Department of Physics
UMEÅ UNIVERSITY
Claude Dion[1]

January 22, 2009

Numerical Methods in Physics
Fast Fourier transforms

PURPOSE: To learn how to use routines for FFT for extracting information from a noisy signal.

LITERATURE:
M. Galassi *et al.*, GNU Scientific Library Reference Manual, 2nd ed. (Network Theory, Bristol, 2006), `http://www.gnu.org/software/gsl/`
W. H. Press *et al.*, Numerical Recipes in C, 2nd ed. (Cambridge University Press, Cambridge, 1992), chap. 12, `http://www.nrbook.com/a/bookcpdf.php`

[1]Based on work by Tord Oscarsson.

Introduction

This project concerns the use of FFT routines, and the main task here is to extract information from a noisy signal. The signal is encoded as an amplitude modulated (AM) carrier wave, and can be thought of as a model of an AM radio signal. Although the model is greatly over-simplified, you will get a feeling for how an AM signal is constructed, the function of the carrier wave and the side bands, as well as the limiting consequences of a finite bandwidth.

In order to get started, and to better understand how the FFT routines work, we start by analyzing a known signal. Then we gradually increase the complexity until, finally, you will be asked to decode a noisy signal.

GSL FFT routines

The GNU Scientific Library uses the following sign convention for the discrete Fourier transform. Given a series x_k of N time samples of a function $x(t)$, *i.e.*,

$$x_k \equiv x(t_k), \quad k = 0, \ldots, N-1, \tag{1}$$

with $x_k \equiv k\Delta t$, Δt being the time interval between samples, the *forward* transform (time to frequency) is defined as

$$X_j = \sum_{k=0}^{N-1} x_k \exp(-2\pi ijk/N). \tag{2}$$

The X_j correspond to the function $X(f)$ in the frequency domain, with the frequency itself given by

$$f_j \equiv \frac{j}{N\Delta t}, \quad j = -\frac{N}{2}, \ldots, \frac{N}{2}. \tag{3}$$

The *inverse* transform is then given by

$$x_k = \frac{1}{N} \sum_{j=-N/2}^{N/2} X_j \exp(2\pi ijk/N). \tag{4}$$

The signals considered in this exercise are real-valued, *i.e.*, $x_k \in \mathbb{R}$, while the Fourier transform is generally defined for complex functions. Within GSL, there exists two possibilities to treat this case. The first and simplest method is to put the values x_k in a complex array, with all the imaginary parts equal to zero. With the declaration

```
gsl_complex_packed_array data = calloc(2*N,sizeof(double));
```

we would have

$$\begin{array}{rcl}
\texttt{data[0]} & = & x_0 \\
\texttt{data[1]} & = & 0 \\
\texttt{data[2]} & = & x_1 \\
\texttt{data[3]} & = & 0 \\
 & \vdots & \\
\texttt{data[2*N-2]} & = & x_{N-1} \\
\texttt{data[2*N-1]} & = & 0
\end{array}$$

and the functions to use would be `gsl_fft_complex_radix2_forward`[2] and `gsl_fft_complex_radix2_inverse`[3]. To use these functions, you have to include the header file `<gsl/gsl_fft_complex.h>` in your source code. After the transform, the value of the X_j are recovered from `data` according to:

$$\begin{array}{rcl}
\texttt{data[0]} & = & \Re(X_0) \\
\texttt{data[1]} & = & \Im(X_0) \\
\texttt{data[2]} & = & \Re(X_1) \\
\texttt{data[3]} & = & \Im(X_1) \\
 & \vdots & \\
\texttt{data[N-2]} & = & \Re(X_{N/2-1}) \\
\texttt{data[N-1]} & = & \Im(X_{N/2-1}) \\
\texttt{data[N]} & = & \Re(X_{\pm N/2}) \\
\texttt{data[N+1]} & = & \Im(X_{\pm N/2}) \\
\texttt{data[N+2]} & = & \Re(X_{-(N/2-1)}) \\
\texttt{data[N+3]} & = & \Im(X_{-(N/2-1)}) \\
 & \vdots & \\
\texttt{data[2*N-2]} & = & \Re(X_{-1}) \\
\texttt{data[2*N-1]} & = & \Im(X_{-1})
\end{array}$$

Note that the middle elements `data[N]` and `data[N+1]` correspond to both $X_{N/2}$ and $X_{-N/2}$ (these two are equal), and that the array contains first the positive frequencies in increasing order, followed by the negative frequencies, also in increasing order.

The second option is to use real array to store the values of x_k, declaring `double data[N];` and using the function `gsl_fft_real_radix2_transform`. This is more efficient, as half the memory is needed and the computation involves fewer data,

[2]We consider here only values of N that are powers of 2, meaning that the radix-2 functions can be used.

[3]Note that there is also a function with the name ending with `backward`. It implements eq. (4) *without* the factor $1/N$, so `inverse` is a better choice.

but it makes the result of the forward FFT more difficult to manipulate, as the complex values X_j are stored in what is called a half-complex array, with a peculiar ordering. Please see section 15.6 of the GSL manual for more information. The inverse transform is then obtained with `gsl_fft_halfcomplex_radix2_inverse`, and both these functions are declared in the header file `<gsl/gsl_fft_real.h>`.

It is recommended you use the complex version of the FFT for this lab, but the choice is yours.

The Fourier transform of a Gaussian

Here we use as input a Gaussian. Since we know that the Fourier transform of a Gaussian is just another Gaussian, this example allows us to check the functionality of the FFT routines.

Write a main program that samples 1024 values from the function

$$x(t) = \frac{1}{\sqrt{\pi\sigma^2}} \mathrm{e}^{-t^2/\sigma^2}. \tag{5}$$

Choose σ to be $16\Delta t$ and use a step length $\Delta t = 1./1024$ s.

What is the frequency resolution in the Fourier transform for this case?

Calculate the Fourier transform and plot its real and imaginary parts. Check that the Fourier transform equals

$$X(f) = \mathrm{e}^{-\frac{1}{4}(2\pi\sigma f)^2}. \tag{6}$$

Check that the Fourier transform has the expected symmetries when x is real.

The spectrum of a simple AM wave

When two harmonic waves with the same amplitude $\bar{a}$ but different frequencies f_1 and f_2 interfere the result is a modulated wave

$$a(t) = 2\bar{a}\sin(2\pi f_\mathrm{c} t)\sin(2\pi f_\mathrm{b} t), \tag{7}$$

where $f_\mathrm{c} = (f_1 + f_2)/2$ is the center (or average) frequency and $f_\mathrm{b} = (f_1 - f_2)/2$ is the beat frequency. An example of such a wave is shown in figure 1.

We can imagine using the beat wave to transfer digital information by varying the modulation. Each modulation then represents a single bit, and we can use a large modulation to represent 1 while a small modulation represents 0.

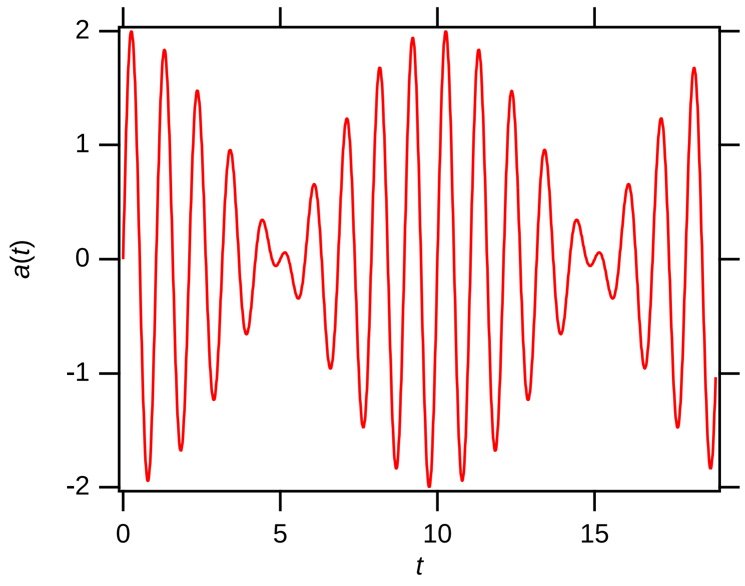

Figure 1: Example of an amplitude modulated wave $a(t) = \sin(2\pi \times 0.9t) + \sin(2\pi t)$.

Your task here is to create a simple AM wave representing the 8-bit pattern 01010101. To do this sample 1024 points with a sample frequency of 1024 Hz from the sinusoidal wave

$$u(t) = \bar{u} \sin(2\pi f_c t), \tag{8}$$

with $f_c = 1/(128\Delta t)$ and $\Delta t = 1/1024$ s. With the given frequency, the sequence will cover precisely 8 periods of the carrier wave and, therefore, to represent the 8-bit pattern we need to modulate each period. Use $\bar{u} = \bar{u}_0 = 1$ to represent the binary digit 0 and $\bar{u} = \bar{u}_1 = 3$ to represent 1. That is, the time sequence you should create is a sinusoidal wave with an amplitude of 1 during the first period, 3 during the second and so on.

Compute the Fourier transform $U(f)$ of the above AM wave, and plot the frequency spectrum

$$S(f) = |U(f)|^2 \tag{9}$$

for $f > 0$. Your spectrum should contain three main peaks corresponding to the carrier frequency, f_c, and the two *side bands* $f = f_c \pm \Delta f/2$, where Δf is the *bandwidth* of the signal. (The spectrum also contains some minor peaks, but these are of less importance.)

What is the bandwidth in this case? Could we have guessed this from the discussion in connection with equation (7)?

Try increasing $\bar{u}_1$ to 10. Does anything happen to the frequency spectrum?

Now, imagine that we double the frequency of the carrier wave, but that we still are restricted to use the same bandwidth as in the previous case. How many bits can then be encoded in a signal of the same length as above? Try it, and examine the spectrum.

Extracting information from a noisy signal

A code word has been encoded into a signal using the technique described in the previous section. The signal is stored in the file `/home/fnm/fft/data/am`*nnnn*`.dat`. Note that the data file is unique for each student. The data file contains 8192 samples and, for simplicity, let us say that the time resolution is 1/8192. (This assumption is not necessary, but simplifies the discussion below.) In addition to the amplitude-modulated carrier wave the signal has been scrambled with random noise.

Plot the data to convince yourself that no bit structure can be observed in the time series.

The carrier wave has a frequency of 1024 Hz and the random noise contains both lower and higher frequencies.

Compute the frequency spectrum and try to estimate the bandwidth Δf of the signal.

Your task here is to filter out the part of the data that contains the message. To do this multiply the Fourier transform of the data by the Gaussian filter

$$H(f) = \exp\left[-\frac{1}{2}\left(\frac{|f| - f_{\mathrm{c}}}{\Delta f}\right)^2\right] \tag{10}$$

and take the inverse transform. Note that the information in the transform lies in both positive and negative frequencies, and therefore, the filter must filter out both positive and negative frequencies with $|f|$ close to f_{c}. If you have done the filtering correctly the filtered signal should contain a clearly visible bit pattern. Each bit corresponds to a wave packet that occupies more than one period of the carrier wave. Each 8-bit binary pattern represents a character in ASCII code (see table 1). For example, the pattern 01000001 is binary for 65, and from table 1 the ASCII code 65 can be identified as the letter A.

The filtered time series is sensible to the bandwidth used in the Gaussian filter. A too narrow filter removes the side bands, and thereby also the information. On the other hand, a too wide filter allows part of the noise into the filtered signal,

Table 1:

ASCII CODES

Code	Char	Code	Char	Code	Char	Code	Char
0	NUL	32	SPACE	64	@	96	‘
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	’	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	\	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	DEL

[Last Update 9/26/95]

which makes the decoding more difficult. If you have trouble seeing a clear bit pattern, you can try experimenting with the bandwidth.

Read off the bit pattern from the filtered time series and decode the message. This message is your receipt for having solved the exercise successfully.

Bonus exercise: Include in your program the automatic decoding of the message. The filename containing the noisy message should be a command line argument and the program should return, on standard output, the sequence of characters constituting the message.

Presentation of results

Present your results in a brief report. All questions found in the instructions above must be answered. Especially, remember to present the code word you obtain, together with the name of the file you have analyzed. Do not append program listings to your report, but only a note giving the path to the code as well as instructions for running it. Make sure that the permissions allow anyone to read the content of the files (`chmod o+r *`). **Please note** that to pass the entire exercise, both the report and the code must be accepted by the reviewers.