Other Optimization Techniques

Conjugate Gradient

Similar to steepest descent, but slightly different way of choosing direction of next step:

$$\vec{x}_{r+1} = \vec{x}_r + \lambda_r \vec{s}_r$$

$$\vec{s}_0 = -\vec{g}_0$$

$$\vec{s}_{r+1} = -\vec{g}_{r+1} + \underbrace{\beta_{r+1}\vec{s}_r}_{\text{new term}}$$

λ_r is chosen to minimize $h(\vec{x}_{r+1})$. This yields $\vec{g}^{\,t}_{r+1}\vec{g}_r = 0$

Here we allow a further step in the $\vec{s}_r$ direction. One choice (Fletcher - Reeves) for β_{r+1} is

$$\beta_{r+1} = \frac{g^2_{r+1}}{g^2_r}$$

Newton-Raphson

Assume the function that we want to minimize is twice differentiable. Then, a Taylor expansion gives

$$h(\vec{x}+\vec{\alpha}) \approx a + \vec{b}^t\vec{\alpha} + \frac{1}{2}\vec{\alpha}^t C\vec{\alpha}$$

where

$$a = h(\vec{x}), \quad \vec{b} = \vec{\nabla}h(\vec{x}) = \vec{g}(\vec{x}), \quad C = \left(\frac{\partial^2 h}{\partial x_i \partial x_j}\right) = H$$

Now $\vec{\nabla}h(\vec{x}+\vec{\alpha}) \approx \vec{b} + C\vec{\alpha}$ Because C is symmetric (check)

For an extremum, we have $\vec{b} + C\vec{\alpha} = 0 \quad \vec{\alpha} = -C^{-1}\vec{b}$

or

$$\vec{x}_{r+1} = \vec{x}_r - H(\vec{x}_r)^{-1}\vec{g}(\vec{x}_r)$$

Newton-Raphson

$$\vec{x}_{r+1} = \vec{x}_r - H(\vec{x}_r)^{-1}\vec{g}(\vec{x}_r)$$

i.e., the search direction is $\vec{s} = H(\vec{x}_r)^{-1}\vec{g}(\vec{x}_r)$ and $\lambda = 1$

This converges quickly (if you start with a good guess), but the penalty is that the Hessian needs to be calculated (usually numerically)

Again, convergence is when $\vec{s}$ is sufficiently small

How would we calculate the Hessian numerically ? Use Lagrange polynomial in several dimensions and work it out

Bounded Regions

The standard tool for minimization in particle physics is the MINUIT program (CERN library). It has also made its way well outside the particle physics community.

Author: Fred James

Here is how MINUIT handles bounded search regions - it transforms the parameter to be optimized as follows:

$$\lambda' = \arcsin\left(2\frac{\lambda - a}{b-a} - 1\right) \qquad \lambda = a + \frac{b-a}{2}(\sin\lambda' + 1)$$

λ is the exernal (user) parameter

λ' is the internal parameter

MINUIT is available within PAW, ROOT …

MINUIT

MINUIT uses a (variable metric) conjugate gradient search algorithm (along with others). Basic idea:

• assume that the function to minimize can be approximated by a quadratic form near the minimum
• build up iteratively an approximation for the inverse of the Hessian matrix. Recall

$$h(\vec{x}+\vec{\alpha}) \approx h(\vec{x}) + \vec{\nabla} h(\vec{x}) \cdot \vec{\alpha} + \frac{1}{2}\vec{\alpha}^t H \vec{\alpha}$$

the approximation for the Hessian is updated as follows:

$$H_{i+1} = H_i + \frac{(\vec{x}_{i+1}-\vec{x}_i)\otimes(\vec{x}_{i+1}-\vec{x}_i)}{(\vec{x}_{i+1}-\vec{x}_i)\cdot(\vec{\nabla}h_{i+1}-\vec{\nabla}h_i)} - \frac{\left[H_i\cdot(\vec{\nabla}h_{i+1}-\vec{\nabla}h_i)\right]\otimes\left[H_i\cdot(\vec{\nabla}h_{i+1}-\vec{\nabla}h_i)\right]}{(\vec{\nabla}h_{i+1}-\vec{\nabla}h_i)\cdot H_i\cdot(\vec{\nabla}h_{i+1}-\vec{\nabla}h_i)}$$

where the $\otimes$ symbol represents an outer product of two vectors (a matrix)

$$\left(\vec{a}\otimes\vec{b}\right)_{ij} = a_i b_j$$

Fourier Transforms

Fourier transforms are very important
• as a way of summarizing the data with a few parameters
• because the transform of the data is itself very interesting (e.g., power spectrum, momentum⇔coordinate space representation,…)

$$H(f) = \int_{-\infty}^{\infty} h(t)\, e^{2\pi i f t}\, dt \qquad H(f) \text{ frequency domain representation}$$

$$h(t) = \int_{-\infty}^{\infty} H(f)\, e^{-2\pi i f t}\, dt \qquad h(t) \text{ time domain representation}$$

Warning: there is no unanimity on 2π factors in front of the integral. Often the angular frequency is used $\omega = 2\pi f$

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

Fourier Transform

Fourier Transform is a linear operation:

• transform of the sum of two functions is the sum of the transforms

• the transform of a constant times a function is constant times the transform

$h(t)$ real	$H(-f) = [H(f)]^*$
$h(t)$ imaginary	$H(-f) = -[H(f)]^*$
$h(t)$ even	$H(-f) = H(f)$
$h(t)$ odd	$H(-f) = -H(f)$
$h(t)$ real,even	$H(f)$ real, even
$h(t)$ real,odd	$H(f)$ imaginary, odd
$h(t)$ imaginary,even	$H(f)$ imaginary, even
$h(t)$ imaginary,odd	$H(f)$ real, odd

Fourier Transform

Further properties:

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

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

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

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

We are typically interested in the Fourier analysis of a discretely sampled data set. Define the time step (taken to be constant here) as Δ. The sampling rate (frequency) is $1/\Delta$. Define the samples as

$$h_n = h(n\Delta) \qquad n = \cdots, -3, -2, -1, 0, 1, 2, 3, \cdots$$

Nyquist frequency

$$f_c \equiv \frac{1}{2\Delta}$$ Nyquist frequency

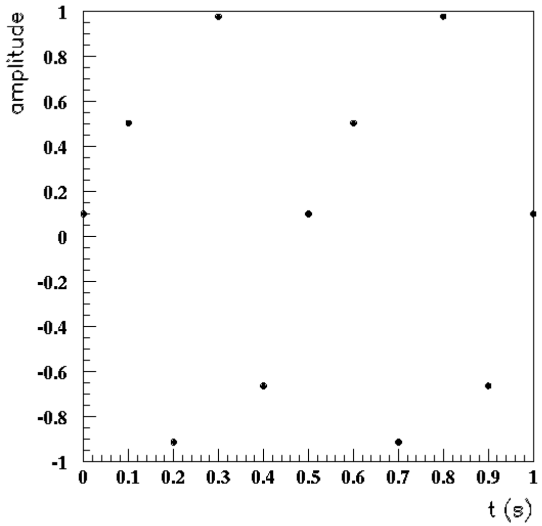
This is the highest frequency which can be resolved with a sampling frequency $f=1/\Delta$. If a continuous function $h(t)$ is limited in frequency components to frequencies less than f_c, then $h(t)$ is completely determined by its samples h_n. It can then be written as follows:

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

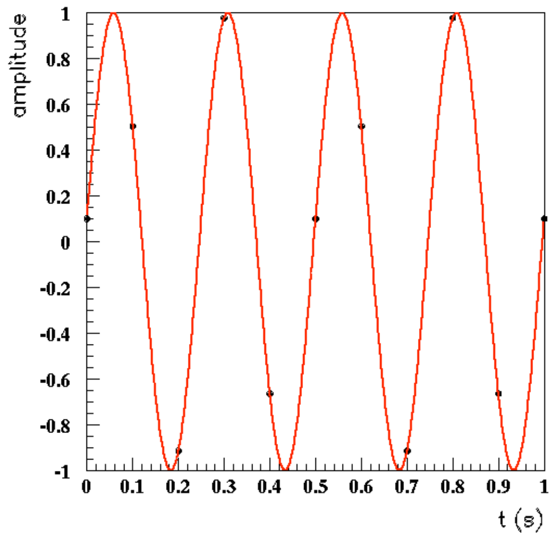
However, if there are frequency components which are higher than f_c, then they will be spuriously moved in the range $f<f_c$ (aliasing).

Example

Data

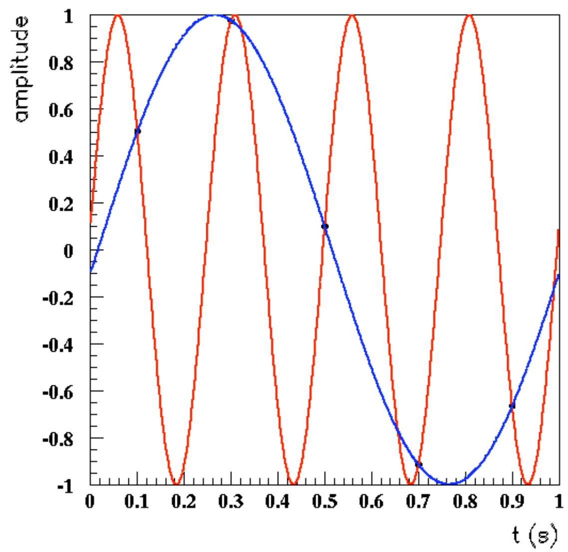


Fit



10 Hz sampling

Aliasing



5 Hz sampling

Conditions are:

Sine wave with f=4 Hz, phase offset φ=0.1 i.e.,

$$h(t) = \sin(\varphi + 2\pi f t) = \sin(0.1 + 8\pi t)$$

Example

$$H(f) = \int_{-\infty}^{\infty} \sin(0.1 + 8\pi t) e^{2\pi i f t} dt$$

$$= \int_{-\infty}^{\infty} \sin(0.1)\cos(8\pi t) e^{2\pi i f t} dt + \int_{-\infty}^{\infty} \cos(0.1)\sin(8\pi t) e^{2\pi i f t} dt$$

$$= \sin(0.1)\int_{-\infty}^{\infty} \frac{e^{8\pi i t} + e^{-8\pi i t}}{2} e^{2\pi i f t} dt + \cos(0.1)\int_{-\infty}^{\infty} \frac{e^{8\pi i t} - e^{-8\pi i t}}{2} e^{2\pi i f t} dt$$

Recall the relation:

$$\int_{-\infty}^{\infty} e^{2\pi i f x}\, df = \delta(x) \quad \text{where } \delta(x) \text{ is the Dirac Delta function}$$

so we have

$$H(f) = \sin(0.1)\int_{-\infty}^{\infty} \frac{e^{8\pi i t} + e^{-8\pi i t}}{2} e^{2\pi i f t} dt + \cos(0.1)\int_{-\infty}^{\infty} \frac{e^{8\pi i t} - e^{-8\pi i t}}{2} e^{2\pi i f t} dt$$

$$= \frac{\sin(0.1)}{2}\int_{-\infty}^{\infty} e^{2\pi i t(f+4)} + e^{2\pi i t(f-4)} dt + \frac{\cos(0.1)}{2}\int_{-\infty}^{\infty} e^{2\pi i t(f+4)} - e^{2\pi i t(f-4)} dt$$

$$= \frac{\sin(0.1)}{2}\left[\delta(f+4) + \delta(f-4)\right] + \frac{\cos(0.1)}{2}\left[\delta(f+4) - \delta(f-4)\right]$$

Discrete Fourier Transform

Suppose we have N consecutive sampled points

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

We can extract the amplitude for N frequency components since we have N data points. Define the frequency components as

$$f_n \equiv \frac{n}{N}\left(\frac{1}{\Delta}\right) \qquad n = -\frac{N}{2},\ldots,\frac{N}{2} \qquad \text{(take N even)}$$

Note: there are N+1 frequencies, but we will find that the two at the ends are equal, so only N independent. Negative frequencies allows us to include sine and cosine terms. So

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

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

Discrete Fourier Transform

Discrete Fourier Transform

The discrete fourier transform does not depend on any dimensional parameters.

Note $H_{-n} = H_{N-n}$ $\left[e^{2\pi ik(N-n)/N} = e^{2\pi ik}e^{-2\pi ikn/N} = e^{-2\pi ikn/N}\right]$

In particular $H_{-N/2} = H_{N/2}$

We can therefore rewrite the sum as follows

$$H_n \equiv \sum_{k=0}^{N-1} h_k e^{2\pi ikn/N} \qquad n = 0, \cdots, N-1$$

Discrete inverse Fourier transform

$$h_k = \frac{1}{N}\sum_{n=0}^{N-1} H_n e^{-2\pi ikn/N} \qquad k = 0, \cdots, N-1$$

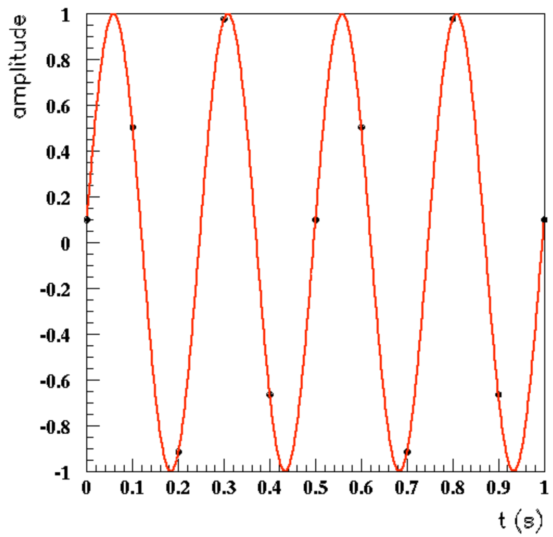
Discrete Fourier Transform

```
*
* Get the discrete Fourier components
*
      Do n=0,63
        Hn(n,1)=0.D0
        Hn(n,2)=0.D0
        Do k=0,63
          Hn(n,1)=Hn(n,1)
     &                  +amplitude(k,1)*dcos(twopi*k*n/64.)
     &                   -amplitude(k,2)*dsin(twopi*k*n/64.)
          Hn(n,2)=Hn(n,2)
     &                   +amplitude(k,1)*dsin(twopi*k*n/64.)
     &                   +amplitude(k,2)*dcos(twopi*k*n/64.)
        Enddo
        Write (11,*) N,Hn(N,1),Hn(N,2)
      Enddo
```

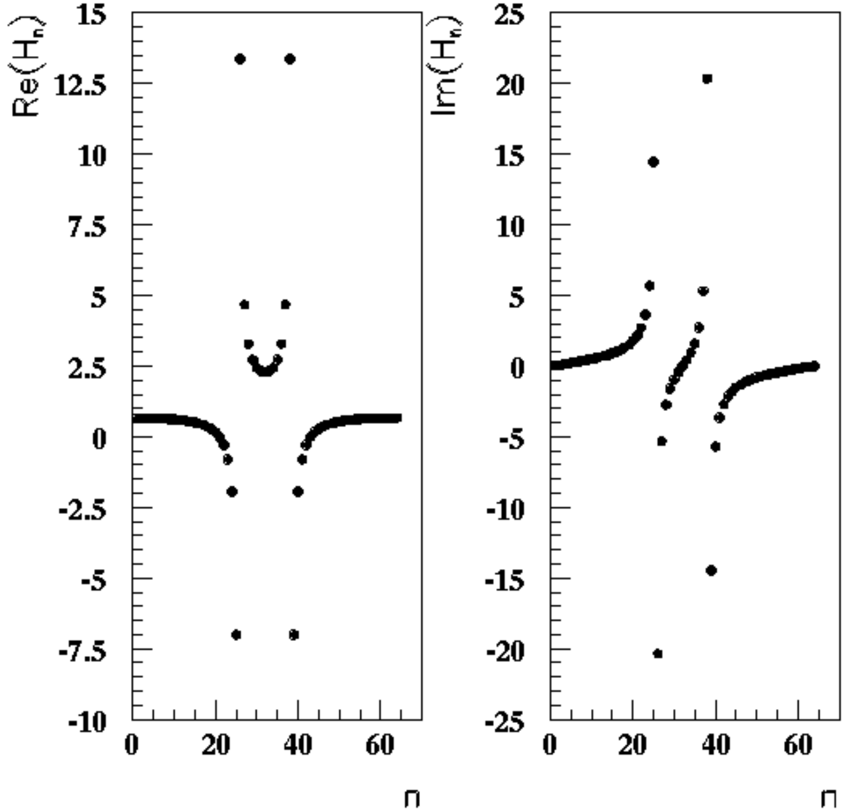
amplitude(k,1) real components
amplitude(k,2) imaginary components

Discrete Fourier Transform

Let's try it out on our sine wave data:
Recall, signal f=4 Hz



Here's the result (64 points):



Large components are:

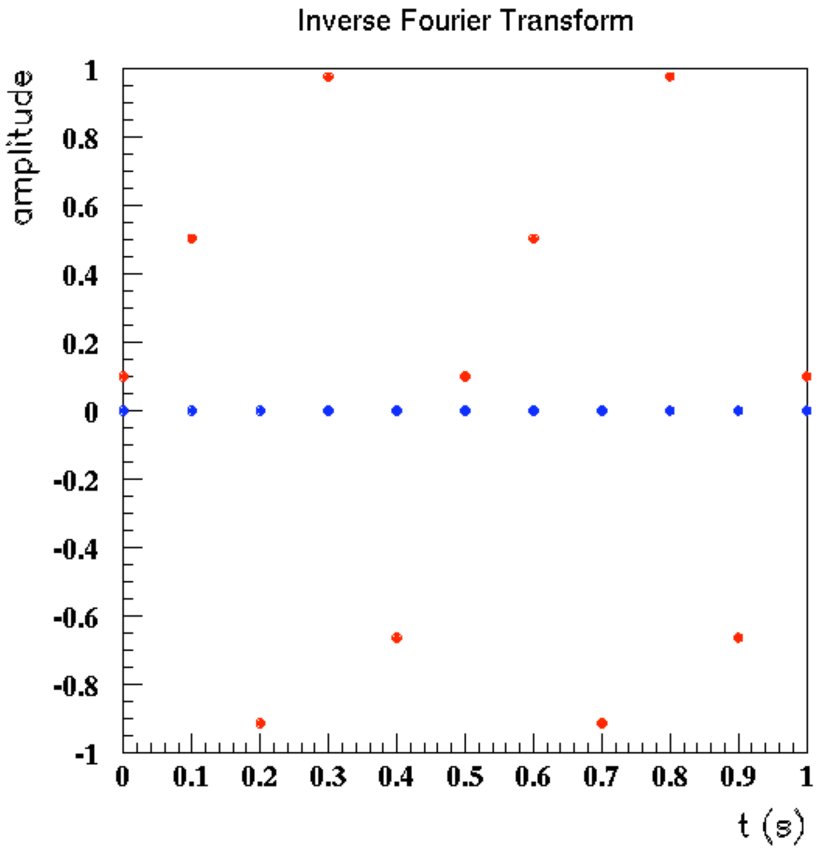
$$f_{25} = \frac{25}{64 \cdot 0.1} = 3.9, \quad f_{26} = \frac{26}{64 \cdot 0.1} = 4.06$$

$$f_{38} = f_{64-26} = f_{-26} \quad f_{39} = f_{64-25} = f_{-25}$$

cos and sin needed because of phase offset.

Here's the inverse transform

```
*
* Now we try the inverse transform
*
      Do n=0,63
        amplitude(n,1)=0.D0
        amplitude(n,2)=0.D0
        Do k=0,63
          amplitude(n,1)=amplitude(n,1)
     &                         +Hn(k,1)*dcos(twopi*k*n/64.)
     &                         +Hn(k,2)*dsin(twopi*k*n/64.)
          amplitude(n,2)=amplitude(n,2)
     &                         -Hn(k,1)*dsin(twopi*k*n/64.)
     &                         +Hn(k,2)*dcos(twopi*k*n/64.)
        Enddo
        Write (12,*) N,amplitude(n,1)/64.,amplitude(n,2)/64.
      Enddo
*
```



Fast Fourier Transform

The discrete Fourier transforms as we described it requires a sum over N terms for each of the N components. I.e., the number of operations scales as N^2. A large part of the success of Fourier transforms for analysis of electronic signals, optical images, x-ray tomography,…, results from the fact that a numerical algorithm was found which requires of order $N\log_2 N$ operations - the so-called Fast Fourier Transform (FFT). Here is how it works:

$$F_k = \sum_{j=0}^{N-1} e^{2\pi ijk/N} f_j = \sum_{j=0}^{N/2-1} e^{2\pi ik(2j)/N} f_{2j} + \sum_{j=0}^{N/2-1} e^{2\pi ik(2j+1)/N} f_{2j+1}$$

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

$$= F_k^e + W^k F_k^o \qquad \text{where} \quad W \equiv e^{2\pi i/N}$$

What is won ? The sums in the individual terms in the last line only have 1/2 as many terms, and the same factor appears

Fast Fourier Transform

To see how this works in detail, we take an explicit example of having 8 data points (taking a power of 2 is important ! If you don't have enough data, pad with zeroes).

$$F_k = F_k^e + W^k F_k^o$$

$$\text{where } \; W \equiv e^{2\pi i/N}, F_k^e \equiv \sum_{j=0}^{3} W^{2kj} f_{2j}, F_k^o \equiv \sum_{j=0}^{3} W^{2kj} f_{2j+1}$$

Now use a binary representation for the index $k=4k_2+2k_1+k_0$ where the k_i's are 0,1. Then,

$$W^{2kj} = \left(e^{2\pi i/8}\right)^{2(4k_2+2k_1+k_0)j} = e^{2\pi i(k_2+k_1/2+k_0/4)} = e^{2\pi i(k_1/2+k_0/4)} = W^{2j(2k_1+k_0)}$$

i.e., the k_2 bit is irrelevant. So,

$$F_k = F^e_{(k_1,k_0)} + W^k F^o_{(k_1,k_0)}$$

$$F^e_{(k_1,k_0)} \equiv \sum_{j=0}^{3} W^{2j(2k_1+k_0)} f_{2j} \qquad F^o_{(k_1,k_0)} \equiv \sum_{j=0}^{3} W^{2j(2k_1+k_0)} f_{2j+1}$$

Fast Fourier Transform

Let's try again:

$$F_k^e = \sum_{j=0}^{3} W^{2j(2k_1+k_0)} f_{2j} = \sum_{j=0}^{1} W^{2(2j)(2k_1+k_0)} f_{2(2j)} + \sum_{j=0}^{1} W^{2(2j+1)(2k_1+k_0)} f_{2(2j+1)}$$

$$= \sum_{j=0}^{1} W^{2(2j)(2k_1+k_0)} f_{2(2j)} + W^{2(2k_1+k_0)} \sum_{j=0}^{1} W^{2(2j)(2k_1+k_0)} f_{2(2j+1)}$$

$$F_k^e = F_k^{ee} + W^{2(2k_1+k_0)} F_k^{eo}$$

$$W^{4j(2k_1+k_0)} = \left(e^{2\pi i/8}\right)^{(8k_1+4k_0)j} = W^{2jk_0}$$

and $$F_k^o = F_k^{oe} + W^{2(2k_1+k_0)} F_k^{oo}$$

Can perform one more step:

$$F_k^{ee} = F^{eee} + W^{4k_0} F^{eeo} \quad F^{eee} = f_0 \quad F^{eeo} = f_4$$

The sums have disappeared !

Fast Fourier Transform

The final pieces are:

$$F_k^{ee} = F^{eee} + W^{4k_0} F^{eeo} = f_0 + W^{4k_0} f_4$$

$$F_k^{eo} = F^{eoe} + W^{4k_0} F^{eoo} = f_2 + W^{4k_0} f_6$$

$$F_k^{oe} = F^{oee} + W^{4k_0} F^{oeo} = f_1 + W^{4k_0} f_5$$

$$F_k^{oo} = F^{ooe} + W^{4k_0} F^{ooo} = f_3 + W^{4k_0} f_7$$

Note k_0=0,1
So, need 8 multiplications and 8 additions for this step

Then,

$$F_k^{e} = F^{ee} + W^{2(2k_1+k_0)} F^{eo}$$

$$F_k^{o} = F^{oe} + W^{2(2k_1+k_0)} F^{oo}$$

Here k_0=0,1 k_1=0,1
So, again 8 multiplications and 8 additions for this step

Finally $$F_k = F^{e} + W^{k} F^{o}$$

again 8 multiplications and 8 additions for this step

Fast Fourier Transform

So we need 2N operations per level, and there are $\log_2 N$ levels. The scaling of the computational time is therefore $N\log_2 N$ rather than N^2.

E.g., N=1000 $N\log_2 N \approx 1000*10=10^4$ $N^2=10^6$

How to implement in practice. Note that the trick is to find out which value of n corresponds to which pattern of e,o in

$$F^{eoeooeoe...} = f_n$$

Answer: reverse pattern of e,o. Assign e=0, o=1, and the binary value gives n.

examples

$$eee \rightarrow eee \rightarrow 000 \rightarrow 0$$

$$oeo \rightarrow eoe \rightarrow 010 \rightarrow 2$$

Some examples

. Agric. Food Chem., 52 (20), 6055 -6060, 2004. 10.1021/jf049240e S0021-8561(04)09240-4
Web Release Date: September 9, 2004

Discrimination of Olives According to Fruit Quality Using Fourier Transform Raman Spectroscopy and Pattern Recognition Techniques

Barbara Muik, Bernhard Lendl, Antonio Molina-Díaz, Domingo Ortega-Calderón,# and María José Ayora-Cañada*

Department of Physical and Analytical Chemistry, University of Jaén, Paraje las Lagunillas s/n, E-23071 Jaén, Spain; Institute of Chemical Technologies and Analytics, Vienna University of Technology, Getreidemarkt 9/164, A-1060 Wien, Austria; and CIFA Venta del Llano, IFAPA, Ctra. Bailén-Motril km 18.5, E-23620 Mengíbar, Jaén, Spain

Multiplication of large integers

The fastest known algorithms for the multiplication of large integers or polynomials are based on the discrete Fourier transform: the sequences of digits or coefficients are interpreted as vectors whose convolution needs to be computed; in order to do this, they are first Fourier-transformed, then multiplied component-wise, then transformed back.

…

Power Spectrum

The autocorrelation of a function is

$$Corr[y](\tau) = \int_{-\infty}^{\infty} y(t)^* y(t+\tau)\, dt$$

and the power spectrum is defined as the Fourier transform of the autocorrelation

$$PS[y](f) = \int_{-\infty}^{\infty} Corr[y](\tau) e^{2\pi i f \tau}\, d\tau$$

For a periodic function, the correlation is often defined as the expectation value. There is no convention on the normalization, so be careful about the values. Best to see which frequencies dominate a given spectrum. Here is a practical approach for a discretely sampled function:

$$C_k = \sum_{j=0}^{N-1} c_j e^{2\pi i jk/N} \qquad k = 0, 1, \ldots, N-1$$

Power Spectrum

$$P(0) = P(f_0) = \frac{1}{N^2}\left|C_0\right|^2$$

$$P(f_k) = \frac{1}{N^2}\left[\left|C_k\right|^2 + \left|C_{N-k}\right|^2\right] \quad k = 1,2,\ldots,\left(\frac{N}{2}-1\right)$$

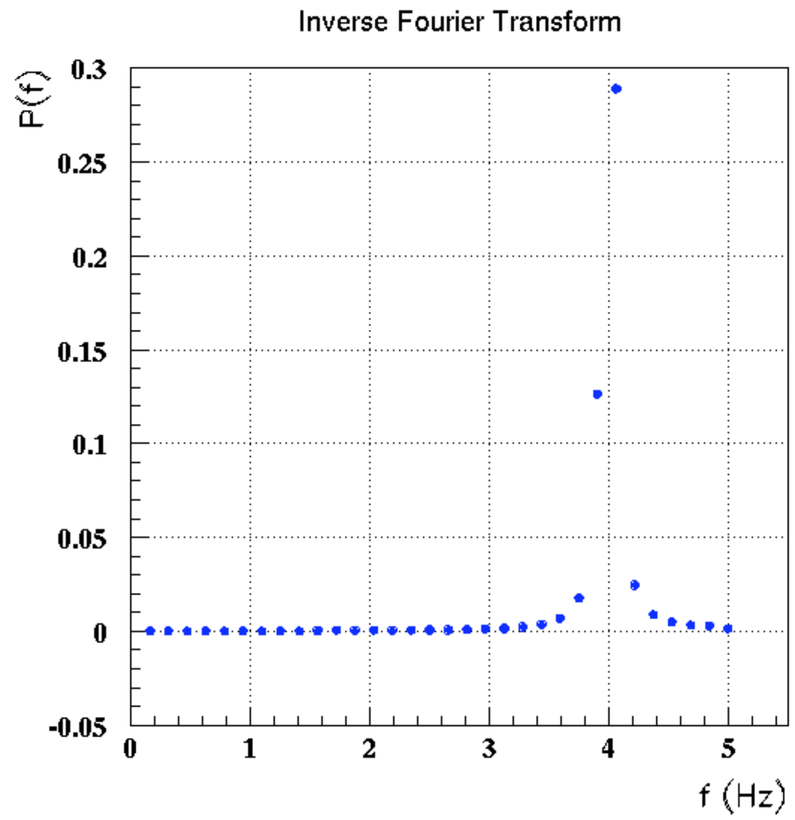
$$P(f_c) = P(f_{N/2}) = \frac{1}{N^2}\left|C_{N/2}\right|^2$$

where only positive frequencies are considered:

$$f_k \equiv \frac{k}{N\Delta} = 2 f_c \frac{k}{N} \qquad k = 0,1,\ldots,\frac{N}{2}$$

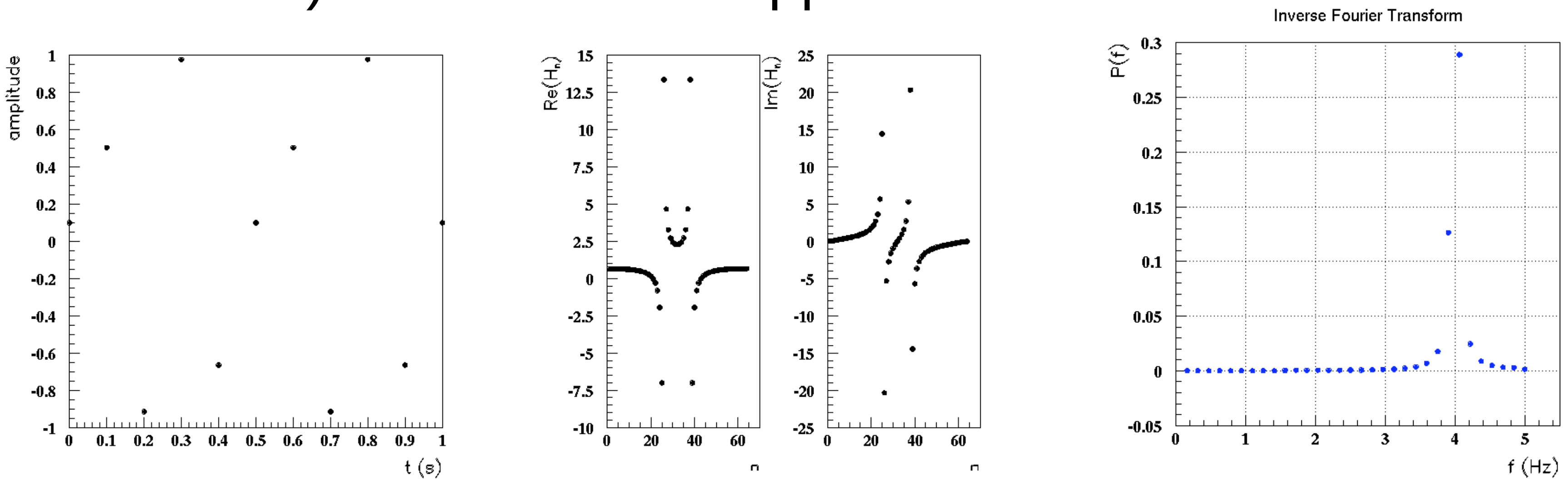
Let's try it out on our example:

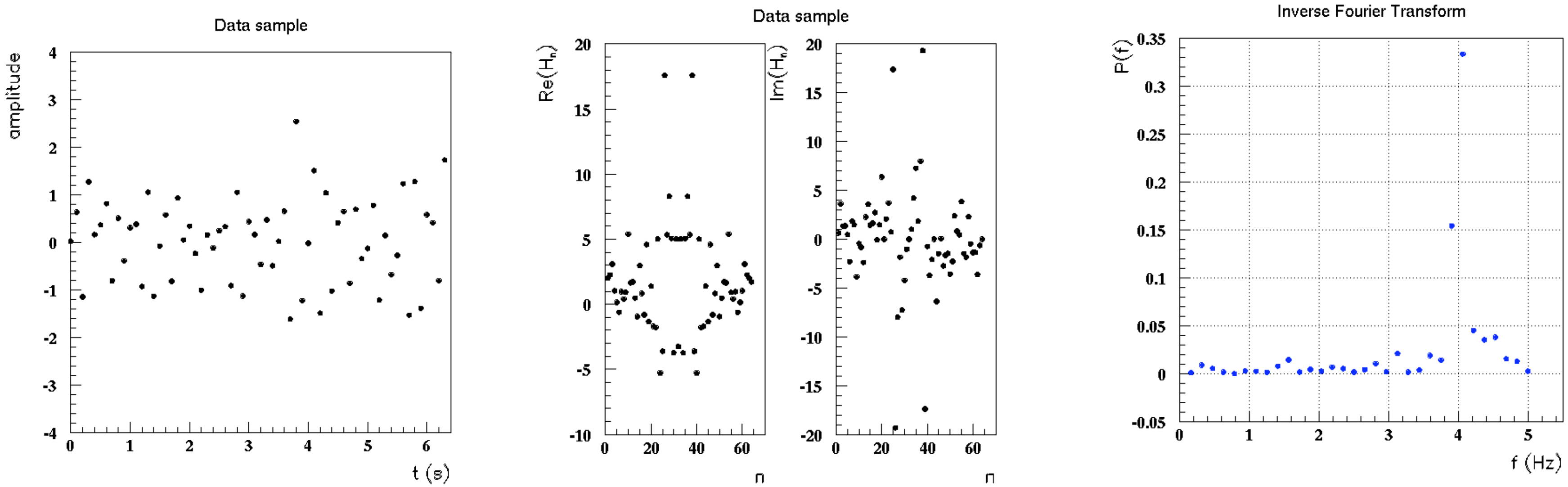
The 4Hz frequency is picked out.



Noise

We now add some noise to our spectrum (Gaussian smearing with σ=0.5) and see what happens:





Exercizes

1. Solve the χ^2 minimization problem from last lecture with MINUIT.

2. Generate 64 data points using

$$f(t) = \cos(\pi/4 + 2\pi f_1 t) + \cos(2\pi f_2 t) \qquad f_1 = 0.5, f_2 = 1 \qquad \Delta = 0.2$$

   and fit with a discrete Fourier transform. Extract the power spectrum.