Fourier analysis

• In many numerical problems methods based on Fourier transforms of a function of signal are needed:

- Solving differential equations is in many cases easier in $\mathbf{k}$ space:

1. Diffusion equation:

$$\frac{\partial c}{\partial t} = D\nabla^2 c, \quad \text{FT: } \tilde{c}(k,t) = \int_{-\infty}^{\infty} c(x,t)e^{-ikx}dx \rightarrow \frac{\partial \tilde{c}(k,t)}{\partial t} = -Dk^2\tilde{c}(k,t) \rightarrow \tilde{c}(k,t) = \tilde{c}_0 e^{-Dk^2 t}.$$

$$\text{Initial condition: } c(x,0) = \delta(x) \rightarrow \tilde{c}(k,0) = 1 \rightarrow c(x,t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{-Dk^2 t}e^{ikx}dk = \frac{1}{2\sqrt{\pi Dt}}e^{-\frac{x^2}{4Dt}}.$$

2. Elastic Green's function of a point force.

3. Electronic structure calculations.

.
.

- Signal and data processing:

1. Audio and video signal compression (perceptual coding): masking in the frequency domain.

2. Power spectrum of a signal

.
.

- In this chapter we deal with numerical Fourier transformation of data and its implementation fast Fourier transform.

Fourier analysis

• **Fourier transform** (**FT**) $H(f)$ of a function $h(t)$ is defined as

$$H(f) = \int_{-\infty}^{\infty} h(t)e^{2\pi i f t}dt.$$

- Inverse Fourier transform (**IFT**) is respectively

$$h(t) = \int_{-\infty}^{\infty} H(f)e^{-2\pi i f t}df.$$

- The above notation suggest that the original data is a function of time and that the transformation makes it a function of frequency.
- The original data may also be e.g. a function of position and the transformed data a function of the wave number (or vector).
- Sometimes instead of f the angular frequency ω is used:

$$H(\omega) = \int_{-\infty}^{\infty} h(t)e^{i\omega t}dt$$

$$h(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} H(\omega)e^{-i\omega t}d\omega$$

Fourier analysis

- Symmetry and reality properties of the original function are reflected also to its FT:

1. $h(t)$ real $\Rightarrow$ $H(-f) = [H(f)]^*$
2. $h(t)$ imaginary $\Rightarrow$ $H(-f) = -[H(f)]^*$
3. $h(t)$ even $\Rightarrow$ $H(f)$ even
4. $h(t)$ odd $\Rightarrow$ $H(f)$ odd
5. $h(t)$ real and even $\Rightarrow$ $H(f)$ real and even
6. $h(t)$ real and odd $\Rightarrow$ $H(f)$ imaginary and odd
7. $h(t)$ imaginary and even $\Rightarrow$ $H(f)$ imaginary and even
8. $h(t)$ imaginary and odd $\Rightarrow$ $H(f)$ real and odd

- These symmetries can be utilized in speeding up FT algorithms.

- FT has also a couple of handy properties:

$$h(at) \Leftrightarrow \frac{1}{|a|} H(\frac{f}{a})$$

$$\frac{1}{|b|} h(\frac{t}{b}) \Leftrightarrow H(bf)$$

$$h(t - t_0) \Leftrightarrow H(f) e^{2\pi i f t_0}$$

$$h(t) e^{-2\pi i f_0 t} \Leftrightarrow H(f - f_0) \quad .$$

Fourier analysis

- **Convolution** $g*h$ of two functions $g(t)$ and $h(t)$ is defined as

$$g*h = \int_{-\infty}^{\infty} g(\tau) h(t - \tau) d\tau \, .$$

- One can easily show that the FT of a convolution is the product of the FT's of the individual functions $G(f)$ and $H(f)$:

$$g*h \Leftrightarrow G(f) H(f) \quad .$$

- **Correlation function** is defined as

$$\mathrm{Corr}(g, h) = \int_{-\infty}^{\infty} g(\tau + t) h(\tau) d\tau \, .$$

- This essentially a convolution so that one can easily show that if g and h are real then

$$\mathrm{Corr}(g, h) \Leftrightarrow G(f) H^*(f)$$

- Correlation of the function with itself is so called **autocorrelation function**

$$\mathrm{Corr}(g, g) = \int_{-\infty}^{\infty} g(\tau + t) g(\tau) d\tau \, , \qquad \mathrm{Corr}(g, g) \Leftrightarrow |G(f)|^2 \, .$$

Fourier analysis

- According to **Parseval's theorem** the total power of a signal is the same in the time and frequency domains:

$$P_{\text{tot}} \equiv \int_{-\infty}^{\infty} |h(t)|^2 dt = \int_{-\infty}^{\infty} |H(f)|^2 df \ .$$

- The power at the frequency interval $[f, f+df]$ is defined as (in this case positive and negative frequencies are often treated with equal footing)

$$P_h(f) \equiv |H(f)|^2 + |H(-f)|^2 \ , \qquad 0 \le f < \infty \ .$$

- When the original signal is real this becomes

$$P_h(f) \equiv 2|H(f)|^2 \ .$$

Fourier analysis: discrete FT

• In practice the data is not continuous but is a set of discrete points h_n placed on the time axis:

$$h_n = h(\Delta n), \quad n = \ldots, -3, -2, -1, 0, 1, 2, 3, \ldots .$$

- The inverse of the time step Δ is called the **sampling rate**.

- The **Nyquist critical frequency** is defined as

$$f_c \equiv \frac{1}{2\Delta} \ .$$

- Assume we take samples from function $h(t)$ with frequency $1/\Delta$.

- If the **bandwidth** of the function is restricted in such a way that for all frequencies $|f| \ge f_c \quad H(f) = 0$, then the samples h_n determine the signal completely.

- We can now express the function as

$$h(t) = \Delta \sum_{n=-\infty}^{\infty} h_n \frac{\sin[2\pi f_c(t-\Delta n)]}{\pi(t-\Delta n)} .$$

- We often do know that our data is bandwidth-limited.
- In these cases it is enough to sample the data at a frequency which is twice the maximum frequency in the data.

Fourier analysis: discrete FT

- If we sample the data with a too low frequency we observe that all frequencies outside the range $[-f_c, f_c]$ is folded to this range. This phenomenon is called **aliasing**.

- For example signals $\exp(2\pi i f_1 t)$ and $\exp(2\pi i f_2 t)$ give the same samples at an interval Δ when the difference in the frequencies $|f_1 - f_2|$ is a multiple of $1/\Delta$.

- In order to prevent aliasing one should make sure that the signal does not contain too large frequencies (e.g. by a low-pass filter).

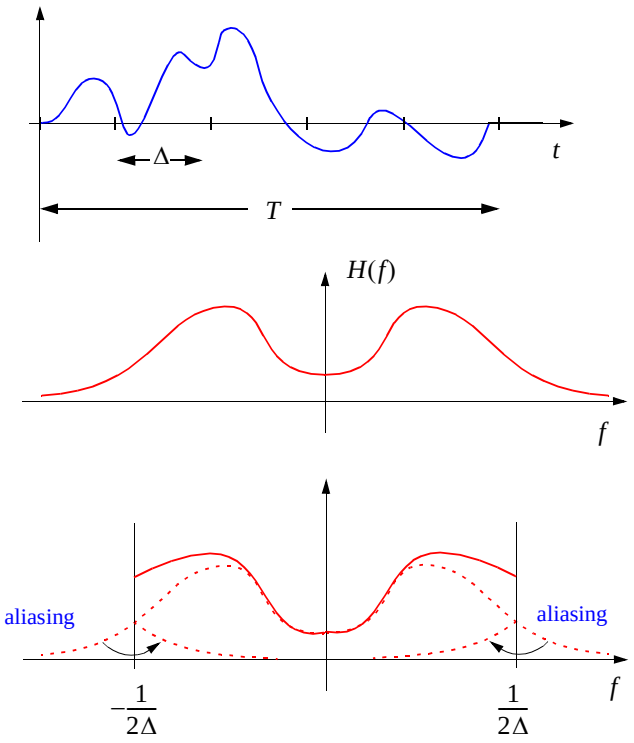
- It is easy check whether there is aliasing: if the FT goes to zero when approaching f_c then the signal does not contain too hugh frequencies.

- With the **discrete Fourier transformation** (**DFT**[1]) one can approximate FT of a continuous function based on samples taken from it.

- Assume we have N samples:

$$h_k \equiv h(t_k), \quad t_k \equiv k\Delta, \quad k = 0, 1, 2, \ldots, N-1.$$

- Moreover, let N be even.



1. Not to be mixed with another DFT, namely the density functional theory ;-)

Fourier analysis: discrete FT

- We only have N input data we only can obtain N output data.
- Because of this we compute DFT only for points

$$f_n \equiv \frac{n}{N\Delta}, \quad n = -\frac{N}{2}, \ldots, \frac{N}{2}.$$

- The limits $f_{-N/2}$ and $f_{N/2}$ correspond to the Nyquist critical frequency $f_c = 1/2\Delta$.

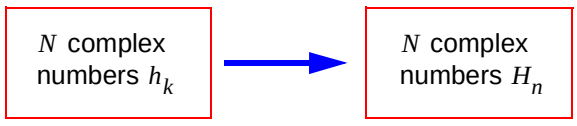
- DFT is computed by approximating the integral by a sum:

$$H(f_n) = \int_{-\infty}^{\infty} h(t) e^{2\pi i f_n t} dt \approx \sum_{k=0}^{N-1} h_k e^{2\pi i f_n t_k} \Delta = \Delta \sum_{k=0}^{N-1} h_k e^{2\pi i k n / N}.$$

- The last sum in the above expression is called the DFT. Let's denote it as

$$H_n \equiv \sum_{k=0}^{N-1} h_k e^{2\pi i k n / N}.$$

- So the DFT can be depicted as a following transformation



Fourier analysis: discrete FT

- Note that H_n does not depend on any dimensional quantity (like Δ).

- The connection between the DFT and FT is

$$H(f_n) \approx \Delta H_n \ ,$$

where the frequency is computed as $f_n \equiv n/N\Delta$.

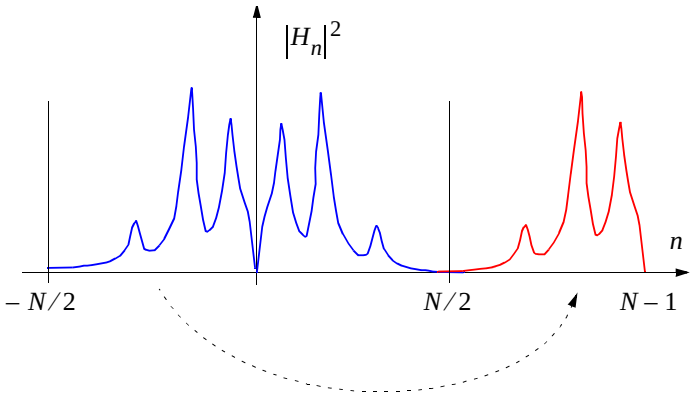
- According to the definition of f_n we assumed that the index n is in the interval $\left[-\frac{N}{2}, \frac{N}{2}\right]$
- Now, when we look at the definition of H_n we see that it is periodic with a period of N.
- This means that

$$H_{-n} = H_{N-n} \quad \text{for} \quad n = 1, 2, \ldots$$

- Because of this the index n normally takes the values $n = 0, 1, 2, \ldots, N-1$ in the definition of the frequency: $f_n \equiv n/N\Delta$.
- In this case the positive frequencies are located at $n = 1, 2, \ldots, N/2 - 1$ and the negative frequencies at $n = N/2 + 1, \ldots, N-1$.
- Value $n = 0$ is the 'DC component'. $n = N/2$ corresponds to both $-f_c$ and f_c .

Fourier analysis: discrete FT

- If this sounds complicated the figure below should clarify the issue.



- The same symmetry properties as for the continuous FT apply also to DFT.
  - For example changing the sign of the argument corresponds to a shift by N: $-f \Leftrightarrow N - n$.

- The discrete inverse FT is defined as

$$h_k = \frac{1}{N} \sum_{n=0}^{N-1} H_n e^{-2\pi i k n/N}.$$

Fourier analysis: discrete FT

- Discrete version of the Parseval's theorem is

$$\sum_{k=0}^{N-1} |h_k|^2 = \frac{1}{N} \sum_{n=0}^{N-1} |H_n|^2 .$$

Fourier analysis: FFT

• The computation time of a DFT behaves as $O(N^2)$.

- This can justified as follows:

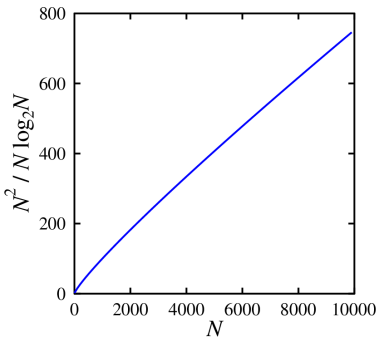
  Define a complex number $W \equiv e^{2\pi i/N}$.
  Definition of DFT can now be written in the form

$$H_n = \sum_{k=0}^{N-1} h_k e^{2\pi i k n/N} = \sum_{k=0}^{N-1} h_k W^{nk} .$$

  In other words a N-vector h_k is multiplied by a $N \times N$ matrix,
  the elements of which are powers of a complex number W.
  Matrix multiplication scales as $O(N^2)$.

- In the mid-60's John Tukey and John Cooley published an article describing an algorithm for DFT that scales as $O(N \log_2 N)$.

- The algorithm is called **fast Fourier transform** (**FFT**)[1].



Fourier analysis: FFT

• A sidenote: Remember the list of top ten algorithms of the 20th century
(http://www.siam.org/siamnews/05-00/current.htm)

1. Metropolis Monte Carlo. von Neumann et al., 1946
2. Simplex linear programming method (optimization). Dantzig, 1947
3. Krylov subspace iteration method (solving $\mathbf{Ax} = \mathbf{b}$ for large matrices). Hestenes et al., 1950
4. Decompositional approach to matrix computations. Householder, 1951
5. Fortran optimizing compiler. Backus, 1957
6. QR algorithm for eigenvalue problems. Francis, 1959-61
7. Quicksort. Hoare, 1962
8. **Fast Fourier transform. Cooley and Tukey, 1965**
9. Integer relation detection algorithm. Ferguson and Forcade, 1977
10. Fast multipole algorithm. Greengard and Rokhlin, 1987

1. For a historical background of FFT see D. Kahaner, C. Moler, S. Nash: *Numerical Methods and Software*, Prentice-Hall, London, 1989, sec. 11.14.

Fourier analysis: FFT

• FFT is based on the fact that the DFT of N samples f_i can be expressed in terms of two DFTs of data sets with $N/2$ samples.

- One of the DTFs consists of samples with even indices and the other of samples with odd indices[1]:

$$F_k = \sum_{j=0}^{N-1} f_j e^{2\pi i j k/N}$$

$$= \sum_{j=0}^{N/2-1} f_{2j} e^{2\pi i (2j) k/N} + \sum_{j=0}^{N/2-1} f_{2j+1} e^{2\pi i (2j+1) k/N}$$

$$= \sum_{j=0}^{N/2-1} f_{2j} e^{2\pi i j k/(N/2)} + W^k \sum_{j=0}^{N/2-1} f_{2j+1} e^{2\pi i j k/(N/2)}$$

$$= F_k^{\mathrm{e}} + W^k F_k^{\mathrm{o}}$$

- Here the constant W is as above, i.e. $W = e^{2\pi i/N}$.
- F_k^{e} is the k^{th} element of the DFT computed from the **even** samples of the original data.
- F_k^{o} is correspondingly computed from the **odd** samples.

1. Note that now f_j is not a frequency but the sample j of the data set.

Fourier analysis: FFT

- The idea of FFT is to apply this kind of division recursively:

  Divide F_k^{e} into two parts F_k^{ee} and F_k^{eo} each of which is a DFT of data sets of length $N/4$.

- Assuming that N is a power of two ($N = 2^m$) we can continue the division process until we have a DFT of length one, which is nothing more than an identity operation.

- So, for all different combinations of symbols o and $\mathbf{e}$ with length $\log_2 N$ we can find a DFT of length one:

$$F_k^{\mathrm{eooeoeoeeeeoo...oeoee}} = f_n \,.$$

- The problem is now to find out the value of n that corresponds to a particular string ‘eooeoeoeeeeoo...oeoee‘.

  - The first division is based on the least significant bit of n:

    If it is 0 then n is even and the element belongs to term F_k^{e},

    otherwise to term F_k^{o}.

  - In the next division we examine the second least significant bit and so on.
  - The index n of the element is obtained by the reversing the string ‘eooeoeoeeeeoo...oeoee‘ and interpreting it as a binary number such that

$$\begin{cases} \mathrm{e} \to 0 \\ o \to 1 \end{cases}.$$

Fourier analysis: FFT

- As a figure for $N = 16$:

N	$N/2$	$N/4$	$N/8$	$N/16$
0	0	0	0	0000
1	2	4	8	1000
2	4	8	4	0100
3	6	12	12	1100
4	8	2	2	0010
5	10	6	10	1010
6	12	10	6	0110
7	14	14	14	1110
8	1	1	1	0001
9	3	5	9	1001
10	5	9	5	0101
11	7	13	13	1101
12	9	3	3	0011
13	11	7	11	1011
14	13	11	7	0111
15	15	15	15	1111

Fourier analysis: FFT

- We see that in the last column the data is in the **bit reversed order** and it consists of one-element DFTs.

- The FT of length N is then obtained by successively combining adjacent elements in this bit reversed array by applying the formula

$$F_k = F_k^{\mathrm{e}} + W^k F_k^{\mathrm{o}}$$

- So, the FFT algorithm consists of two steps:

    1. Sort the data table in bit reversed order.

    2. Recursively combine adjacent elements in this array to obtain the final FT.

- Combination of the elements takes time as $O(N)$, and this is done $O(\log_2 N)$ times so that the algorithm behaves as $O(N\log_2 N)$. (Assuming that the initial sorting of the array does not take more time than $O(N)$.)

- The algorithm can further speeded up by restricting the calls to trigonometric functions to outer loops.

Fourier analysis: FFT

• There is no lack of FFT implementations

MATLAB: **`FFT(x)`**, also 2D FFT: **`FFT2(x)`**

GSL: **`gsl_fft_complex_radix2_forward`**,
`gsl_fft_complex_radix2_backward`,
`gsl_fft_real_radix2_transform`,
. . .

SLATEC: **`RFFTF`** (only single precision)
. . .

FFTW: A C library for FFT (http://www.fftw.org/)

FFTPACK: A Fortran library for FFT (http://www.netlib.org/fftpack/)

- When using a particular implementation of FFT be careful to store your data in the right format.
    - Especially in C because there is no complex data type in the language.

Fourier analysis: FFT

- Example of using GSL FFT routine **`gsl_fft_complex_radix2_forward`**:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <gsl/gsl_errno.h>
#include <gsl/gsl_fft_complex.h>

#define DR(z,i) ((z)[2*(i)])   /* Real part  */
#define DI(z,i) ((z)[2*(i)+1]) /* Imag. part */
double gasdev(int *);

int main (int argc, char **argv)
{
  int i,np,np2;
  double per1,per2,std,*data,sr,si;
  int seed;

  np=atoi(*++argv);
  per1=atof(*++argv);
  per2=atof(*++argv);
  std=atof(*++argv);
  seed=atol(*++argv);
  np2=2*np;
  data=(double *)
     malloc((size_t)((2*np)*sizeof(double)));

  for (i=0;i<np;i++) {
    DR(data,i)=cos(2.0*M_PI*(i-1)/per1)+
               cos(2.0*M_PI*(i-1)/per2)+
               std*gasdev(&seed);
    DI(data,i)=0.0;
  }

  for (i=0;i<np;i++) {
    printf ("ORIG: %d %e %e\n",
             i,DR(data,i),DI(data,i));
  }
  printf ("\n");

  gsl_fft_complex_radix2_forward(data,1,np);

  for (i=0;i<np;i++) {
    sr=DR(data,i)*DR(data,i)/np;
    si=DI(data,i)*DI(data,i)/np;
    printf ("FFT: %d %g \n",i,sr+si);
  }

  return 0;
}


/*

Program uses the routine gasdev to generate
Gaussian random numbers.

Only the absolute value of the FT is printed.
That's for plotting purposes.

*/
```


Fourier analysis: FFT

- The program transforms the function

$$\cos\left(\frac{2\pi t}{P_1}\right) + \cos\left(\frac{2\pi t}{P_2}\right) + \sigma_n r,$$

where P_1 and P_2 are the periods of the signal and r is a random number with Gaussian distribution ($\langle r \rangle = 0$, $\sigma_r = 1$).

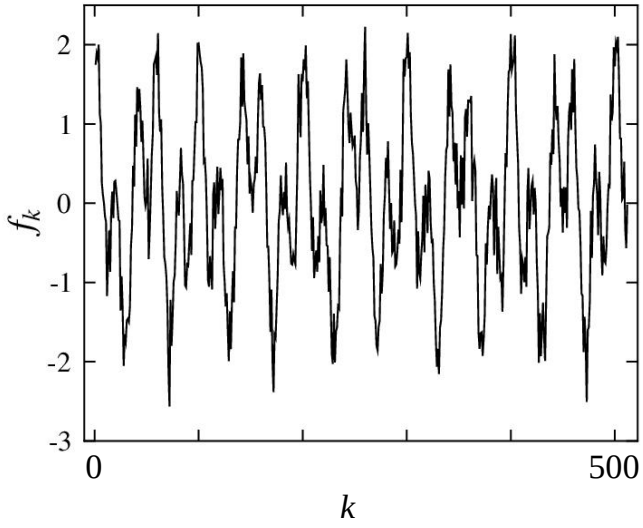
- Compilation and run:

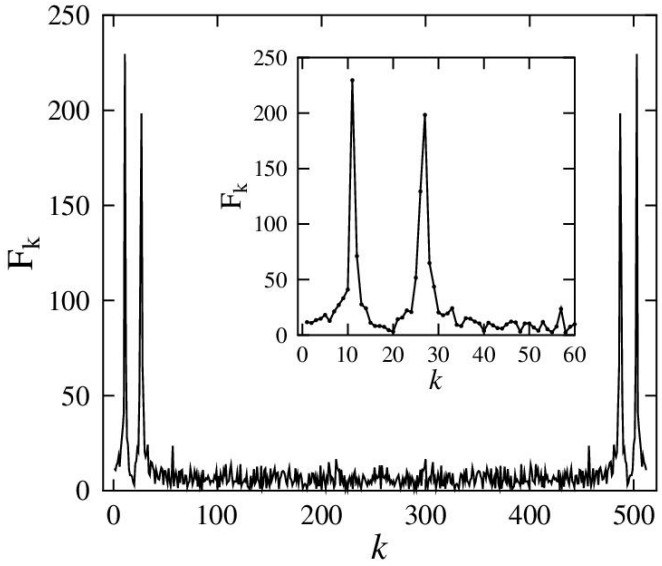
```
gsl> gcc -o gsl_fft gsl_fft.c -lgsl -lgslcblas -lm
gsl> ./gsl_fft 16 3 5 0.3 34535
ORIG: 0 -4.420391e-01 0.000000e+00
ORIG: 1 1.948316e+00 0.000000e+00
ORIG: 2 -1.348478e-01 0.000000e+00
ORIG: 3 -8.243850e-01 0.000000e+00
ORIG: 4 1.379508e-01 0.000000e+00
ORIG: 5 4.960397e-03 0.000000e+00
ORIG: 6 3.360909e-01 0.000000e+00
ORIG: 7 1.367261e+00 0.000000e+00
ORIG: 8 -1.031304e+00 0.000000e+00
ORIG: 9 -9.477208e-01 0.000000e+00
ORIG: 10 1.768183e+00 0.000000e+00
ORIG: 11 9.333204e-02 0.000000e+00
ORIG: 12 -1.755363e-01 0.000000e+00
ORIG: 13 4.970365e-01 0.000000e+00
ORIG: 14 -1.676865e+00 0.000000e+00
ORIG: 15 2.156618e-02 0.000000e+00

FFT: 0 0.0554601
FFT: 1 0.0834737
FFT: 2 0.219583
FFT: 3 2.29423
FFT: 4 0.247893
FFT: 5 1.72874
FFT: 6 1.73657
FFT: 7 0.743871
FFT: 8 0.71349
FFT: 9 0.743871
FFT: 10 1.73657
FFT: 11 1.72874
FFT: 12 0.247893
FFT: 13 2.29423
FFT: 14 0.219583
FFT: 15 0.0834737
```

Fourier analysis: FFT

- The figures below were computed with

$$P_1 = 20,\ P_2 = 50,\ \sigma_n = 0.3,\ N = 512.$$





Fourier analysis: FFT

- Aliasing can be illustrated by using the same function and by varying the period P_1.

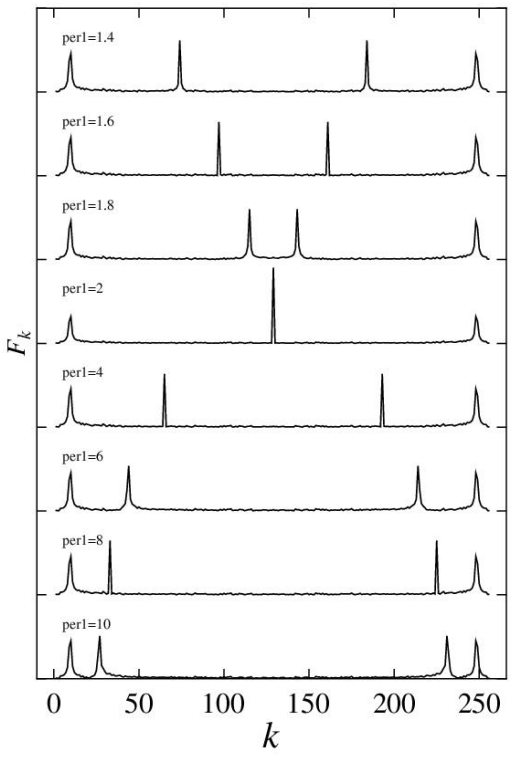
  - In the figure on the right the period is varied as $P_1 = 1.4 \ldots 10.0$.
  - Note that in this case, because $\Delta = 1$ the Nyquist critical frequency is

$$f_c = \frac{1}{2}.$$

  - Other parameters are

$$P_2 = 30$$
$$\sigma_n = 0.1$$
$$N = 256.$$

  - We see that when $P_1 < 2$ the peak of the corresponding signal appears in a wrong place.

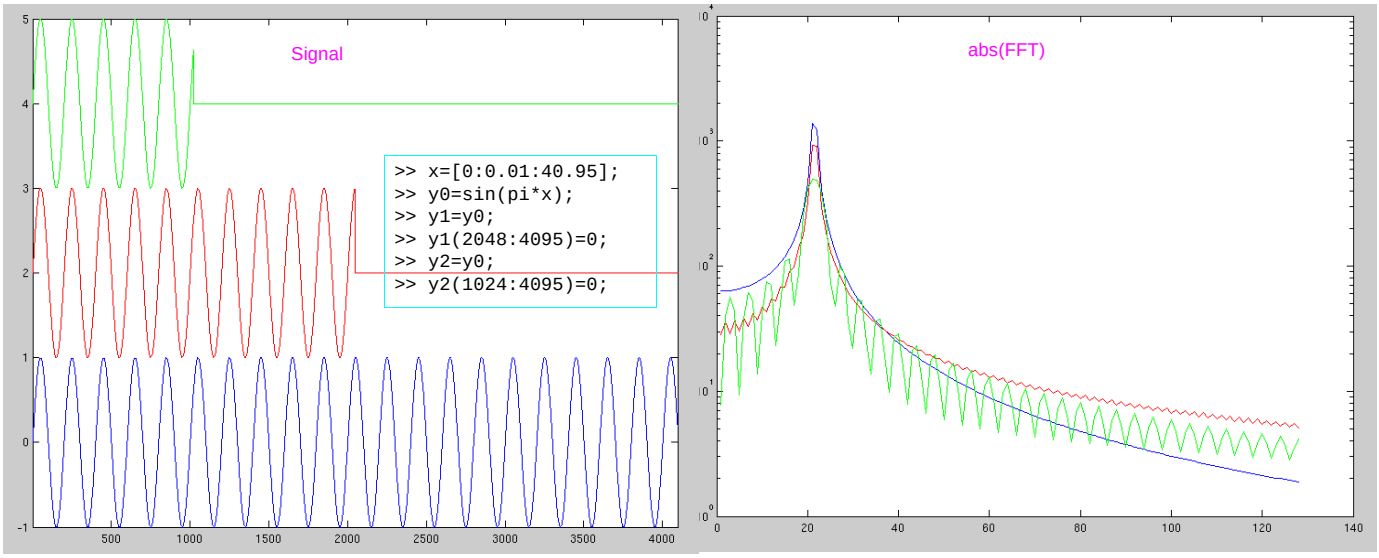


• FT can easily be generalized to more than one dimension:

$$H(n_1 \ldots, n_L) = \sum_{k_L = 0}^{N_L - 1} \cdots \sum_{k_1 = 0}^{N_1 - 1} \exp\left(-\frac{2\pi i k_L n_L}{N_L}\right) \ldots \exp\left(-\frac{2\pi i k_1 n_1}{N_1}\right).$$

Fourier analysis: FFT

• Note that above we assumed that the size of the data set is a power of two.
- If this not the case we can add zeros to our data to increase the size up to the nearest power (zero padding).
- Figures below demonstrate how this affects the power spectrum of a signal.



- Other that radix-2 FFT algorithms are used when the data size is something else than a power of two.
- For example the GSL library allow one to ise so called mixed radix algorithms where the data decomposition is based on the factorization of the size of the data set in terms of 2, 3, 4, 5, 6 and 7. The fraction of data that can not be factored is computed by the $O(N^2)$ DFT.

Fourier analysis: windowed FT

• In Fourier analysis in principle only the frequency information is obtained.

- Time information can be obtained by using **windowed Fourier transform** (**WFT**).

$$H(f,t) = \int_{-\infty}^{\infty} h(u)g(u-t)e^{2\pi i f u}du \ ,$$

where $g(u-t)$ is so called window function.

- For example let's transform the so called chirp signal

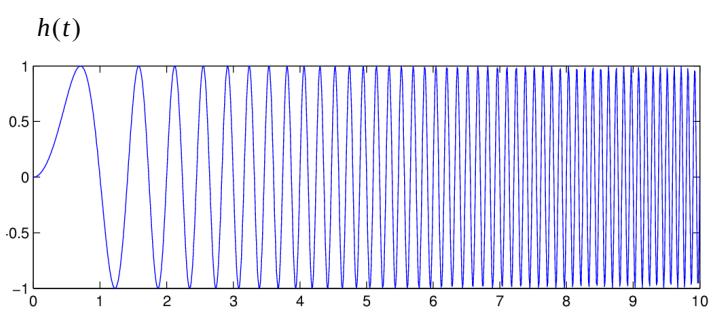
$$h(t) = \sin(\pi t^2).$$

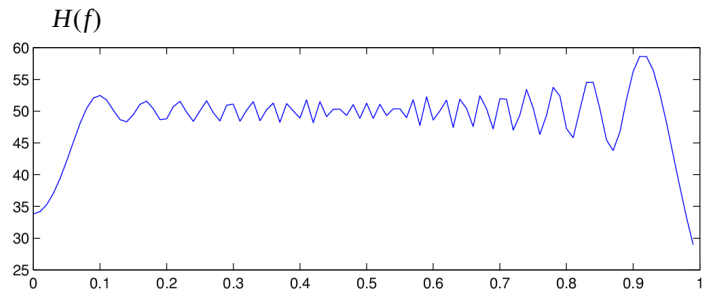
- As a windows function we take

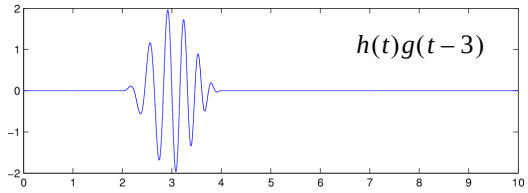
$$g(u) = \begin{cases} 1+\cos(\pi u), & -1 \le u \le 1 \\ 0, & \text{otherwise} \end{cases}$$

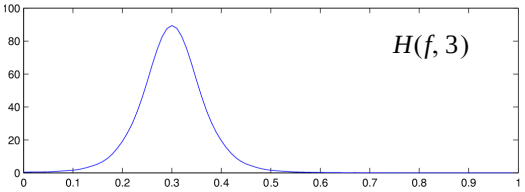
Fourier analysis: windowed FT

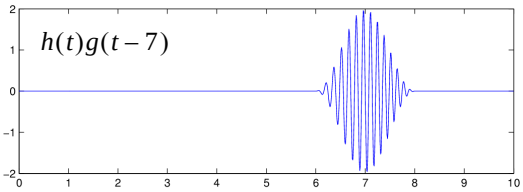
- Below the signal and its WFT (at different times t)

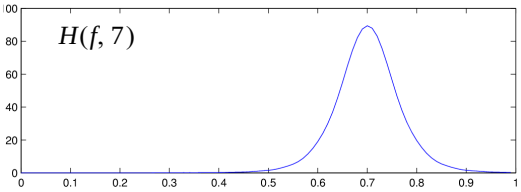












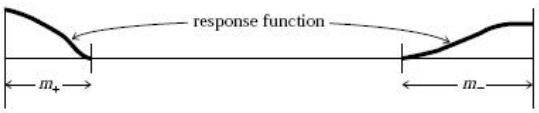
- In the last exercise you will become acquainted with different windows functions and their effect on the power spectrum.

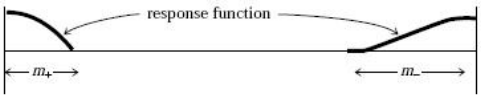
Fourier analysis: spectral applications of FT

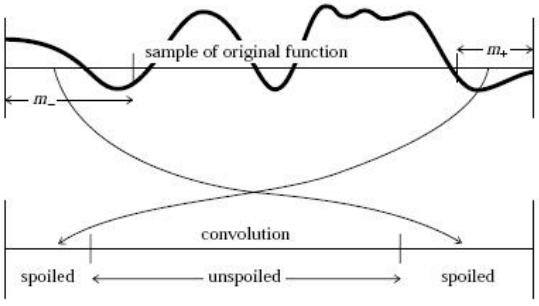
• As shown in the beginning of the chapter convolution and correlation functions can be easily computed by using FT.

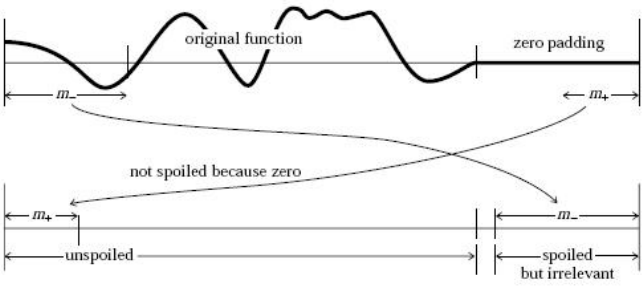
- Convolution: $\sum_{k=-N/2+1}^{N/2} s_{j-k} r_k \Leftrightarrow S_n R_n$.

  - When convoluting nonperiodic data one has to keep in mind the wrap-around problem which can be cured by **zero-padding**:[1]





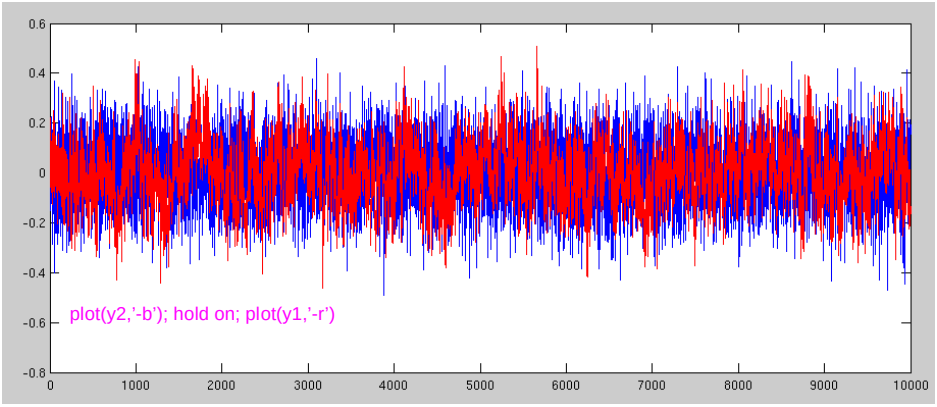


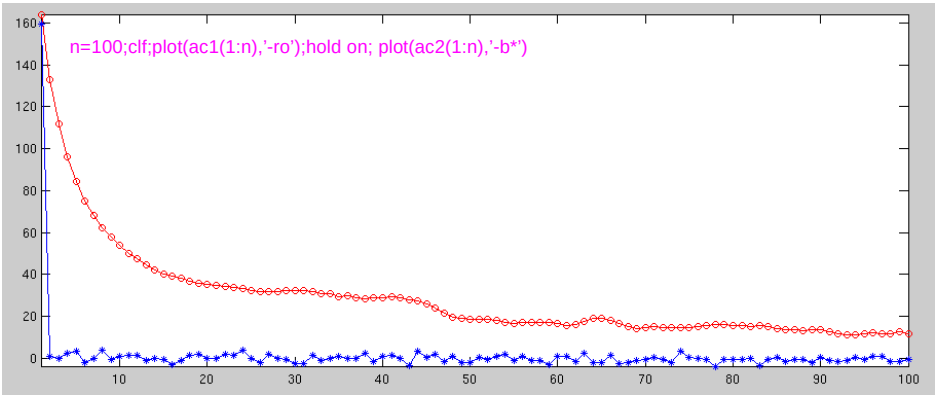


1. Numerical Recipes, Figures 13.1.3 and 13.1.4.

Fourier analysis: spectral applications of FT

- Correlation: $\mathrm{Corr}(g,h)_j \Leftrightarrow G_k H_k^*$ and autocorrelation $\mathrm{Corr}(g,g)_j \Leftrightarrow G_k G_k^* = |G|^2$
- Example: Exercise 10, problem 1:





Matlab:

```
>> ac1=ifft((abs(fft(y1))).^2);
>> ac2=ifft((abs(fft(y2))).^2);
```


Fourier analysis: spectral applications of FT

• Digital filters are often used to smoothing or filtering data.

- A **general linear filter** transforms the input signal x_t to output y_t:

$$y_t = \sum_{s=1}^{p} a_s y_{t-s} + x_t - \sum_{s=1}^{q} b_s x_{t-s}.$$

- The coefficients a_s and b_s determine the filter behavior.
- If all a_s are zero then the filter is called nonrecursive or finite impulse response (FIR) filter. Otherwise it is called recursive or infinite impulse response (IIR) filter.
- Impulse response is the output of the filter when the input is a delta peak: $x_t = \delta_{t,t_0}$.
- Impulse response of a FIR is

$$y_t = \begin{cases} \delta_{t,t_0} - b_{t-t_0} & 1 \le t - t_0 \le q \\ \delta_{t,t_0} & \text{otherwise} \end{cases}$$

and as the name says: finite.

- For IIR the response is so straightforward to calculate. Let's take an example:

$$y_t = a y_{t-1} + x_t.$$

  - This gives the following filtered data

$$y_1 = x_1, \quad y_2 = a y_1 + x_2, \ \ldots$$

  - Solving this recursively gives

$$y_t = x_t + a x_{t-1} + a^2 x_{t-2} + \ldots + a^{t-1} x_1$$

showing an exponential decay.

Fourier analysis: spectral applications of FT

- The frequency response of the filter is defined as[1]

$$Y(f) = \frac{|B(f)|^2}{|A(f)|^2} X(f) \quad ,$$

where $Y(f)$ and $X(f)$ are the Fourier transforms of the output y_t and input x_t, respectively, and

$$A(f) = 1 - \sum_{s=1}^{p} a_s e^{-2\pi i f s},$$

$$B(f) = 1 - \sum_{s=1}^{q} b_s e^{-2\pi i f s}.$$

- This means that if we know the coefficients a_s, b_s we can immediately calculate the frequency response of the filter.
- In practice it is the other way around: we know the desired response and want to build a corresponding filter.

- For nonrecursive filters the frequency response is $B(f)$, from which we can by inverse FT get the coefficients.
  - This is roughly done such that initially you take a rather large number of coefficients (large q) and compute their values from the known $B(f)$. Then you zero all except the first and last K coefficients (and cyclically shift the last coefficients to the beginning of the array to get rid of negative lags). If the response computed from this set is not satisfactory then you can e.g. increase the number of non-zero coefficients.

  - This is only one way to design filters and recursive filters are more complicated to handle. There is a vast literature dealing with the subject.

1. R.H.Shumway, *Applied Statistical Time Series Analysis*, Prentice-Hall, 1988; Section 2.9.

Fourier analysis: spectral applications of FT

- Note that a simple rolling average is also a linear filter:

$$y_t = \frac{1}{N} \sum_{s=0}^{N-1} x_{t-s}.$$

and its frequency response function can be calculated as

$$|B(f)|^2 = \frac{1}{N^2} \left| \sum_{s=0}^{N-1} e^{-2\pi i f s} \right|^2 = \frac{1}{N^2} \left[\left| \sum_{s=0}^{N-1} \cos(2\pi f s) \right|^2 + \left| \sum_{s=0}^{N-1} \sin(2\pi f s) \right|^2 \right].$$

- A simple example: $N = 2$

$$y_t = \frac{1}{2} x_t + \frac{1}{2} x_{t-1}$$

$$|B(f)|^2 = \frac{1}{4}[(1 + e^{-2\pi i f})(1 + e^{2\pi i f})] = \frac{1}{4}[2 + e^{2\pi i f} + e^{-2\pi i f}] = \frac{1}{2}[1 + \cos(2\pi f)] \quad .$$

- **Now a demo**.

Wavelet transforms

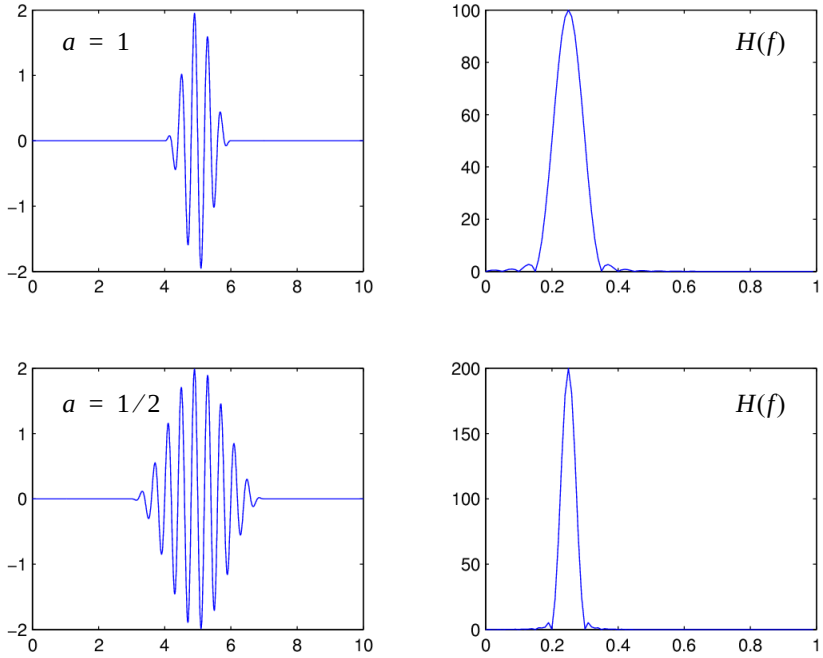
• In Fourier transform one can not at the same time get an accurate picture of the original signal both on the frequency and time axis.

- Uncertainty principle: In order to determine the frequency accurately one has to take a long (in time) sample.

Signal $h(t) = \sin(5\pi t)$.

Window

$$g(u) = \begin{cases} 1 + \cos(a\pi u), & -\frac{1}{a} \le u \le \frac{1}{a} \\ 0, & \text{otherwise} \end{cases}$$



Wavelet transforms

• In order to get accurate description of data in the time axis (locality) we need many FT basis functions (because the are not localized at all).

- Using the windowed FT we can get some localization but the scale is set to the width of the window T:

  Features with time scales shorter than T are produced in frequency domain.
  Features with time scales longer than T are produced in time domain.

- In **Wavelet transformation** (**WF**) the basis functions (wavelets) are derived from so called mother wavelet $\psi(u)$

  - This function is scaled and translated so that we get basis functions that 'probe' the original signal at different times (t) and scales (s):

$$\psi_{s,t}(u) \equiv s^{-1/2}\psi\left(\frac{u-t}{s}\right)$$

  - In order to be a wavelet the function $\psi(u)$ must be more or less localized and satisfy the admissibility condition:

$$\int_{-\infty}^{\infty} \psi(u)du = 0, \quad \int_{-\infty}^{\infty} [\psi(u)]^2 du = 1$$

$$c_\psi = \int_{-\infty}^{\infty} \frac{|\hat{\psi}(\omega)|^2}{|\omega|^2} d\omega < \infty; \quad \hat{\psi}(\omega) \text{ is the FT of the wavelet.}$$

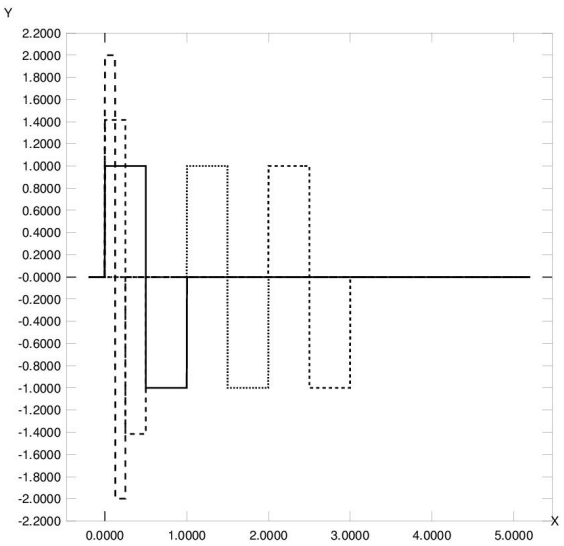
  - This implies that $|\hat{\psi}(\omega)|_{\omega=0} = 0$ i.e. wavelet acts as a band-pass filter.

Wavelet transforms

- An example: the Haar wavelet

$$\psi(u) = \begin{cases} 1, & 0 \le t < 1/2 \\ -1, & 1/2 \le u \le 1 \\ 0, & \text{otherwise} \end{cases}.$$

$$\psi_{jk}(u) = 2^{j/2}\psi(2^j x - k) .$$



- FT of the mother wavelet is

$$\hat{\psi}(\omega) = \frac{ie^{-i\omega/2}\sin(\omega/4)\mathrm{sinc}(\omega/4)}{\sqrt{2\pi}},$$

$$\mathrm{sinc}(x) = \frac{\sin \pi x}{\pi x}$$

$$c_\psi = 2\ln 2 .$$

Wavelet transforms

- The wavelet transform of function $f(u)$ is defined as

$$\tilde{f}(s,t) = c_\psi^{-1/2}|s|^{-1/2}\int_{-\infty}^{\infty} f(u)\psi_{s,t}(u)du .$$

- The signal can be reconstructed from the WT by the **inverse wavelet transform** (**IWT**)

$$f(u) = c_\psi^{-1/2}\int\int \frac{|s|^{-1/2}}{s^2}\psi\left(\frac{u-t}{s}\right)\tilde{f}(s,t)dsdt .$$

- WT and IWT can be computed based on these equations by utilizing Fourier transforms.

- In addition to actual wavelets usually on adds to the function basis so called scaling functions $\phi(u)$ that take care of the fact that the wavelets can not well describe a constant function (DC component).

- The most common and fastest way to compute the WT is the **discrete wavelet transformation** (**DWT**).

Wavelet transforms

- In a way there is redundancy in the CWT.

- In DWT we produce N wavelet coefficients from N original data points.

  - This is usually accomplished by only considering a certain subset of scales.

    - Assume the number of data points is a power of two: $N = 2^J$
    - We compute the coefficients only for scales 2^j, $j = 0, \ldots, J-1$
    - For scale 2^{j-1} there are $N_j = N/2^j$ wavelet coefficients and the 'times' (or arguments) associated with these points are $(2n+1)2^{j-1} - 1/2$, $n = 0, \ldots, N_j - 1$

Wavelet transforms

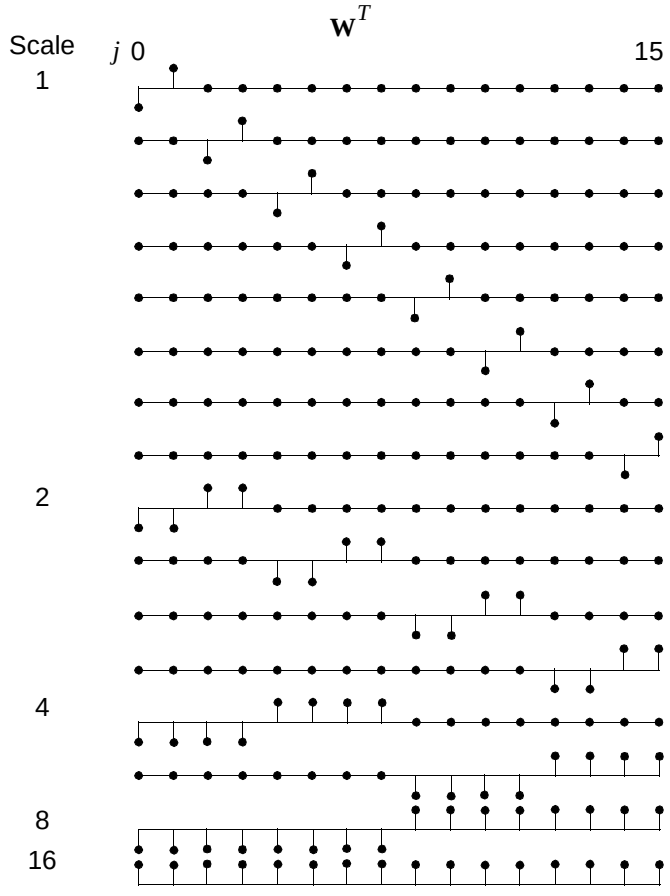
- A simple example:
  - Our original data is $\mathbf{X} = \{X_i\}$, $i = 0, \ldots, 15$.
  - CWT by using the Haar wavelet in matrix form:
    $\mathbf{Y} = \mathbf{W}\mathbf{X}$
  - A few examples of the colum vectors of $\mathbf{W}$

$$\mathbf{W}_{0,j}^T = \left[-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0, \ldots, 0\right]$$

$$\mathbf{W}_{8,j}^T = \left[-\frac{1}{2}, -\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, 0, \ldots, 0\right]$$

$$\mathbf{W}_{14,j}^T = \left[-\frac{1}{4}, \ldots, -\frac{1}{4}, \frac{1}{4}, \ldots, \frac{1}{4}\right]$$

$$\mathbf{W}_{15,j}^T = \left[\frac{1}{4}, \ldots, \frac{1}{4}\right]$$



Wavelet transforms

- To put it another way the output of the CWT gives us the following

$$\mathbf{Y} = \begin{bmatrix} Y_0 \\ \dots \\ Y_7 \\ Y_8 \\ \dots \\ Y_{11} \\ Y_{12} \\ Y_{13} \\ Y_{14} \\ Y_{15} \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}}(X_1 - X_0) \\ \dots \\ \frac{1}{\sqrt{2}}(X_{15} - X_{14}) \\ \frac{1}{2}(X_3 + X_2 - X_1 - X_0) \\ \dots \\ \frac{1}{2}(X_{15} + X_{14} - X_{13} - X_{12}) \\ \frac{1}{\sqrt{8}}(X_7 + \dots + X_4 - X_3 - \dots - X_0) \\ \frac{1}{\sqrt{8}}(X_{15} + \dots + X_{12} - X_{11} - \dots - X_8) \\ \frac{1}{4}(X_{15} + \dots + X_8 - X_7 - \dots - X_0) \\ \frac{1}{4}(X_{15} + \dots + X_0) \end{bmatrix}$$

- CWT gives information of the variation of the original data in different length scales.

Wavelet transforms

• Algorithms performing DWT can be viewed as digital filters.

- Let's take a simple example of the Haar wavelet.

- Now the basis functions are

$$\phi(u) = 1, \quad 0 \le u < 1$$

$$\psi(u) = \begin{cases} 1, & 0 \le u < 1/2 \\ -1, & 1/2 \le u \le 1 \\ 0, & \text{otherwise} \end{cases},$$

- The sequence of expansion functions is

$$n = 2^j + k,\ j \ge 0,\ 0 \le k \le 2^j$$

$$\psi_n(u) = 2^{j/2}\psi(2^j u - k)$$

- Each of these functions is supported on the interval $I_n = [k2^{-j}, (k+1)2^{-j}]$.

- The scaling function added to this set is

$$\phi(u) \equiv \psi_0(u) = 1, \quad u \in [0, 1].$$

Wavelet transforms

- We want to describe our data $x(t)$ in the wavelet basis:

$$x(t) = \sum_i \alpha_i \varphi_i .$$

- The coefficients α_i can be obtained from $\alpha_i = \langle \varphi_i, x(t) \rangle$.

- The basis functions for the Haar wavelet can now be written as

$$\varphi_{2k}(t) = \begin{cases} 1/\sqrt{2}, & n = 2k, 2k+1 \\ 0, & \text{otherwise} \end{cases}$$

$$\varphi_{2k+1}(t) = \begin{cases} 1/\sqrt{2}, & n = 2k \\ -1/\sqrt{2}, & n = 2k+1 \\ 0, & \text{otherwise} \end{cases} .$$

Wavelet transforms

- In matrix form this can be written as

$$\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \\ \alpha_6 \\ \dots \end{bmatrix} = \begin{bmatrix} y_1(0) \\ y_2(0) \\ y_1(2) \\ y_2(2) \\ y_1(4) \\ y_2(4) \\ \cdots \end{bmatrix} = \begin{bmatrix} a & a & & & & & \\ a & -a & & & & & \\ & & a & a & & & \\ & & a & -a & & & \\ & & & & a & a & \\ & & & & a & -a & \\ & & & & & & \dots \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ \cdots \end{bmatrix}, \qquad a = -\frac{1}{\sqrt{2}} .$$

- The vectors y_1 and y_2 can be seen as the time series of even and odd Haar wavelets.

- So, we generated from one data set of length (say) N two sets of length $N/2$.

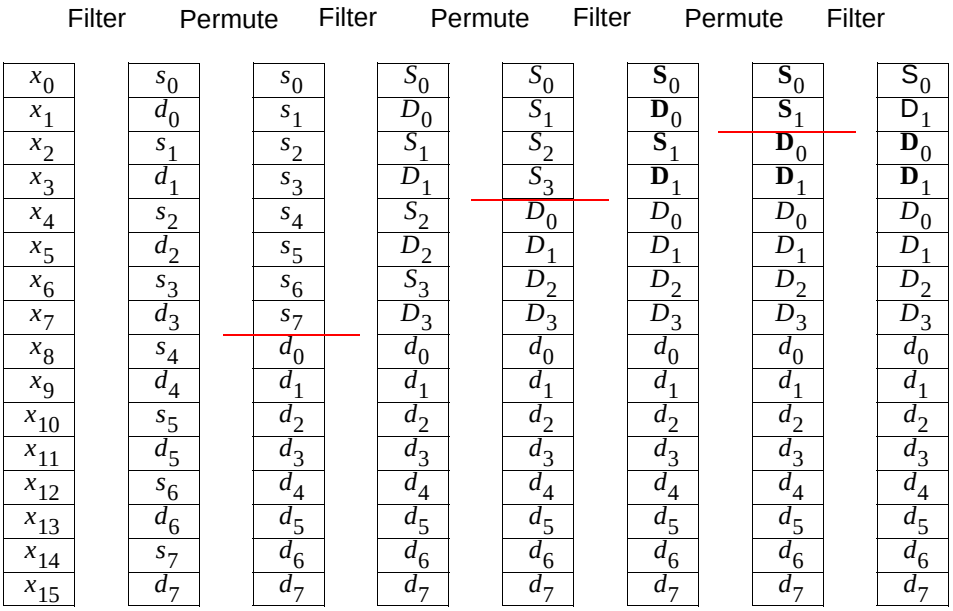
- One contains the smoothed information and the other the ‘detail’ information of the original signal.

- The Haar expansion can now be applied to the smoothed data set and this gives new sets of length $N/4$.

- If our original data set length is $N = 2^m$ we can continue the process until we end up with two data sets of of length 1.
  - One contains the average of the whole set and the other the difference between the averages of the first and second halves of the set.
  - This is called the pyramid algorithm.

Wavelet transforms

- Graphically:



- The inverse WT is obtained by applying the above procedure in the opposite direction and using the inverse of the coefficient matrix.

Wavelet transforms

• The DWT can be viewed as a digital filter:

$$x_n = \sum_{k=1}^{M} c_k x_{n-k}$$

- What wavelet one uses is determined by the **filter coefficients** c_k in the transformation matrix.
- These coefficients must fulfill various conditions:
  1. In order to compute the IWT easily the filter matrix is made orthogonal.
  2. The sequence of coefficients must a certain number of vanishing moments.

- For example so called DAUB4 wavelets are ‘generated’ by the matrix

$$\begin{bmatrix} c_0 & c_1 & c_2 & c_3 & & & & \\ c_3 & -c_2 & c_1 & -c_0 & & & & \\ & & c_0 & c_1 & c_2 & c_3 & & \\ & & c_3 & -c_2 & c_1 & -c_0 & & \\ & & & & c_0 & c_1 & c_2 & c_3 \\ & & & & c_3 & -c_2 & c_1 & -c_0 \\ & & & & & & . & \\ & & & & & & & . \\ & & & & & & & - \end{bmatrix}$$

Wavelet transforms

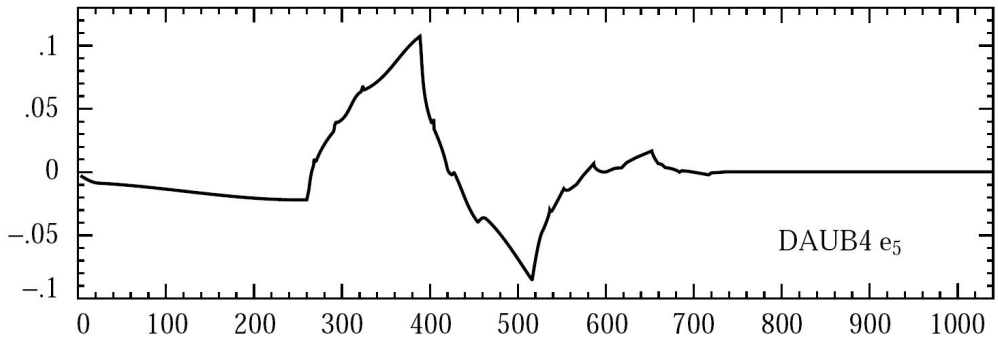
- The coefficients are obtained from equations

$$\begin{cases} c_0^2 + c_1^2 + c_2^2 + c_3^3 = 1 \\ c_2 c_0 + c_3 c_1 = 0 \end{cases}$$

$$\begin{cases} c_3 - c_2 + c_1 - c_0 = 0 \\ 0c_3 - 1c_2 + 2c_1 - 3c_0 = 0 \end{cases}$$

$$\rightarrow \qquad c_0 = \frac{1+\sqrt{3}}{4\sqrt{2}}, \qquad c_1 = \frac{3+\sqrt{3}}{4\sqrt{2}}, \qquad c_2 = \frac{3-\sqrt{3}}{4\sqrt{2}}, \qquad c_3 = \frac{1-\sqrt{3}}{4\sqrt{2}}.$$

- Graphically (from *NR*, Fig. 13.10.1)



Wavelet transforms

• Wavelet routines can be found in

GSL:
```
gsl_wavelet_transform_forward(w,data,1,n,work);
gsl_wavelet_transform_inverse(w,data,1,n,work);
```

Matlab: Wavelet Toolbox (from Mathworks; costs money)
Various free toolboxes (or collections of routines):
http://www-stat.stanford.edu/~wavelab/
http://taco.poly.edu/WaveletSoftware/

• Wavelets have been applied (among other things) in data compression, noise reduction and linear algebra.

- FBI uses wavelets to compress its fingerprint image database.

- The JPEG2000 standard uses wavelets.