += 2*cos(theta)   + 3*sin(theta);
    X[j] += 4*cos(2*theta) + 5*sin(2*theta);
    X[j] += 6*cos(3*theta) + 7*sin(3*theta);

    // comment out as required
    X[j] += 0.07*cos(5*theta) + 0.06*sin(5*theta);
    X[j] += 0.05*cos(6*theta) + 0.04*sin(6*theta);
    X[j] += 0.03*cos(7*theta) + 0.02*sin(7*theta);
    X[j] += 0.01*cos(8*theta)

    // comment out as required
    X[j] += 0.001*cos(9*theta)  + 0.002*sin(9*theta);
    X[j] += 0.003*cos(10*theta) + 0.004*sin(10*theta);
    X[j] += 0.005*cos(11*theta) + 0.006*sin(11*theta);
  }

  bool inverse = false;
  FFT_top(inverse, num_bits, npts, X, F);
  inverse = true;
  FFT_top(inverse, num_bits, npts, F, Finv);

  // print output
}
```

31.2 Moving average

- We begin with functions $f(x)$ of a real variable x.
- Later we apply the formalism to periodic functions $f(\theta)$ of an angle θ,
- In many situations, a function $f(x)$ consists of a true signal $f_{\rm sig}(x)$ and also random fluctuations $f_{\rm fl}(x)$
$$f(x) = f_{\rm sig}(x) + f_{\rm fl}(x)\,. \tag{31.2.1}$$
- The signal is a smooth function and we suppose it varies relatively slowly.
- The fluctuations, by definition, average to zero.
- A common technique to cancel the fluctuations and smooth out the data (the measured values of $f(x)$) to obtain an estimate of the signal $f_{\rm sig}(x)$ is to employ a **moving average.**
  1. We select a window of width $2a$, where $a > 0$.
  2. We calculate the average of $f(x)$ over the interval $x - a$ to $x + a$:
  $$f_{\rm avg}(x;a) = \frac{1}{2a}\int_{x-a}^{x+a} f(u)\,du\,. \tag{31.2.2}$$
  3. The function $f_{\rm avg}(x;a)$ is called the **moving average** of the function $f(x)$.
  4. By definition, the value of the moving average depends on the value of a.
  5. Then instead of $f(x)$, we employ $f_{\rm avg}(x,a)$ in our data analysis, e.g. to plot graphs.
- To keep the presentation simple, suppose $f(x)$ is measured at uniformly spaced points x_j.
- Also suppose that the window parameter a corresponds to m intervals $a = m\Delta x$.
- Then replace the integral in eq. (30.2.2) by a discrete sum of $2m + 1$ points:
$$f_{\rm avg}(x_j;a) = \frac{1}{2m+1}\sum_{\ell=j-m}^{j+m} f(x_\ell)\,. \tag{31.2.3}$$
- If there are totally n values of j, the computational complexity is $(2m + 1)n$.
- There is an elegant and faster implementation, using the Fast Fourier Transform (FFT).

- Before we proceed to the FFT, let us analyze the moving average in more detail.
- The obvious question is, how do we determine the value of the window parameter a?
  1. We select the value of a so that (hopefully) the fluctuations average to zero over the interval $x-a$ to $x+a$, for all the values of x of interest to us.
$$\frac{1}{2a}\int_{x-a}^{x+a} f_{\rm fl}(u)\,du \simeq 0 \qquad \text{(for all relevant values of } x\text{)}\,. \tag{31.2.4}$$
  2. Therefore the value of a should be reasonably large.
  3. We also select the value of a so that the value of the signal $f_{\rm sig}(x)$ does not change too much over the interval $x-a$ to $x+a$, so that the value of $f_{\rm avg}(x;a)$ is a good approximation to $f_{\rm sig}(x)$.
$$\frac{1}{2a}\int_{x-a}^{x+a} f_{\rm sig}(u)\,du \simeq f_{\rm sig}(x) \qquad \text{(for all relevant values of } x\text{)}\,. \tag{31.2.5}$$
  4. Therefore the value of a should be reasonably small.
  5. These are conflicting requirements.
  6. Hence we require some context about each problem, to choose a suitable value for a.
  7. There is no unique answer, in general.

- We recognize the integral in eq. (30.2.2) as a **convolution.**
- Define the window function $f_{\text{win}}(x)$ with width $2a$ via

$$f_{\text{win}}(x) = \begin{cases} \dfrac{1}{2a} & |x| < a\,, \\ \\ 0 & |x| \ge a\,. \end{cases} \tag{31.2.6}$$

- Let us calculate the convolution of $f(x)$ with $f_{\text{win}}(x)$.
  1. The convolution integral is

$$h(x) = (f * f_{\text{win}})(x) = \int_{-\infty}^{\infty} f(u) f_{\text{win}}(x-u)\, du \tag{31.2.7}$$

  2. Then $f_{\text{win}}(x-u)$ is nonzero only in the interval $|x-u| < a$ or $x-a < u < x+a$.
  3. Hence the convolution integral simplifies to

$$h(x) = \int_{x-a}^{x+a} f(u) f_{\text{win}}(x-u)\, du \;=\; \frac{1}{2a} \int_{-\infty}^{\infty} f(u)\, du\,. \tag{31.2.8}$$

  4. This is the moving average in eq. (30.2.2).
- Next recall that the Fourier transform of a convolution of two functions $f(x)$ and $g(x)$ is the product of their Fourier transforms

$$\text{FT}[f * g] = H(k) = F(k)G(k)\,. \tag{31.2.9}$$

- Recall the Fourier transform of the window function with parameter a is

$$F_{\text{win}}(k) = \frac{\sin(ka)}{ka}\,. \tag{31.2.10}$$

- Hence the Fourier transform of the moving average $f_{\text{win}}(x; a)$ is given by

$$H(k) = F(k)\,\frac{\sin(ka)}{ka}\,. \tag{31.2.11}$$

- Hence we compute the Fourier transform $F(k)$ only once.
- We multiply by $\sin(ka)/(ka)$ and invert the Fourier transform to obtain the moving average.
- This procedure can be implemented efficiently using the Fast Fourier Transform.

- We apply the above procedure to periodic functions.
- Following the notation for the FFT, we say the input function is $X(\theta)$
- Suppose we have a periodic function $X(\theta)$ with period 2π.
- We compute the function at n points $\theta_j = 2\pi j/n$, where $j = 0, \dots, n-1$.
- We employ the FFT algorithm to compute the Fourier coefficients F_k, for $k = 0, \dots, n-1$.
- For a window function of width $2a$, the Fourier coefficients must be defined carefully because of the wraparound of the complex Fourier harmonics.
- Recall that the interval $\frac{1}{2}n < k \le n-1$ corresponds to $k < 0$.
- Hence the Fourier coefficients of the window function, say $W_k(a)$, are given by

$$W_k(a) = \begin{cases} 1 & (k = 0) \\ \dfrac{\sin(ka)}{ka} & (1 \le k \le \frac{1}{2}n) \\ \dfrac{\sin((k-n)a)}{(k-n)a} & (\frac{1}{2}n < k \le n-1)\,. \end{cases} \tag{31.2.12}$$

- For $k \ne 0$, it is simpler to define $k_{\rm wrap} = ((k + \frac{1}{2}n) \,\%\, n) - \frac{1}{2}n$. Then

$$W_k(a) = \frac{\sin(k_{\rm wrap} a)}{k_{\rm wrap} a} \qquad (k \ne 0)\,. \tag{31.2.13}$$

- Next we multiply to obtain $G_k = F_k W_k(a)$.
- Then we employ the inverse FFT algorithm using G_k to compute the moving average, say $M_j(a) = M(\theta_j, a)$, for $j = 0, \dots, n-1$.
- **Computational complexity.**
  1. ***Why not calculate the moving average directly?***
  2. Suppose $a = 2\pi m/n$. Then the moving average using $(2m+1)$ points is

$$M(\theta_j) = \frac{1}{2m+1} \sum_{\ell=j-m}^{j+m} X(\theta_\ell)\,. \tag{31.2.14}$$

  3. The two FFTs require $n \log_2(n)$ computations, plus n multiplications to compute G_k.
  4. *The computational complexity does not depend on the value of m, i.e. the width a.*
  5. The direct sums require totally $(2m+1)n$ computations.
  6. The comparison of the complexities is $2n\log_2(n)$ (FFT) vs. $2mn$ (direct sum).
  7. Hence using the FFT is faster if $\log_2(n) < m$ or $n < 2^m$. For $m = 10$ this is $n < 1024$.

- Let us calculate an example.
- For simplicity suppose one year has 360 days.
- Suppose we have a function with seasonal variations and fluctuations.
- There is an annual cycle and a monthly cycle. The signal function is

$$X_{\rm sig}(\theta) = \sin(\theta) + 0.2\,\cos(12\theta)\,. \tag{31.2.15}$$

- There are fluctuations with a daily frequency, which we model by the following

$$X_{\rm fl}(\theta) = 0.1\,N(0,1)\,\sin(360\theta)\,. \tag{31.2.16}$$

- Here $N(0,1)$ is a normally distributed random variable with zero mean and unit variance.
- The function $X(\theta)$ is plotted in Fig. 1 (a).
- The moving average is plotted as the dashed curve in Fig. 1 (b).
- The true signal function $X_{\rm sig}(\theta)$ is also displayed as the solid curve in Fig. 1 (b).
- A total of $n = 2^{10} = 1024$ sample points were employed.
- A 7 day moving average was employed, i.e. $a = 7 \times (2\pi/360)$.
- This corresponds to $19.1 \times (2\pi/1024)$, hence $m \simeq 19$ for this example.
- Then $n = 2^{10}$ and $2^m = 2^{19}$.
- Hence for this example it is faster to compute the moving average using the FFT.
- The program code is displayed below. The FFT code is given in Lecture 30.

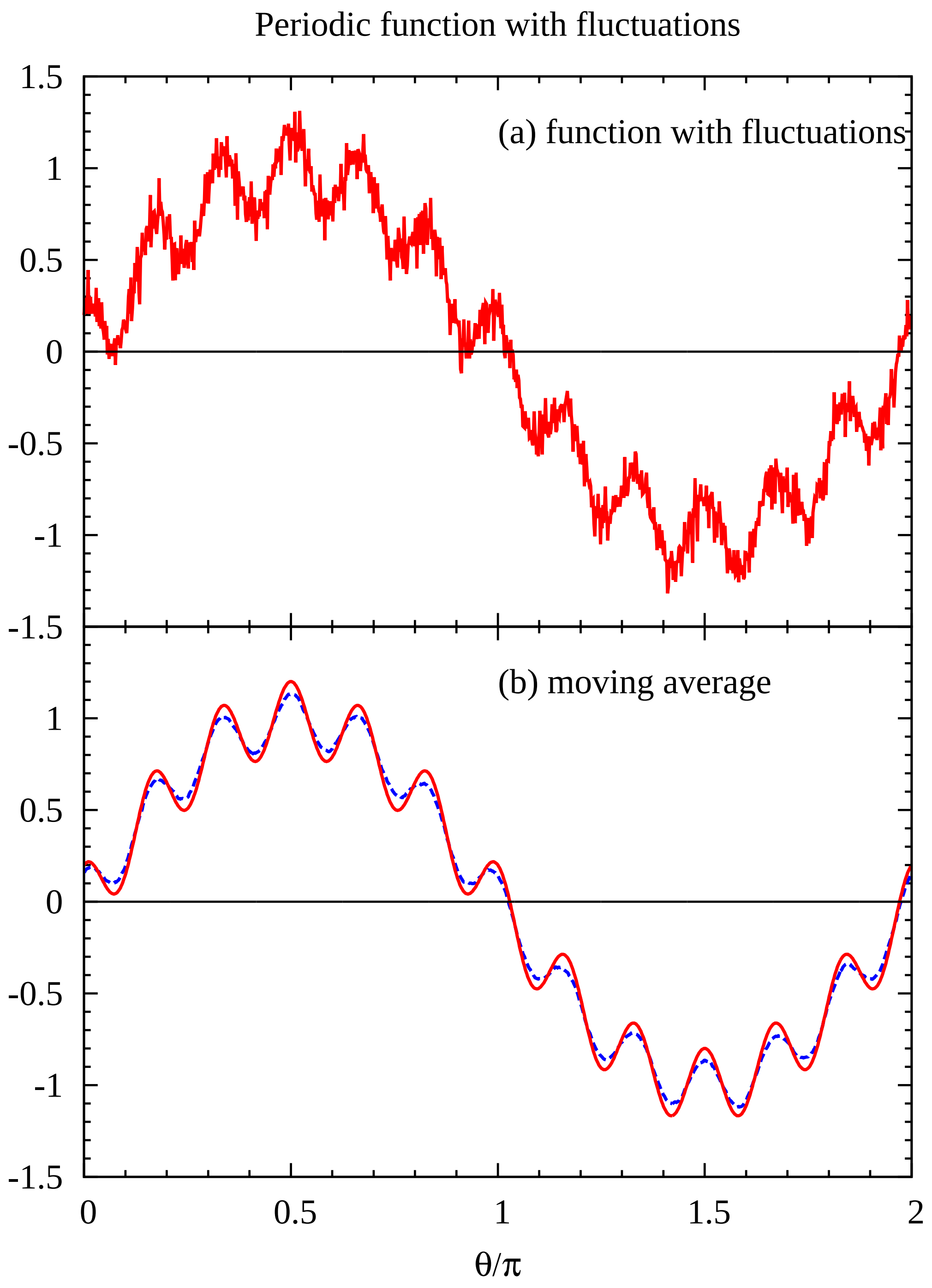

Figure 1: Plot of (a) periodic function with fluctuations and (b) moving average using FFT. The true signal function is also displayed.

```
#include <random>
#include <cmath>
#include <complex>

void FFT_mov_avg()
{
  const double pi = 4.0*atan2(1.0,1.0);

  long iseed = (use your student id, or anything else)

  std::default_random_engine generator;
  generator.seed( iseed );
  std::normal_distribution<double> n_distribution(0.0, 1.0);

  std::ofstream ofs("mov_avg.txt");

  const int num_bits = 10;
  const int npts = 1 << num_bits;
  std::complex<double> X[npts];
  std::complex<double> F[npts];
  std::complex<double> G[npts];
  std::complex<double> M[npts];

  for (int i = 0; i < npts; ++i) {
    X[i] = 0.0;
    F[i] = 0.0;
    G[i] = 0.0;
    M[i] = 0.0;
  }

  for (int j = 0; j < npts; ++j) {
    double theta = 2.0*pi*double(j)/double(npts);
    double xsig = sin(theta) + 0.2*cos(12*theta);
    double xfluc = 0.1*n_distribution(generator) *sin(360*theta);

    X[j] = xsig + xfluc;
  }

  bool inverse = false;
  FFT_top(inverse, num_bits, npts, X, F);

  // window
  double a = (2.0*pi/360.0)*7.0;
  for (int k = 0; k < npts; ++k) {
    int kwrap = ((npts/2 + k) % npts) - (npts/2);
    double sinc = (k==0) ? 1.0 : sin(a*kwrap)/(a*kwrap);
    G[k] = F[k]*sinc;
  }

  inverse = true;
  FFT_top(inverse, num_bits, npts, G, M);

  for (int j = 0; j < npts; ++j) {
    double theta = 2.0*pi*double(j)/double(npts);
    double x = X[j].real();
    double movavg = M[j].real();
    double xsig = sin(theta) + 0.2*cos(12*theta);

    ofs << std::setw(16) << theta/pi;
    ofs << std::setw(16) << x;
    ofs << std::setw(16) << movavg;
    ofs << std::setw(16) << xsig;
    ofs << std::endl;
  }
}
```

31.3 Filter

- Let us treat only a periodic function $X(\theta)$ in this section.
- Suppose the information context of a function (the 'signal' part of the function) has Fourier harmonics which extend up to some value $m_{\rm sig}$, and the amplitudes of the Fourier harmonics $F_k^{\rm sig}$ of the signal are negligible for $k > m_{\rm sig}$.
- Suppose also that the function contains noise at high frequencies, which we wish to eliminate.
- This can be accomplished using a **frequency filter.**
- In our example, we assume the unwanted part of the Fourier spectrum, i.e. the noise, is at high harmonics, and the portion of the Fourier spectrum we wish to retain consists of the low harmonics $k \le m_{\rm sig}$.
- Obviously, we simply cut out all the Fourier harmonics for $k > m_{\rm sig}$.
- *How to do this in practice?*
- Call the filtered spectrum $\hat{F}_k$. Then

$$\hat{F}_k = \begin{cases} F_k & (k \le m_{\rm sig}) \\ 0 & (k > m_{\rm sig})\,. \end{cases} \tag{31.3.1}$$

- This is called a **low pass filter.**
- Conversely, a **high pass filter** would cut all harmonics below some threshold $k < k_{\rm filter}$ and pass harmonics $k \ge k_{\rm filter}$.
  1. In our example, the low pass filter is a window function which extends from $k = 0$ to $k = m_{\rm sig}$.
  2. This is a model of an **ideal filter.**
  3. Real life frequency filters are not so abrupt in their cutoff.
  4. Real life low pass filters are approximately constant for $k < m_1$ and then fall off to zero, but not instantly, for $k > m_{\rm sig}$.

- Let us make a simple model of a function with noise and filter it.
- The function is a Lorentzian with parameter a

$$X_L(\theta) = \frac{1}{1+a^2(\theta-\pi)^2}\,. \tag{31.3.2}$$

- The Fourier transform of a Lorentzian was previous derived (for a function of x), but for our purposes (functions of θ) we can say

$$F_k = \frac{e^{-|k|/a}}{2a}\,. \tag{31.3.3}$$

- Let us use $n = 2^{10} = 1024$ sampling points.
- We also set $a = 100/\pi$.
- A graph of $X_L[j] = X_L(\theta_j) = X_L(2\pi j/n)$ and F_k is shown in Fig. 2.
- A close up view of Fig. 2 is shown in Fig. 3.
  1. The Lorentzian $X_L(\theta)$ is centered on $\theta = \pi$ so that the values of $X_{L,j}$ will be centered in the middle of the display, at $j = n/2$.
  2. Because we are using a periodic function of θ, the Fourier transform F_k is not just an exponential but has $\pm$ values.
  3. Furthermore, because of the normalization convention for the FFT, the value of F_k is actually
  $$F_k = \pm\frac{n}{2a}\,e^{-|k|/a}\,. \tag{31.3.4}$$
  4. The red curve in Fig. 3 is a graph of the 'envelope' $|F_k|$ from eq. (30.3.4).
  5. Note also that because of the wrap-around, negative values $-|k|$ appear as $n-|k|$.
- Next let us add some high-frequency noise to $X(\theta)$.
- Define a sum of high frequency harmonics with random amplitudes:

```
for (int inoise = 150; inoise <= 400; ++inoise) {
  n_rand[inoise] = N(0,1);    // pseudocode for Normal random distribution
}
```

$$S(\theta) = \sum_{k=150}^{400} \frac{n_{\rm rand}[k]}{2\sqrt{k}}\,\sin(k\theta)\,. \tag{31.3.5}$$

- Here $N(0,1)$ is a normally distributed random variable with zero mean and unit variance.
- Then define a noisy signal 'ns'

$$X_{\rm ns}(\theta) = X_L(\theta)(1+S(\theta))\,. \tag{31.3.6}$$

- Also define a low-pass filter function with a cutoff k_c via

$$F_{\rm lpf}[k] = \frac{1}{1+e^{(|k_w|-k_c)/10}}\,. \tag{31.3.7}$$

- Here k_w is the wrap-around value $k_w = ((k+\frac{1}{2}n)\,\%\,n) - \frac{1}{2}n$.
- We set the cutoff at $k_c = 120$ in this example.
- A graph of the low pass filter in eq. (30.3.7) and the ideal filter (dashed line) is shown in Fig. 4.
- We calculate the Fourier harmonics $F_{\rm ns}[k]$ of the noisy signal $X_{\rm ns}[j]$ using the FFT.
- We filter the Fourier harmonics by multiplying by the low-pass filter

$$F_{\rm filt}[k] = F_{\rm ns}[k]F_{\rm lpf}[k]\,. \tag{31.3.8}$$

- Graphs of the noisy Fourier harmonics (top panel) and the filtered Fourier harmomics (bottom panel) are shown in Fig. 5, for the harmonics $0 \le k \le n/2$.
- Note that in this model, the true signa (Lorentzian) is a cosine and the noise is a sum of sines, hence the Fourier harmonics of the signal are all in the real part of $F_{\rm ns}[k]$ (plotted in red in Fig. 5) and the Fourier harmonics of the noise are all in the imaginary part of $F_{\rm ns}[k]$ (plotted in green in Fig. 5)
- Similarly, the real and imaginary parts of the filtered harmonics $F_{\rm filt}[k]$ are plotted in blue and green, respectively, in the bottom panel of Fig. 5.
- We then apply the inverse FFT to compute the filtered function $X_{\rm filt}[j]$, which should hopefully be a cleaned up function.
- A graph of the noisy signal $X_{\rm ns}[j]$ and the reconstructed filtered signal $X_{\rm filt}[j]$ is shown in Fig. 6, for values of j close to the peak of the Lorentzian.
- We observe that the filtered signal is smooth and captures the ideal signal closely.
- The C++ code for this example is given nelow.

```
#include <random>
#include <cmath>
#include <complex>

void FFT_filter()
{
  const double pi = 4.0*atan2(1.0,1.0);

  long iseed = (use your student id, or anything else)

  std::default_random_engine generator;
  generator.seed( iseed );
  std::normal_distribution<double> n_distribution(0.0, 1.0);

  std::ofstream ofs("filter.txt");

  const int num_bits = 10;
  const int npts = 1 << num_bits;
  std::complex<double> X[npts];
  std::complex<double> F[npts];
  std::complex<double> F_filt[npts];
  std::complex<double> FS[npts];

  std::vector<double> n_rand(npts, 0.0);
  for (int ir = 0; ir < npts; ++ir) {
    n_rand[ir] = n_distribution(generator);
  }

  double a = 100.0/pi;
  for (int j = 0; j < npts; ++j) {
    double theta = 2.0*pi*double(j)/double(npts);
    double x = 1.0/(1.0 + a*a*(theta-pi)*(theta-pi));

    double sum = 0.0;
    for (int knoise = 150; knoise <= 400; ++knoise) {
      sum += n_rand[knoise] *sin(knoise*theta) *(0.5/sqrt(knoise));
    }

    double xnoise = x * (1.0 + sum);
    X[j] = xnoise;
  }

  // initialize fft to zero
  for (int k = 0; k < npts; ++k) {
    F[k] = 0;
    F_filt[k] = 0;
    FS[k] = 0;
  }

  bool inverse = false;
  FFT_top(inverse, num_bits, npts, X, F);

  // filter
  int cutoff = 120;
  for (int k = 0; k < npts; ++k) {
    int kwrap = ((k + npts/2) % npts) - npts/2;
    double low_pass_filter = 1.0/(1.0 + exp((std::abs(kwrap)-cutoff)/10.0));
    F_filt[k] = F[k] * low_pass_filter;
  }

  // reconstruct filtered signal FS
  inverse = true;
  FFT_top(inverse, num_bits, npts, F_filt, FS);

  // print output
}
```

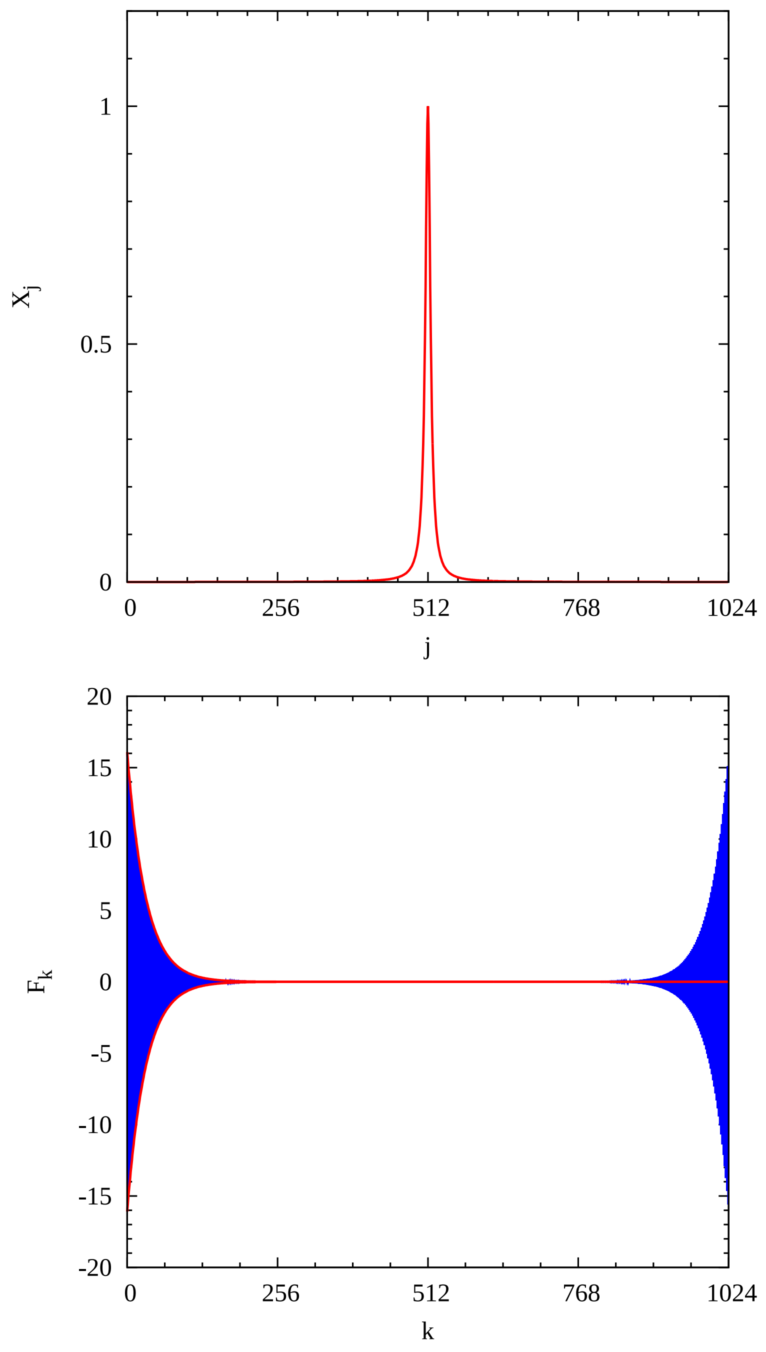

Figure 2: Graph of Lorentzian function X_j and Fast Fourier Transform F_k.

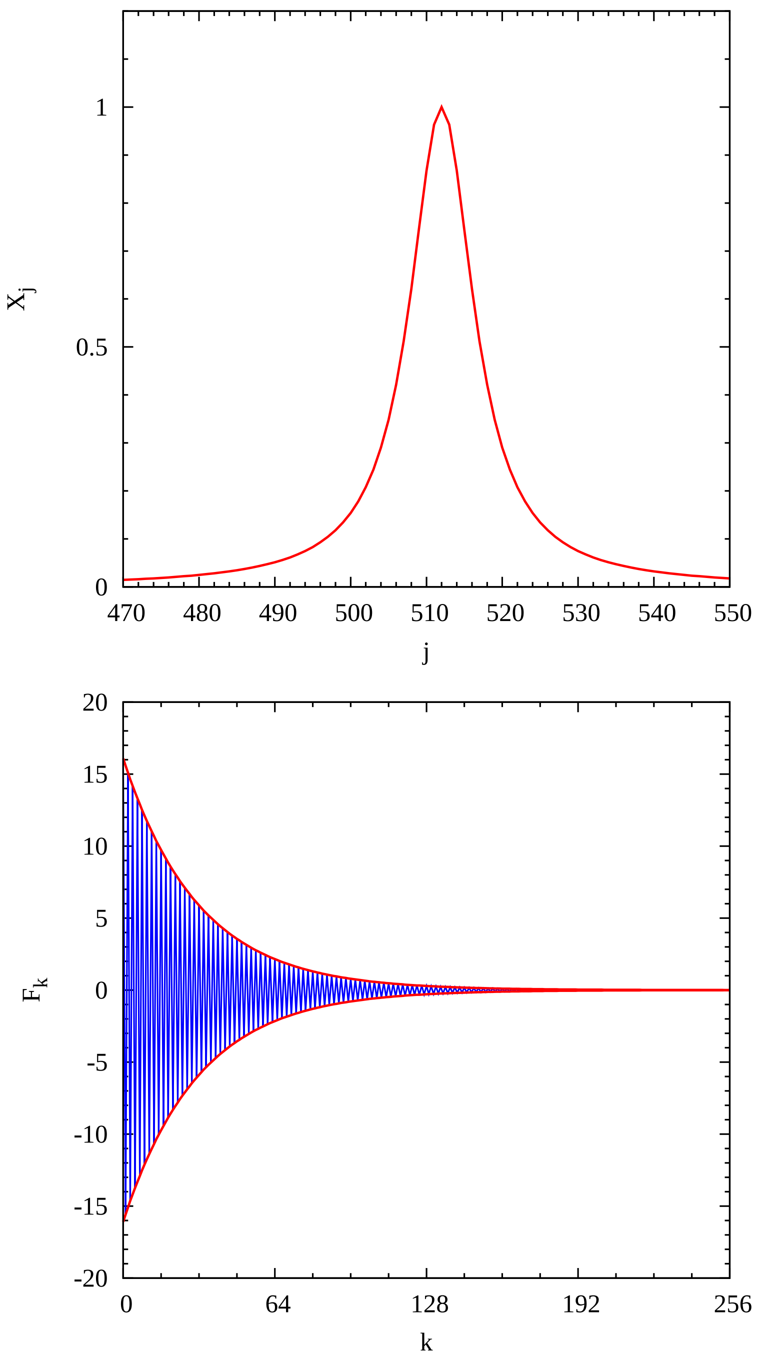

Figure 3: Close up view of Fig. 2.

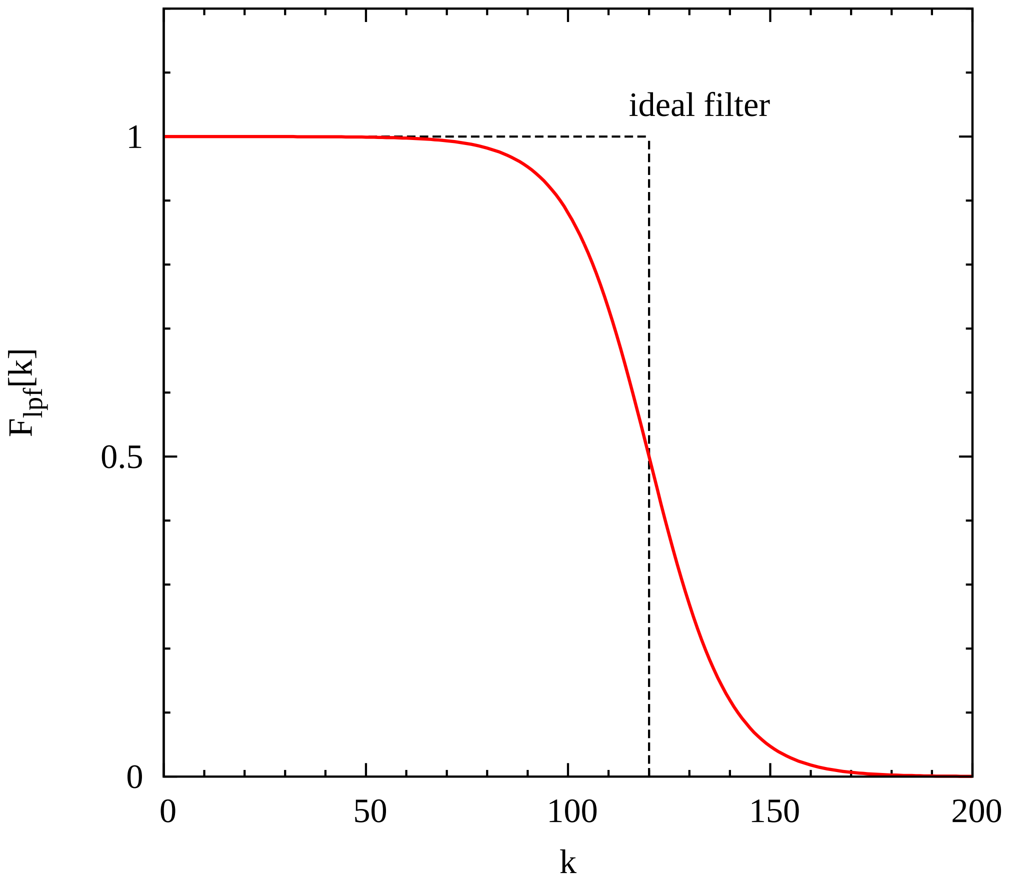

Figure 4: Graph of the frequency (Fouier harmonic) profile of an ideal and a more realistic low pass filter.

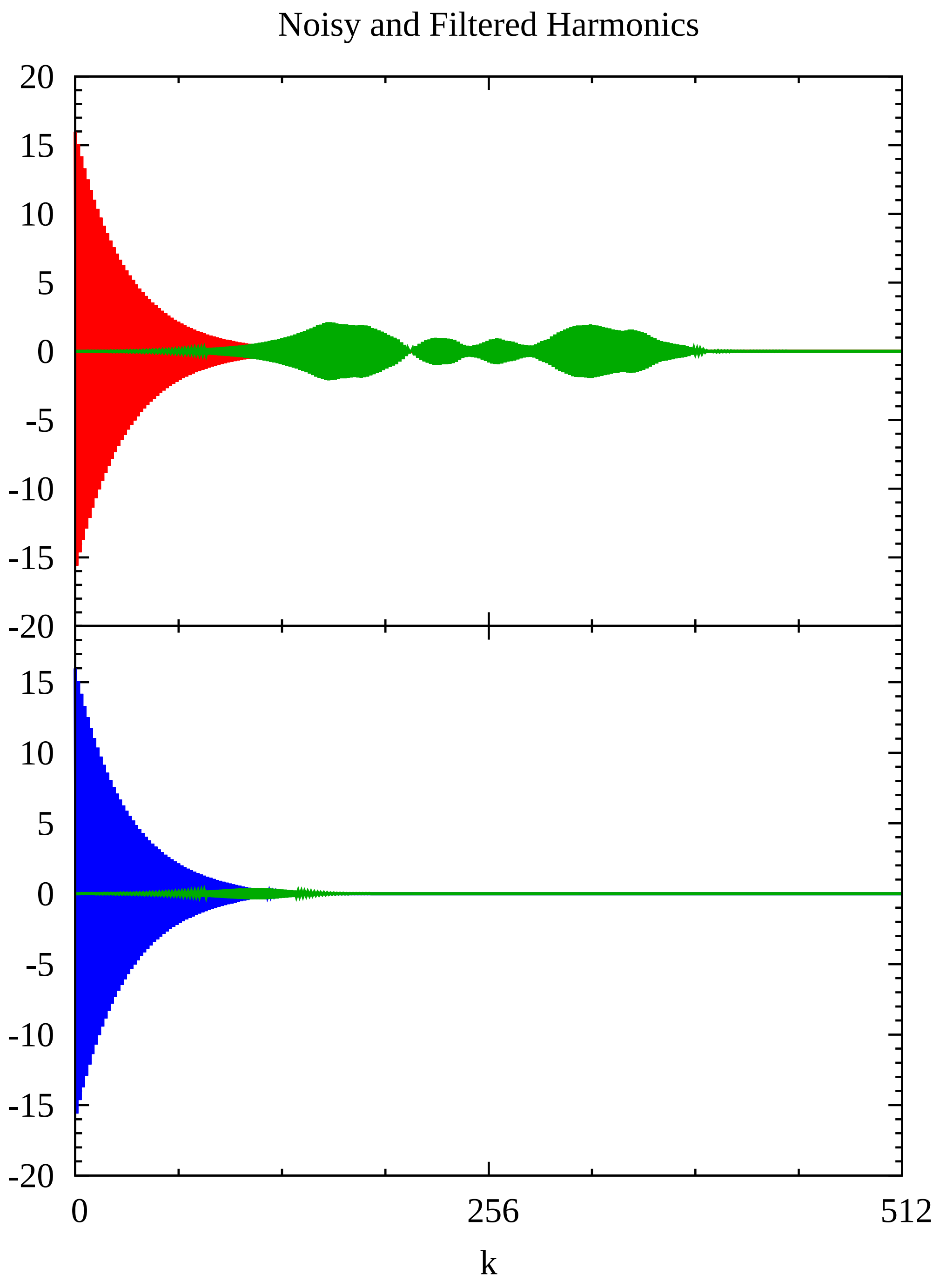

Figure 5: Graph of noisy Fourier harmonics and filtered harmonics. The real and imaginary parts of the noisy Fourier harmonics are plotted in red and green, respectively, in the top panel. The real and imaginary parts of the filtered Fourier harmonics are plotted in blue and green, respectively, in the bottom panel.

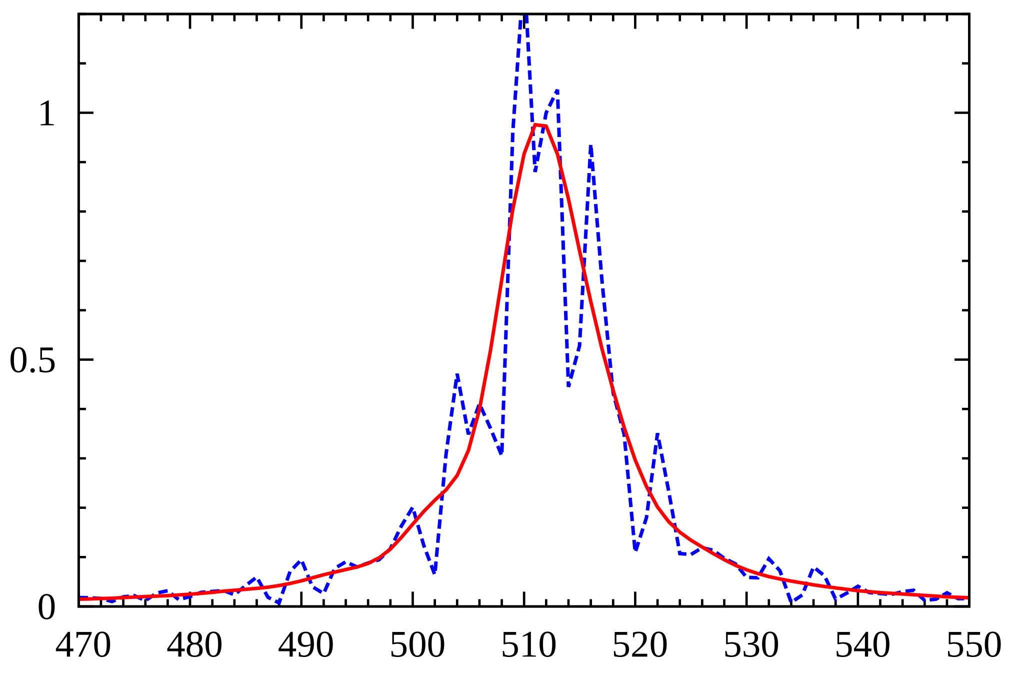

Figure 6: Graph of original function with noise (dashed) and reconstruction after filtering (solid).