B1 Engineering Computation Project

Wireless Communication Channels:
Characterisation using orthogonal functions

Michaelmas 2013 Justin Coon
Rev: October 30, 2013 justin.coon@eng.ox.ac.uk

What to do first

1. You will need a copy of these notes[1].
2. Once logged into a machine using your single-sign-on username and password visit the B1 Mini Projects directory on Weblearn. Navigate to the directory associated with this project, then download and unzip the file contained within.
3. Unzipping will create a B1 directory with various subdirectories in it.
   - Notes: containing pdf of these notes
   - Matlab: containing subdirectories
     - Fourier: contains .m files
     - GramSchmidt: empty
     - Laguerre: empty
     - FittingData: contains .m files and a .mat data file
4. Start up Matlab and point it to the Fourier directory ...

Take care to make backups

As this is an extended project, and you may wish to work on your own machine as well as on those in the Department, it is vital that you maintain a backup of your work, and keep track of the latest version. I suggest treating the Departmental copy as your master copy (as is it backed up properly). Also never rely on a memory stick for backup. Sticks are great for transporting stuff from A to B, but flash memory can die suddenly.

[1]Adapted from notes by David Murray and Paul Taylor.

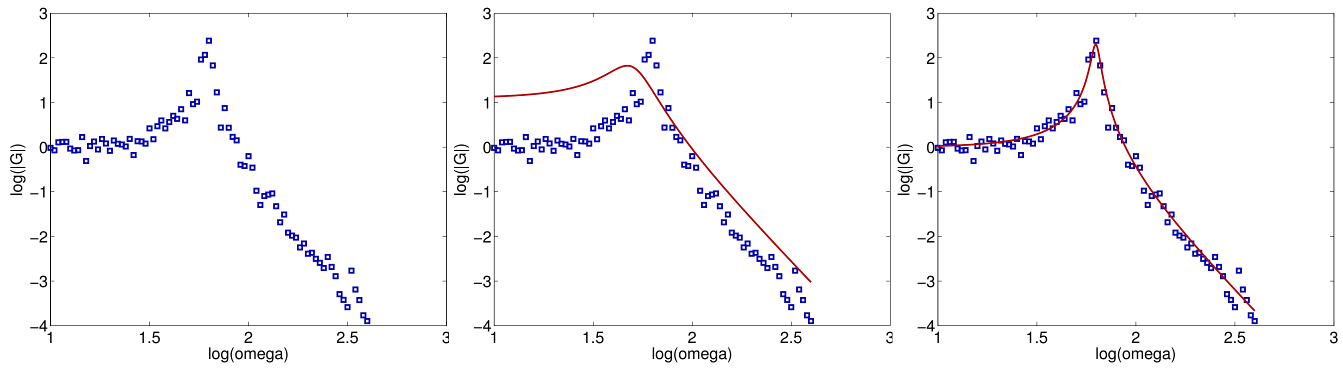
1 Introduction

1.1 Approaches to explaining data

In all areas of engineering, and in all empirical sciences, there is a need to explain experimentally observed data. While some experiments are designed to give a clear-cut yes/no answer to a question — Does the neutrino have mass? How many Oxford Engineering graduates does it take to change a lightbulb? — most measure the variation of one quantity with another, and some sort of model is devised which reproduces that variation. Indeed science usually progresses by experimentalists making observations, followed by theoreticians explaining them by devising a physically plausible, generative model. That model may predict other effects that further experiments may or may not confirm — or it may fail to predict something that is observed when more refined measurements are made.

1.1.1 Physics-based modelling

Often, engineers find the most satisfying modelling to be based on the physics of the underlying processes. For example you may have measured the gain of the output from a system as the frequency of the input is changed, as in Fig. 1.1(a).



(a) (b) (c)

Figure 1.1: Data description based on a physical model. (a) The measurements. (b) A fit with poorly-chosen parameters and (c) a fit with the optimal set of parameters.

You may postulate that the gain can be explained by a spring/damper model

$$|G|(\omega) = \frac{A\omega_0^2}{\sqrt{(\omega_0^2 - \omega^2)^2 + (2\zeta\omega_0\omega)^2}}. \tag{1.1}$$

The input here is the known angular frequency ω, which is under your control, and the output is the measured $|G|$ at that frequency. However, to describe the data we need to vary the parameters of the model — here the zero-frequency gain A, the resonant frequency ω_0, and the damping factor ζ. Figs. 1.1(b) and (c) show the data fitted using poor and good sets of parameters.

Note that although this model is based on physics, we are far removed from the inter-atomic forces giving rise to elasticity and viscosity. We have relied on someone else having devised simplified explanations of the bulk properties of matter.

1.1.2 Description rather than modelling

Physics-based modelling might be regarded as the ideal. However, in some areas of endeavour, although one can measure the variation of one quantity with another, it is not feasible to devise simple physical generative models that involve just a few parameters.

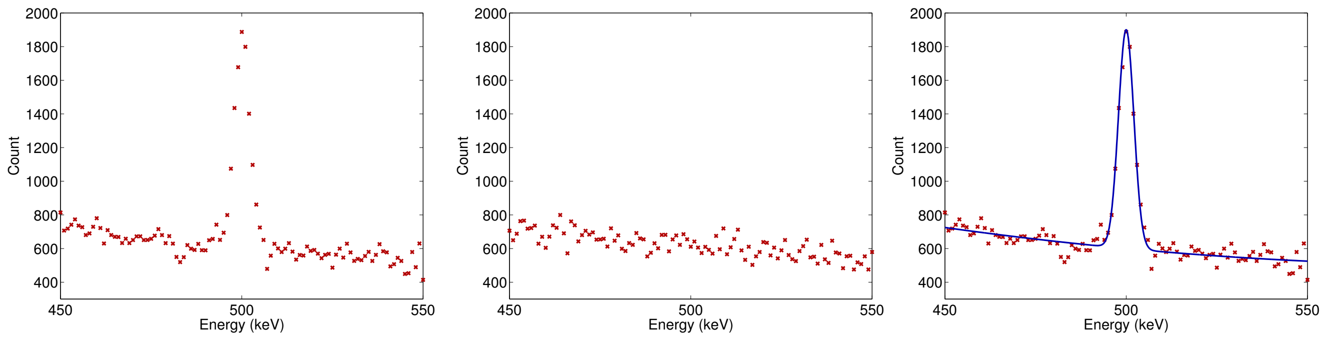
Despite this, it is often still useful to describe observed data economically, and in terms of trends (e.g., power law decay, exponential growth), because this allows easier comparison of the results from different experiments, and indeed might inspire the creation of a model. There are two approaches to this problem.

Parametric modelling

One approach is to represent the data with simple functions described by a few parameters. For example, suppose you were asked to analyse an experiment in gamma-ray spectroscopy — a subject about which you know nothing. You find that if a particular decaying isotope is present you observe a spectrum with a peak as in Figure 1.2(a); but if it is absent you observe a background spectrum as in Figure 1.2(b). You cannot explain why the background is as it is, but you might nonetheless describe it with a linear or weakly quadratic function, and similarly you might describe the decay peak with a Gaussian — both choices made because they provide good *descriptions* of the data, rather than reasoned *physical explanations* of the data.

The model has 6 parameters describing the spectrum $z(x)$ as a function of energy x, where x_0 is the centre energy of the peak, σ its width, and S its central height relative to a vertical offset a,

$$z(x) = [a + b(x - x_0) + c(x - x_0)^2] + S \exp\left[-\frac{(x - x_0)^2}{2\sigma^2}\right] . \tag{1.2}$$



(a) Spectrum with sample present (b) Spectrum of background (c)Description using function

Figure 1.2: Describing data using parametrised functions with no reasoned physical basis.

Non-parametric modelling

Another approach is to avoid parametric fitting, and use non-parametric methods. For example, in non-parametric Nadaraya-Watson *kernel regression* each observed datum position x_i is use as a centre on which a kernel function is placed. The fitted curve is given by a weighted average of the kernels

$$y_{fit}(x) = \frac{\sum_{i=1}^{n} y_i K_h(x - x_i)}{\sum_{i=1}^{n} K_h(x - x_i)}$$

where the weighting is the observed y_i at x_i.

A variety of kernel functions are used — top hats, triangular functions, quadratics, Gaussians and so on — but common traits are that they are symmetrical and have a characteristic width (or "bandwidth").

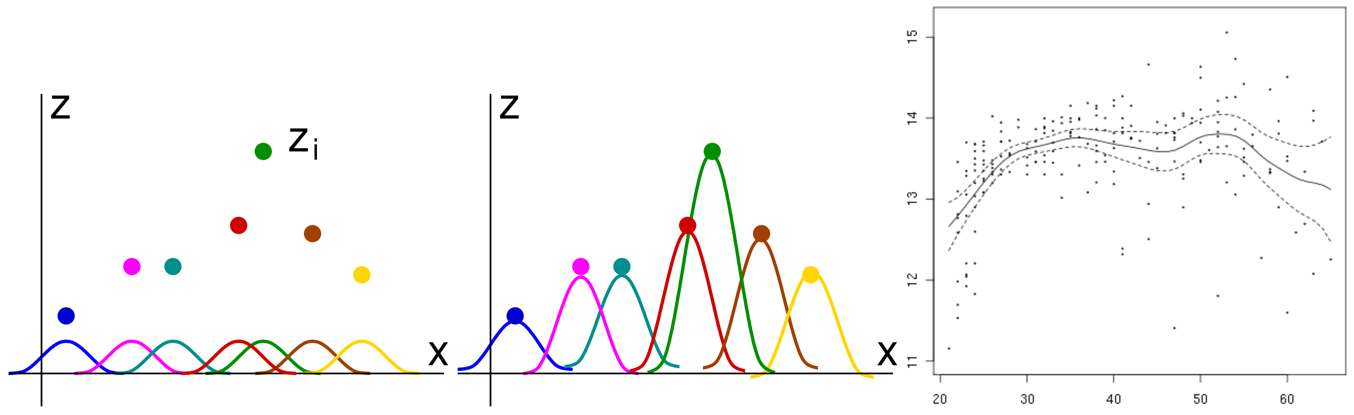


Figure 1.3: Fitting with Gaussian kernels. (a) Kernels placed at each x_i. (b) Scaled by y_i. (c) Example.

1.1.3 What do we mean by a good fit?

The B course on Estimation and Inference will tell you more about what constitutes a good description of data, but here we will assume the 'least-squares' criterion is appropriate.

Suppose we have a number of measurements z_i obtained at values x_i of an independent variable x (the dots in Fig. 1.4), and we have a model with a set of parameters $\mathbf{p} = (p_1, p_2, ..., p_n)$ that given x_i will produce the expected value $z_{fit}(x_i, \mathbf{p})$ (the line in Fig. 1.4).

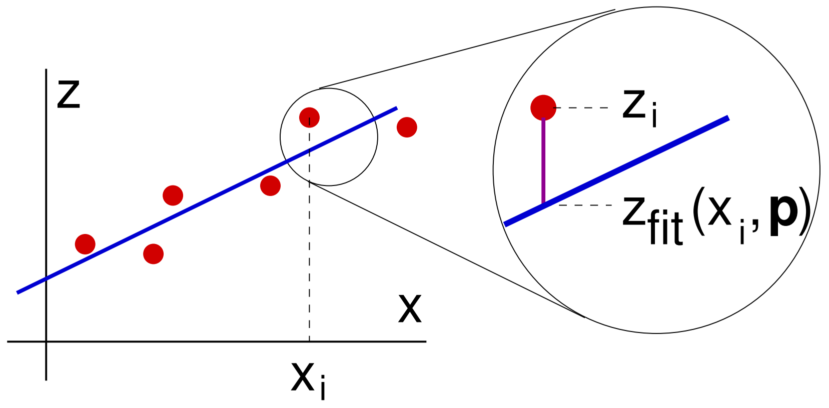


Figure 1.4: Fitting a model to observed data.

Assuming the uncertainty on each point is the same, the optimal least squares parameter set is that which minimises a cost $C(\mathbf{p})$ equal to the sum of the squares of the differences between measured and fitted values. That is we seek the optimal parameters $\mathbf{p}^*$, where

$$\mathbf{p}^* = \arg\min_{\mathbf{p}} [C(\mathbf{p})] = \arg\min_{\mathbf{p}} \left[\sum_i (z_i - z_{fit}(x_i, \mathbf{p}))^2 \right] . \quad (1.3)$$

The general method to find the n parameters $\mathbf{p}^* = (p_1, \dots, p_n)^*$ is to solve the set of n simultaneous equations

$$\frac{\partial C}{\partial p_1} = 0, \quad \frac{\partial C}{\partial p_2} = 0, \quad \dots, \quad \frac{\partial C}{\partial p_n} = 0 \quad (1.4)$$

and to check that all $\partial^2 C/\partial p_i^2 > 0$ so that the solution really is a minimum.

The above applies to any cost function, one that is non-linear or linear in the parameters. It is worth noting here that if the model is linear in the parameters, such as $z_{fit}(x_i) = p_0 + p_1 x_i + p_2 x_i^2 + p_3 x^3$, one can write all the fitted values in as a vector

$\mathbf{z}_{fit} = \mathtt{A}\mathbf{p}$ where $\mathtt{A}$ is called the *design matrix*. For the particular cubic example,

$$\mathtt{A} = \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & x_n^3 \end{bmatrix}.$$

The cost to minimise in Eq. (1.3) can be written as

$$C = (\mathbf{z} - \mathbf{z}_{fit})^T(\mathbf{z} - \mathbf{z}_{fit}) = (\mathbf{z} - \mathtt{A}\mathbf{p})^T(\mathbf{z} - \mathtt{A}\mathbf{p}). \tag{1.5}$$

Expanding this, differentiating with respect to $\mathbf{p}$, and solving $\partial C/\partial p_1 = 0$, etc, gives

$$\mathbf{p}^* = \left[\mathtt{A}^T\mathtt{A}\right]^{-1}\mathtt{A}^T\mathbf{z} \tag{1.6}$$

where $\left[\mathtt{A}^T\mathtt{A}\right]^{-1}\mathtt{A}^T$ is the Moore-Penrose pseudo-inverse of the design matrix $\mathtt{A}$.

1.2 This project

This B1 project involves representing data in the absence of a physics-based model, but using parametrised functions. Rather like the spectroscopy experiment, it is in an area about which you know precious little, but we will see how modelling can help us to get into the subject.

You are going to investigate the statistical properties of electromagnetic communication signals, more specifically in relation to Wi-Fi systems operating within an indoor environment. You will do this by constructing a mathematical model of the probability density function describing the attenuation caused by the scattering process in the wireless communication medium. Your model will be based on real measured data acquired by members of the Oxford Communications Research Group.

To fit the data you will use a set of orthogonal basis functions called *Laguerre functions*, but the first part of the project involves understanding why orthogonality is a useful property and verifying that the Laguerre functions are indeed orthogonal.

These notes will guide you through the initial stages of the project by providing examples, code and discussion. Eventually the detail will fade out, and you will be left to model the density function of the data provided on your own.

1.3 Your report

Imagine that you are a member of a research group in a company. Your group has undertaken a project to characterise the performance of Wi-Fi receivers operating in indoor environments. Your group leader is experienced in such activities, and decided to conduct a 'wireless channel measurement campaign' in order to acquire data that describes the power fluctuations of a received Wi-Fi signal.

You, on the other hand, are not experienced in the nuances of electromagnetic propagation and how it relates to communication system analysis and design, and enquired about the purpose of the measurement campaign to the group leader. It was explained to you that power fluctuations occur in wireless communication systems as a result of the information-bearing electromagnetic waves experiencing *scattering* in the environment. Scattering (as it relates to wireless communication systems) is the phenomenon by which waves impinge upon small objects relative to the wavelength, which causes them to change direction in the medium. Many of these waves eventually reach the receiver having taken different paths, and thus they have different amplitudes and phases (see Fig. 1.5). The communications receiver effectively collects the waves as a single observation, in which the collected waves may add up constructively (good - the received signal power is high) or destructively (bad - it is difficult to decode the transmitted information). The statistical properties of this observation (particularly the distribution of the power) are key to understanding how a communication system will perform in a given environment.

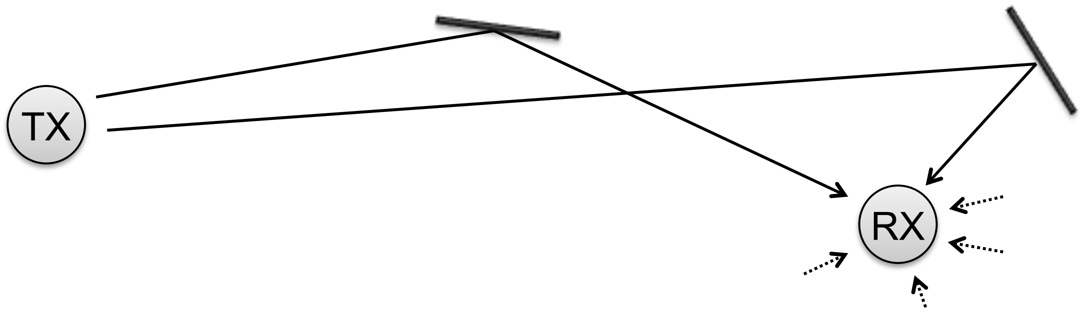


Figure 1.5: Illustration of radio wave propagation in a communication system.

The measured data is now in hand in the form of an empirical probability mass function for the received electromagnetic signal power (normalised in some way). This data is plotted in Fig. 1.6 below. Mass functions are useful for describing discrete random events, but your group leader informs you that you must have a continuous approximation (a *density function*) in order to analyse communication system performance (after all, power is a continuous quantity in this case). Thus, you have been asked to research ways to describe the data with a continuous mathematical model.

Your group leader is keen to understand what you learn about modelling, particularly as it applies to the Wi-Fi project data, and has thus requested that you write a

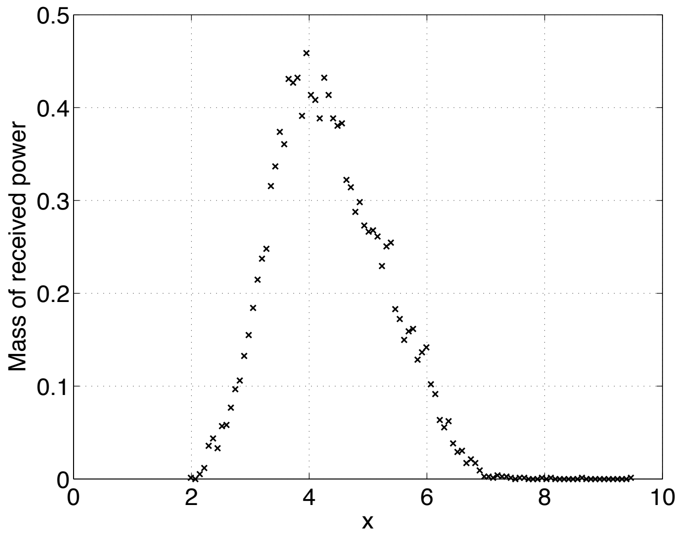


Figure 1.6: Measured mass plot of received signal power.

report giving details of your findings. Your report must be no more than 10 pages. It must read well in itself, and should be regarded as a mathematical and computational technical annexe to the larger Wi-Fi project. You are not expected to give any details of the Wi-Fi project in your report unless they directly relate to the modelling task. Remember, the goal of the report is to inform your group leader of modelling techniques, and in particular various aspects of orthogonal function theory and application.

2 Revision of Orthogonality and Matlab

We begin by (re-)revising the concepts of *orthogonality* and *orthonormality*. Then we will work up some Matlab code to test and use orthogonality in Fourier Series. You will not actually be using Fourier Series directly in your project, but their familiarity will help both (re-)establish techniques in mathematics and in Matlab coding.

2.1 Orthogonal basis functions - re-revision

Consider the length-3 vectors $\mathbf{g}_0$, $\mathbf{g}_1$, $\mathbf{g}_2$, which are of different magnitudes but lie at right angles to each other in three dimensional space. They form a set of *orthogonal basis vectors* in 3D. It should be evident that any 3D vector $\mathbf{f}$ can be described by a unique linear combination of the vectors

$$\mathbf{f} = a_0\mathbf{g}_0 + a_1\mathbf{g}_1 + a_2\mathbf{g}_2. \tag{2.1}$$

How would you find these unique coefficients for a particular $\mathbf{f}$? To find a_0, take the scalar ("dot") or inner product of both sides with $\mathbf{g}_0$

$$\mathbf{f}\cdot\mathbf{g}_0 = a_0\mathbf{g}_0\cdot\mathbf{g}_0 + a_1\mathbf{g}_1\cdot\mathbf{g}_0 + a_2\mathbf{g}_2\cdot\mathbf{g}_0 \tag{2.2}$$

and then exploit orthogonality which tells you that $\mathbf{g}_1\cdot\mathbf{g}_0 = 0$ and $\mathbf{g}_2\cdot\mathbf{g}_0 = 0$, so that

$$a_0 = \frac{\mathbf{f}\cdot\mathbf{g}_0}{\mathbf{g}_0\cdot\mathbf{g}_0}. \tag{2.3}$$

The denominator is important — $\mathbf{g}_0$ and so on were not unit vectors. The vectors were *orthogonal* but not *orthonormal*.

In the early 19th century it was realised that it was possible to treat functions in the same way and to make them up from a set of orthogonal *basis functions*. Let us assume that there is a set of orthogonal "g-functions", so that, similar to the vector case, but without limit to three dimensions,

$$f(x) = a_0g_0(x) + a_1g_1(x) + a_2g_2(x) + \cdots . \tag{2.4}$$

How do we find a_0 and so on? First we need to define the equivalent of the inner product between functions — but for now just assume we can and that the operation is linear. We denote it by $\langle f, g\rangle$. Then, by analogy

$$\langle f, g_0\rangle = a_0\langle g_0, g_0\rangle + a_1\langle g_1, g_0\rangle + a_2\langle g_2, g_0\rangle + \cdots . \tag{2.5}$$

But $\langle g_1, g_0 \rangle = 0$, and so on, so that

$$a_0 = \frac{\langle f, g_0 \rangle}{\langle g_0, g_0 \rangle}. \tag{2.6}$$

This approach can be used to find any coefficient a_n, and thus

$$a_n = \frac{\langle f, g_n \rangle}{\langle g_n, g_n \rangle} \quad \text{for } n = 0, 1, 2, \ldots. \tag{2.7}$$

2.2 Why use orthogonal functions?

Why is modelling or fitting to data using orthogonal basis functions desirable? The fundamental reason is that the coefficients a_n are unique. Suppose you model data using a non-orthogonal set, for example

$$f(x) = a_0 + a_1 x + a_1 x^2 + a_3 x^3 + \ldots \tag{2.8}$$

Suppose you have worked out (using least squares) the coefficients a_0 to a_3 for the first four terms, and then decide to add an extra term. Because this is a non-orthogonal set, adding in the extra term will change **all** the coefficients a_0 to a_3 too. Not only is this inefficient, but the lack of stability may lead to practical problems in cases where a robust design is required.

2.3 Fourier Series

The orthogonal basis set that you are probably most familiar with is set of sine and cosine functions making up a Fourier series, so we will use these as useful exemplars, not least to revise aspects of Matlab coding. Fourier found that the functions $\cos(n\omega x)$ and $\sin(n\omega x)$, $n = 0, 1, \ldots$ formed an orthogonal set when the inner product is defined by the integral over a period $T = 2\pi/\omega$, divided by the period.

$$\langle c_m, c_n \rangle = \frac{1}{T}\int_{-T/2}^{T/2} \cos m\omega x \cos n\omega x dx = \begin{cases} 0 & m \neq n \\ 1/2 & m = n > 0 \\ 1 & m = n = 0 \end{cases} \tag{2.9}$$

$$\langle s_m, s_n \rangle = \frac{1}{T}\int_{-T/2}^{T/2} \sin m\omega x \sin n\omega x dx = \begin{cases} 0 & m \neq n \\ 1/2 & m = n > 0 \\ 0 & m = n = 0 \end{cases} \tag{2.10}$$

$$\langle c_m, s_n \rangle = \frac{1}{T}\int_{-T/2}^{T/2} \cos m\omega x \sin n\omega x dx = 0 \tag{2.11}$$

Consequently, a periodic function $f(x)$, with spatial period T, can be written as the sum over these orthogonal cosine and sine functions

$$f(x) = \frac{1}{2}A_0 + \sum_{n=1}^{\infty} A_n \cos(n\omega x) + \sum_{n=1}^{\infty} B_n \sin(n\omega x) \tag{2.12}$$

with, following the recipe in Eq. (2.6),

$$\begin{aligned} A_n &= \frac{\langle f, c_n \rangle}{\langle c_n, c_n \rangle} = \frac{2}{T}\int_{-T/2}^{+T/2} f(x)\cos(n\omega x)dx \quad n = 0, 1, 2, \ldots \\ B_n &= \frac{\langle f, s_n \rangle}{\langle s_n, s_n \rangle} = \frac{2}{T}\int_{-T/2}^{+T/2} f(x)\sin(n\omega x)dx \quad n = 1, 2, \ldots . \end{aligned} \tag{2.13}$$

2.4 Using Matlab to compute Fourier series

The theory surrounding Fourier series is relatively straightforward. However, we can gain a lot of understanding of these rather remarkable and hugely important entities by considering practical examples and computation issues. Enter Matlab....

2.4.1 Matlab to check orthogonality

We begin with the basics. We wish to verify (at least for some examples) that the Fourier basis functions do indeed satisfy the orthogonality principle as outlined above. A numerical exploration of the orthogonality properties of Fourier series, or any orthogonal basis set for that matter, requires (i) sampled versions of the functions involved and (ii) a method of numerical integration.

Consider the Matlab script below, stored as fs_script1.m in B1/Matlab/Fourier. Note the use of `trapz` for numerical integration.

```
T = 1;                   % Define period
w = 2*pi/T;              % Derive omega
                         % Integral limits are -T/2 to +T/2
nint = 1000;             % We'll use a 1000 intervals
x = (-T/2:T/nint:T/2);   % to give vector of x values

% Here we test cos(mwx) times sin(nwx) with m=3 and n=5
m = 3; c = cos(m*w*x);   % Because x is a vector, so is c
n = 5; s = sin(n*w*x);   % and so is s
cs = c .* s;             % Compute their element by element product
% Now use the Trapezium Rule to integrate.
result = trapz(x,cs)     % This is the result: hope it is small!
```

We can plot the cos, sin and product function using

```
% Plot the c, s, and prod in red, green and blue
hold off; set(gca, 'FontSize', 18);
plot(x, c,  '-', 'LineWidth',2, 'Color', [0.7 0 0]); hold on;
plot(x, s,  '-', 'LineWidth',2, 'Color', [0 0.7 0]); hold on;
plot(x, cs, '-', 'LineWidth',3, 'Color', [0 0 0.7]); hold on;
xlabel('x'); ylabel('c,s, c times s');
% Save diagram as colour postscript
print('-depsc', 'fs_script1.eps');
```

TASK:

- Review the code. Note the use of comments, the use of many small intermediate steps, and the use of variable names that help to relate the code with theory.
- At the Matlab prompt, cd into your B1/Matlab/Fourier directory and run the code.
- Check the result is small! On my machine $\frac{1}{T}\int_{-T/2}^{T/2} cos(3\omega x) sin(5\omega x) dx \approx -6.8 \times 10^{-18}$.
- A plot appears on the screen of the cosine and sine functions, and their product (the integrand) as in Fig. 2.1. Check that it has been saved into the encapsulated-postscript .eps file as specified in the code.
  - You can save figures from the graphics window, but hard-coding print() is useful for the forgetful!
  - If you would prefer your diagrams did not get mixed up with the code you might write them to a higher directory
    print('-depsc', '../fs_script1.eps'); or
    print('-depsc', '../../EPSFigs/fs_script1.eps');
  - Other formats can be print()ed too. Use help print to see drivers other than -depsc.
- By editing the script you might adjust various aspects of the code — the number of intervals for example. The number might be too small (causing inaccuracy), or excessively large (wasting computational time).

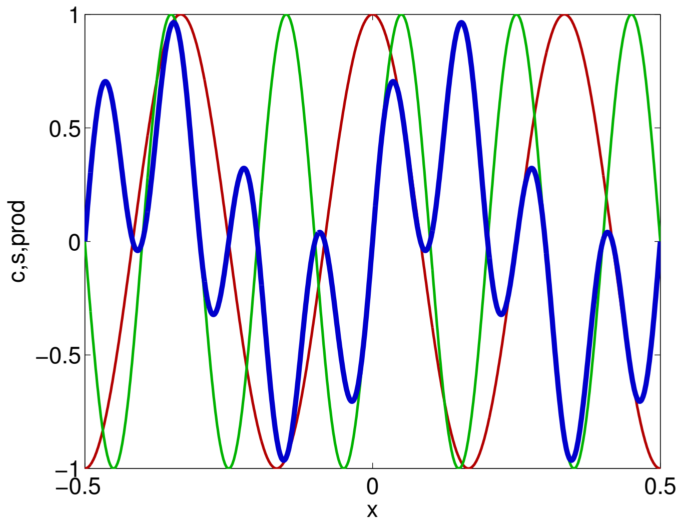


Figure 2.1: Plots of $cos(3\omega x)$ (red), $sin(5\omega x)$ (green) and their product (thick blue) over one period ($T = 1$). It is the thick blue curve that is integrated using the trapezium rule.

2.4.2 Reminder about Matlab functions

You just checked one instance of orthogonality. Suppose you wished to check orthogonality systematically for many different m and n values, and for sin $\times$ sin and cos $\times$ cos as well. It makes sense then to write a *function*, and call the function from a script.

Remember to put the function in a ".m" file with the same name. Here it is. (Notice the slightly feeble attempt at checking for errors in one of the input parameters. If you were writing production code you would want to ensure that all the parameters were in their proper ranges, and all functions would return error codes pre-defined for the particular application.)

```
% Computes inner products from Fourier series
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% fs_orthog.m DWM 5/9/10
% Integrates Fourier basis functions over a period
% e.g. \int cos(m w x)  sin(n w x) dx
% using the trapezium rule and returns scalar result
% Inputs:  T        period
%          nint     no of intervals
%          m,n      harmonics
%          code     'cc', 'cs', 'ss', 'sc'
%          indicates cos(m w x) * cos(n w x) etc
% Outputs: result (a scalar)
%          codeok =1 if ok, =0 if bad code
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [result,codeok] = fs_orthog(T,nint,m,n,code)
```

```
  codeok = true;                  % assume code is ok until ...
  w      = 2*pi/T;                % Derive omega
  x      = (-T/2:T/nint:T/2); % Make the  vector of x values

  if code(1) == 'c'               % Is the first code cos?
    v1 = cos(m*w*x);              % Yes. Hence cos (m w x)
  elseif code(1) == 's'           % No. Is it sin?
    v1 = sin(m*w*x);              % Yes. hence sin (m w x)
  else                            % Is it neither c nor s?
    codeok = false;               % Must be a bad code
  end

  if code(2) == 'c'               % Is the second code cos?
    v2 = cos(n*w*x);              % Yes. Hence cos (n w x)
  elseif code(2) == 's'           % etc
    v2 = sin(n*w*x);
  else
    codeok = false;
  end
  if codeok                        % Only integrate if code is ok
    result = trapz(x,v1.*v2)/T;% Trapezium rule integrates prods
  else
    result = 999;                  % return a silly result
  end
```

An example of checking Eq. 2.11 for $m = 3$ and $n = 3$ involves typing the following at the prompt.

```
>> [integral, codeok]=fs_orthog(1,1000,3,3,'cc')
integral = 0.5000
codeok = 1
```

TASK:

- Write a script which calls this function for $\cos \times \cos$ products inside two nested for-loops which run from $m = 0$ to 6 and $n = 0$ to 6.
- Store the integral result as elements of a 2D array called, say, `coscos(m,n)`.
- Repeat for $\sin \times \sin$ and $\sin \times \cos$ products.

You'll find all seems to be working! However, just because this code works a few times, you must not mindlessly rely upon it. For example instead of returning 0.5, this gives...

```
>> [integral, codeok]=fs_orthog(1,1000,1000,1000,'cc')
integral = 1.000
codeok = 1
```

What is the reason for this failure? (There must be a reason for the answer being spookily equal to 1.000.) How might you design your code to ensure such issues do not arise?

2.4.3 Code to compute Fourier series coefficients

You have tested orthogonality by multiplying samples of the sin and cos functions and integrating numerically. But computationally this is very similar to what you need to do to evaluate Fourier coefficients — multiply samples of the function $f(x)$ with either cos or sin, and integrate numerically, as stated in Eq. (2.13).

As an example, find the coefficients of the Fourier series for the even function $f(x)$ with period $T = 1$

$$f(x) = \begin{cases} 1 + 2x & \text{for} \quad -1/2 \le x < 0 \\ 1 - 2x & \qquad\quad 0 \le x < 1/2 \end{cases} \tag{2.14}$$

The plot of one period is given below.

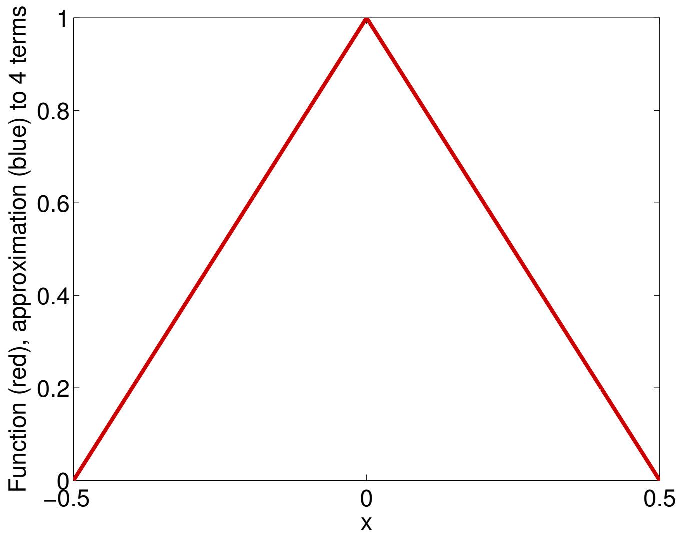


Figure 2.2: Plot of one period of the triangle function defined in Eq. (2.14).

First, we must write a Matlab function to compute $f(x)$ for a range of x values, perhaps held in a vector. The Matlab function below does exactly this. It computes samples of the function $f(x)$ for different values of x held in the vector, and returns the values in a vector. It is much more fiddly than one would expect because the function is periodic but the mathematical definition above applies to just one period. (There is probably a faster way using vectors throughout rather than the for loop!)

```
% Triangle function with period T. x is a vector.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Generates triangle with period T, unit height at
% origin and +/-nT, falling to zero at +/-(2n+1)T/2.
% Takes care not to assume x(i) is between -T/2 and +T/2
% jpc 11/10/13
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  function t = fs_periodictriangle(T,x)
  for i=1:length(x)
% First scale x(i) and shift to the right,
% so v has unit period defined by the interval [0,1].
% Then if v is outside the interval, bring it inside,
% and shift back into the central position.
    v = x(i)/T + 0.5;
    v = mod(v,1) - 0.5;
% v is now definitely between -0.5 and +0.5
    if v < 0
      t(i) = 1+2*v;  % use function definition for -ve
    else
      t(i) = 1-2*v;  % use function definition for +ve
    end
  end
```

Given the samples of x and $f(x)$, the coefficients are obtained from two functions which use the trapezium rule to perform the integration in Eq. (2.13).

First the cosine A_m coefficients:

```
function Am = fs_Acoeff(T,m,f,x)
    w = 2*pi/T;
    c = cos(m*w*x);
    Am = (2/T)*trapz( x, f.*c );
```

Then the the sine B_m coefficients:

```
function Bm = fs_Bcoeff(T,m,f,x)
    w = 2*pi/T;
    s = sin(m*w*x);
    Bm = (2/T)*trapz( x, f.*s );
```

Now stick everything together in a top-level function fs_triangle to compute the coefficients and series up to nterms. Note that here we limit x to one central period.

```
% Compute Fourier coeffs and series upto nterms for a triangle
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% fs_triangle.m DWM 6/9/10
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  function fs_triangle(T,nint,nterms)
  w = 2*pi/T;                       % Derive omega
                                    % Integral from -T/2 to +T/2
  x = (-T/2:T/nint:T/2);            % Make vector of x values
  f = fs_periodictriangle(T,x);     % Make vector of f(x) values
% Compute coefficients
  A0 = fs_Acoeff(T,0,f,x);          % A0 done separately
  for m=1:1:nterms
    A(m) = fs_Acoeff(T,m,f,x);      % The rest in a loop
    B(m) = fs_Bcoeff(T,m,f,x);      %
  end
% For fun, build the series approximation to nterms
  fseries = (A0/2)*ones(size(x));
  for m=1:1:nterms
    fseries = fseries + A(m)*cos(m*w*x) + B(m)*sin(m*w*x);
  end
% Plot original triangular function
  hold off;
  set(gca, 'FontSize', 18);
  plot(x, f, '-', 'LineWidth',3, 'Color', [0.8 0 0]);
  xlabel('x'); ylabel('Periodic Triangle Function'); hold on;
% Save diagram as colour postscript
  print('-depsc', 'fs_triangle.eps');
% Plot FS approximation to nterms
  hold off;
  set(gca, 'FontSize', 18);
  plot(x, fseries, '-', 'LineWidth',3, 'Color', [0.0 0.0 0.8]);
  xlabel('x');
  ylabel(sprintf('Fourier series: %d terms',nterms));
  hold on;
% Save diagram as colour postscript
% Filename will include the number of terms.
  print('-depsc', sprintf('fs_triangle%d.eps',nterms));
```

TASK:

- Run fs_triangle at the prompt with `nterms=4`. You should obtain results like those in Fig. 2.3(b).
- Vary `nterms` to assess at what point the reproduction is decent.
- Create a new Matlab function(s) to generate the periodic function shown in Fig. 2.4, where a is a parameter. Explore the number of terms required to provide a decent representation of the function as a gets smaller.
- How might you define "decent"?
- What do you notice about the B_m coefficients?

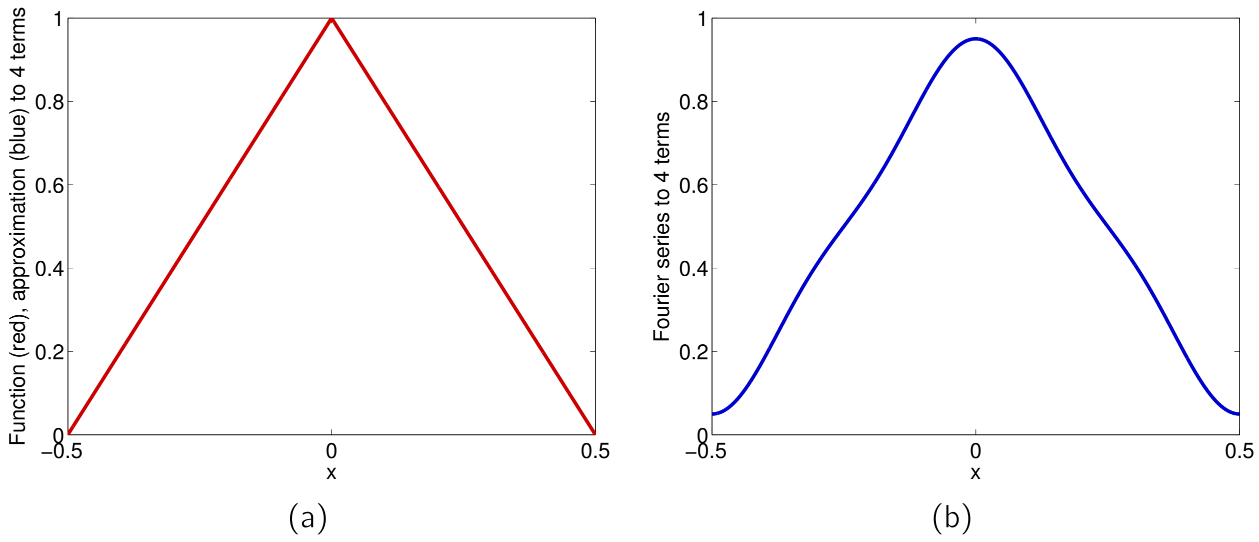


Figure 2.3: Plots of the triangle function, and its approximation computed up to $m = 4$.

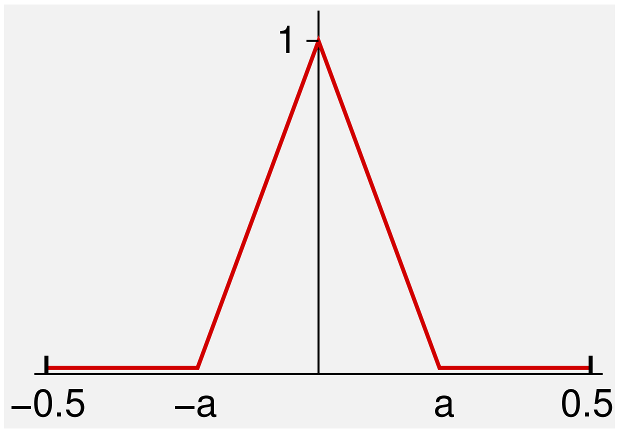


Figure 2.4: Function with period $T = 1$ comprising a triangle of basewidth $2a$ on a zero background.

2.5 Summary

So far you should have

- thought about issues in data description, and
- reminded yourself of orthogonality and why it is useful.

In Matlab, you have

- revised the use of Matlab scripts and functions,
- recalled that functions are placed in m-files with the same name. Functions have multiple arguments (input parameters) but return one item. To return multiple values, wrap them in a vector.
- remembered how Matlab and its functions can operate on vectors and matrices as single entities (e.g. cos(vector) returns a vector of cos's),
- explored input/output mechanisms, for-loops, arrays, and basic Matlab programming technique.

You should have noted too

- the use of the trapezium rule `trapz` for numerical integration
  - to test the orthogonality of two functions, and
  - to derive series coefficients using the inner product formulation.

So, on with the project proper.

3 Orthogonal Functions

In the last section, we (re-)revised Fourier series, paying particularly close attention to the orthogonality properties of the trigonometric basis functions sin and cos. While Fourier series have many interesting attributes and are incredibly useful for a myriad of engineering applications, they constitute but a drop in the ocean when it comes to the broader subject of orthogonal functions. In this part of the project, we will begin to explore other orthogonal basis functions, first in a general context (through the Gram-Schmidt orthogonalisation method) and later by considering particular example functions, all the while placing emphasis on the computational side of the subject. The code library developed in this section will play a key role in generating an appropriate fit to the measured/observed data in subsequent sections.

3.1 The Gram-Schmidt process

Let us assume that we have a set of functions (or vectors), which are linearly independent, and from which we wish to construct an orthogonal basis with respect to an inner product operator $\langle \cdot, \cdot \rangle$. We have already seen that from a modelling point of view such a basis is desirable as it leads to robustness in the model. Clearly, not all sets of linearly independent functions are orthogonal[1]. Thus, the question now is how can we construct an orthogonal basis set from these functions? Thinking geometrically, one approach might be to use one of the functions as a reference, then modify a second function such that it is orthogonal to the first (with respect to some definition of inner product). A third function could then be modified such that it is orthogonal to the first as well as the orthogonalised second function, and so on. This is exactly what the Gram-Schmidt process does in a systematic way.

Let us consider the process in more detail. Suppose that the linearly independent functions are $v_0(x)$, $v_1(x)$ and so on. Note that we do not necessarily have to constrain ourselves to a finite set of such functions. After all, we are working in a function space! In general, the Gram-Schmidt algorithm sets the first orthogonal function, call it g_0, equal to the first linearly independent function v_0. The second orthogonal function g_1 is the second linearly independent function minus any projection of it onto the first orthogonal direction, and so on. Repeating this process

[1]For example, one might consider the two lines $y_0(x) = 1$ and $y_1(x) = x$, which are linearly independent since there do not exist $c_0, c_1 \in \mathbb{R}$ that satisfy $c_0 y_0(x) + c_1 y_1(x) = 0$, but which are not orthogonal.

yields

$$\begin{aligned} g_0(x) &= v_0(x) && (3.1)\\ g_1(x) &= v_1(x) - e_{10}g_0(x) && (3.2)\\ g_2(x) &= v_2(x) - e_{20}g_0(x) - e_{21}g_1(x) && (3.3)\\ g_3(x) &= v_3(x) - e_{30}g_0(x) - e_{31}g_1(x) - e_{32}g_2(x) && (3.4)\\ &\vdots \end{aligned}$$

The requirement is that all the g_n functions should be orthogonal to one another, so working down through the list of equations we can find all the e values. For example

$$\begin{aligned} \langle g_1, g_0\rangle &= \langle v_1, g_0\rangle - e_{10}\langle g_0, g_0\rangle = 0 && (3.5)\\ \Rightarrow \qquad e_{10} &= \frac{\langle v_1, g_0\rangle}{\langle g_0, g_0\rangle} && (3.6) \end{aligned}$$

which allows $g_1(x)$ to be worked out for use in the next equation. From the next equation,

$$e_{20} = \frac{\langle v_2, g_0\rangle}{\langle g_0, g_0\rangle} \quad \text{and} \quad e_{21} = \frac{\langle v_2, g_1\rangle}{\langle g_1, g_1\rangle}. \tag{3.7}$$

By repeating this process, we are able to obtain an *orthogonal* basis set, but the functions in the set will not necessarily be *orthonormal*. To achieve orthonormality, we must perform the final normalisation step of dividing each orthogonal function by its norm (defined with respect to the associated inner product), i.e., the orthonormal functions are given by

$$\tilde{g}_n(x) = \frac{g_n(x)}{\sqrt{\langle g_n, g_n\rangle}}. \tag{3.8}$$

3.2 An important example

Now that we understand the basic premise of orthogonalisation via the Gram-Schmidt process, let's apply this to a specific example that we will use later to obtain a good fit to statistical data. For this example, we wish to obtain a set of polynomials that are orthogonal with respect to the inner product[2]

$$\langle g_n, g_m\rangle = \int_0^\infty g_n(x)g_m(x)e^{-x}dx. \tag{3.9}$$

[2]The reason we choose this definition of the inner product will become clear later.

It turns out that this can be done by considering the linearly independent set of monomials

$$v_0(x) = 1 \tag{3.10}$$
$$v_1(x) = x \tag{3.11}$$
$$v_2(x) = x^2$$

and so on. (...or we can succinctly write $v_n(x) = x^n$ to save us a lot of trouble.)

MAJOR TASK: Write structured, intelligible Matlab code that achieves the following:

1. Uses Gram-Schmidt to generate numerical values of the orthogonal set $g_0(x)$, $g_1(x)$ and so on, up to $g_5(x)$, over an appropriate range of x values. The computation of g_0, g_1 and so on, and the associated e-coefficients should occur within for-loops, so that higher orders can be computed easily by changing the upper bound of the loop.
2. Prints out values of the e coefficients used.
3. Verifies that all the functions g_n are orthogonal to one another.
4. Computes for each the normalising constant $\sqrt{\langle g_n, g_n \rangle}$.
5. Normalises the functions g_n, and verifies their orthonormality.
6. Plots the functions in different colours on a single graph.

Although one of the pleasures of Matlab is that one can play at the prompt, much time is saved through design. A good approach is to design the overall structure using functions to hide the detail, then write draft code for the functions so that they give "decent" results, allowing testing of the whole. Finally, you can go back to the functions and improve the code.

Some thoughts ...

1. You are going to need one vector of x-values, multiple vectors of the linearly independent functions $\{v_n(x)\}$, and multiple vectors of the resulting orthogonal functions $\{g_n(x)\}$.
2. It would be useful to construct a function that computes the inner product of function sample vectors a and b, given x.

3. What range for x? Think about the inner product definition. You know that the exponential will decay quicker than any polynomial can grow, which is why the inner product integral given in Eq. (3.9) converges. The trick is to make sure the volume of the integrand is mostly accounted for in the numerical computations. A conservative approach might be to plot the leading order term of the integrand for inner products of interest.

4. As the functions have indices, and you have to run through equations in order, it might be useful to store the vectors as rows in a 2D matrix, with each row containing the values for a particular function. You remember how to use a complete row ... for example ...

```
% Assume vx is the 2D matrix of linearly independent function values
  vector_of_3rd_order_values = vx(4,:);
```

   Did you notice the subtle indexing problem? We would like to include v_0, v_1, etc in this scheme, but Matlab does not use index 0 in vectors and matrices. The approach shown above uses an offset, but (if you are forgetful) you could write an incrementing function:

```
function o = id(i)
         o = i+1;
```

   and use it as ...

```
  vector_of_0th_order_values = vx(id(0),:); % mother function
  vector_of_3rd_order_values = vx(id(3),:);
```

   This will allow you to put everything into for-loops.

5. Be careful of numerical overflow problems. These polynomials grow quickly! What might you do to the definition of $v_n(x)$ to account for this problem?

One final, but very important, note: now that we have a set of orthogonal (indeed, orthonormal) polynomials defined with respect to a given inner product, we can easily construct a related set of orthogonal functions defined with respect to the inner product

$$\langle f_n, f_m \rangle = \int_0^\infty f_n(x) f_m(x) dx. \tag{3.12}$$

How do we do this? Simple, we let

$$f_n(x) = g_n(x) e^{-x/2}. \tag{3.13}$$

Similarly, we can write the set of corresponding orthonormal functions as

$$\tilde{f}_n(x) = \tilde{g}_n(x)e^{-x/2}. \tag{3.14}$$

We will explore the importance of this step later for a particular application.

In your report

1. explain the mathematics behind your method,
2. describe your implementation and supply the key parts of the code,
3. write clearly how to run your code,
4. describe the computational experiments you designed and performed to verify its performance,
5. give results, and importantly,
6. draw conclusions.

3.3 Explicit polynomial formulation

You now have a nice bit of Matlab code that can generate numerical values of a set of orthogonal polynomials. This is great for visualisation purposes, but your ultimate goal is to be able to express a data set as a weighted summation of a subset of orthogonal functions, analogous to a Fourier series representation of a function or data set.

How can we obtain explicit forms of these orthogonal functions? As it turns out, we can get a little bit of help from Edmond Laguerre....

4 Laguerre Functions

Edmond Laguerre was a 19th century French mathematician who made contributions to the subjects of complex analysis and geometry. He was plagued by health problems in his youth, but showed great talent for languages and mathematics. Laguerre attended the École Polytechnique in Paris, graduating in 1854. He subsequently took up a career as an artillery officer in the military. During the decade that followed, he maintained a keen interest in mathematics, and eventually left the military to return to the École to teach and conduct research. Of his many contributions to mathematics, perhaps the most notable is the work he did on differential equations that resulted in the identification of a set of polynomials that are orthogonal with respect to the exponential weighting function, and which today bear his name.

Figure 4.1: Edmond Laguerre.

4.1 Laguerre polynomials

The Rodrigues formula for the nth order *associated Laguerre polynomial* with parameter α is

$$L_n^{(\alpha)}(x) = \frac{1}{n!} x^{-\alpha} e^x \frac{d^n}{dx^n}(x^{n+\alpha} e^{-x}), \qquad n \in \mathbb{N},\ \alpha \in \mathbb{R}. \tag{4.1}$$

Laguerre polynomials form an orthonormal set under the inner product operation defined by

$$\int_0^\infty x^\alpha e^{-x} L_n^{(\alpha)}(x) L_m^{(\alpha)}(x) dx = \begin{cases} \frac{\Gamma(n+\alpha+1)}{\Gamma(n+1)} & \text{for } m = n \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

where $\Gamma(p) = \int_0^\infty t^{p-1} e^{-t} dt$ is the so-called *gamma function*, which satisfies $\Gamma(p) = (p-1)!$ when p is a positive integer.

Eq. (4.1) is a very handy expression for computing Laguerre polynomials since, as long as we can perform the differentiation efficiently, we can directly calculate the polynomial for any order n. Of course, the tricky part is performing the differentiation. For small n, this is easily done using the product rule repeatedly, but what about for larger n, say, $n = 7$ or even higher? It would take a long time to do this calculation by hand[1].

Upon closer inspection of Eq. (4.1), we can deduce that the nth derivative will just be an algebraic expression with maximum degree $n + \alpha$ multiplied by e^{-x}. Thinking about this further, we see that we can just factor out x^α from this expression and obtain a polynomial of degree n. The x^α and e^{-x} terms will be cancelled by $x^{-\alpha}e^x$ in Rodrigues' formula. So all we need to do is find the coefficients of the polynomial, and multiply them by $1/n!$.

TASK:

- Write a Matlab function that returns the vector of polynomial coefficients for the Laguerre polynomial $L_n^{(\alpha)}(x)$, where α and n are input parameters.

  Hint: It may be helpful to use Leibniz's rule for generalising the product rule in differentiation.

$$(f \cdot g)^{(n)} = \sum_{k=0}^{n} \binom{n}{k} f^{(k)} g^{(n-k)} \tag{4.3}$$

- Generate the polynomial coefficients for $L_5^{(1)}(x)$. Verify that the coefficients are $1/120 \times (-1, 30, -300, 1200, -1800, 720)$.

Suppose we are interested in generating the first n' polynomials in the set $\{L_n^{(\alpha)}(x)\}$ rather than just one polynomial specifically. Computationally, Rodrigues' formula is inefficient (why?). But if we take a close look at this formula, it becomes clear that the nth polynomial has degree n. At a superficial level, this may lead us to ask if there exists a nice recurrence relation for Laguerre polynomials. It turns out that there is; the polynomials can be generated successively by the following relation:

$$nL_n^{(\alpha)}(x) = (2n + \alpha - 1 - x)L_{n-1}^{(\alpha)}(x) - (n + \alpha - 1)L_{n-2}^{(\alpha)}(x) \tag{4.4}$$

where the first two polynomials are given by

$$L_0^{(\alpha)}(x) = 1 \tag{4.5}$$
$$L_1^{(\alpha)}(x) = -x + \alpha + 1. \tag{4.6}$$

[1]Applying the product rule repeatedly for $n = 7$ would require us to write down 128 terms!

Thus, if we wish to generate the polynomials $L_0^{(\alpha)}(x)$, $L_1^{(\alpha)}(x)$, and so on up to $L_{n'}^{(\alpha)}(x)$, it makes sense to write a simple software routine that will achieve this using the recurrence relations given above.

TASK:

- Write a Matlab function that successively computes the coefficients that define the set of associated Laguerre polynomials up to order n, where n and α are inputs. The function should return an array of polynomial coefficients.

  Hint: It may be helpful to write a function that multiplies two polynomials to produce a third polynomial, perhaps by taking as inputs the coefficient vectors describing the two input polynomials (e.g., `polymultiply([a b],[c d])`).

- Include a routine in your function that plots the first six polynomials ($n = 0, \ldots, 5$) in different colours over the domain and range specified by `xmin = -5, xmax = 20, ymin = -10, ymax = 20`. (Use the `axis` function in Matlab to set the plot limits automatically.)

For $\alpha = 0$, the plot of the first six polynomials should look a lot like:

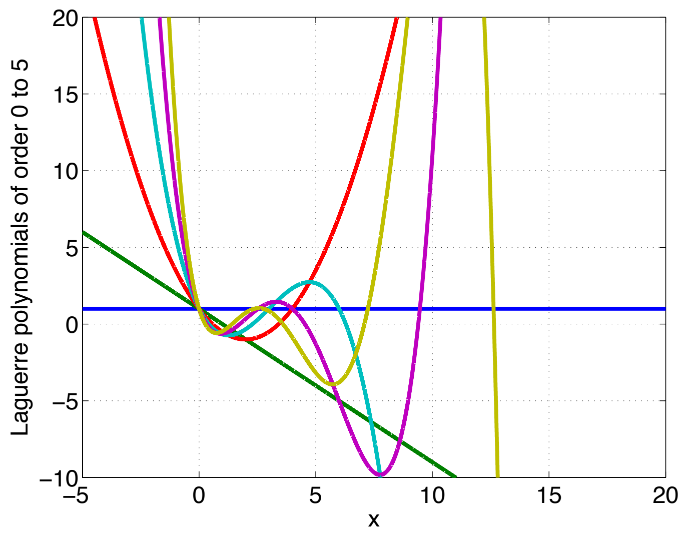


Figure 4.2: The first six Laguerre polynomials for $\alpha = 0$.

4.2 Comparison to your polynomials

The observant reader will have noticed that the inner product definition in Eq. (4.2) is identical to Eq. (3.9) for $\alpha = 0$. Does this imply that your orthonormal polynomial set $\{\tilde{g}_n(x)\}$ is actually a set of Laguerre polynomials?

TASK:

- Compare the orthonormal set of functions $\{\tilde{g}_n(x)\}$ to the set of Laguerre polynomials with $\alpha = 0$ by plotting their difference for $0 \leq x \leq 20$.
- What do you observe through this comparison? There should be a pattern of glaring differences in the functions. Identify the source of these differences (mathematically) and describe the adjustment that should be made to the original set of linearly independent functions $\{v_n(x)\}$ that would, through the application of the Gram-Schmidt process, result in $\tilde{g}_n(x) = L_n^{(0)}(x)$ for all n.
- Finally, create a function that constructs an orthonormal set of *associated Laguerre functions*, which are defined as

$$\psi_n^{(\alpha)}(x) = \sqrt{\frac{\Gamma(n+1)}{\Gamma(n+\alpha+1)}} x^{\alpha/2} e^{-x/2} L_n^{(\alpha)}(x). \tag{4.7}$$

  Design your function to plot $\psi_0^{(\alpha)}$, $\psi_1^{(\alpha)}$, up to $\psi_5^{(\alpha)}$ in different colours for $0 \leq x \leq 40$ and $\alpha = 0, 2$. Why do these functions constitute an orthonormal set under the inner product definition given in Eq. (3.12)? Verify orthonormality using Matlab.

4.3 Summary

So far you have

- generated a set of orthogonal polynomials based on monomials;
- scaled the polynomials to make them orthonormal;
- shown that they are *very* closely linked to a class of Laguerre polynomials;
- made a small adjustment to your original definition of monomials such that your set of orthogonal polynomials is in fact the set $\{L_n^{(0)}(x)\}$;
- constructed a set of orthonormal Laguerre basis functions from Laguerre polynomials and verified their orthonormal properties.

In the next stage of the project, you will construct mathematical models for various data sets using the Laguerre basis functions. You will use synthesised data in the next section in order to ensure your method is working correctly, and so that you can explore the importance of *thinking carefully* about which set of basis functions to use for a given data set (with respect to the choice of α).

IMPORTANT!

In what follows it may be best to switch to using the Laguerre functions $\psi_n^{(\alpha)}$ as defined.

Then if chaos breaks out, it is shared by all!

5 Fitting to Synthesised Data

In this section, you will use the Laguerre basis set to construct mathematical models of observed, synthesised data. First, we embark upon a brief discussion of the relationship between the method of least squares and orthogonal basis functions.

5.1 Least squares and orthogonal basis functions

If you want to approximate a continuous function $f_o(x)$ using an orthogonal basis set (Fourier, Laguerre, Hermite, whatever), we have seen that you write down the series

$$f(x) = a_0\psi_0(x) + a_1\psi_1(x) + \dots \tag{5.1}$$

and find the coefficients using

$$a_n = \langle\psi_n, f_o\rangle \Big/ \langle\psi_n, \psi_n\rangle \ . \tag{5.2}$$

In the case of the Laguerre functions considered in the last section, because the functions are orthonormal, the denominator is unity, and so

$$a_n = \langle\psi_n, f_o\rangle = \int_0^\infty \psi_n(x) f_o(x)dx \ . \tag{5.3}$$

But what has not been shown is that the coefficients obtained this way generate a least-squares fit to $f_o(x)$. Remembering that $f_o(x)$ is continuous, the least squares cost is (we drop the (x) from line 2 onwards)

$$\begin{aligned} C &= \int_0^\infty \big(f(x) - f_o(x)\big)^2 dx & (5.4)\\ &= \int_0^\infty f^2\,dx - 2\int_0^\infty f\,f_o\,dx + \int_0^\infty f_o^2\,dx & (5.5)\\ &= \big(a_0^2 + a_1^2 + \dots\big) - 2\int_0^\infty f\,f_o\,dx + K & (5.6)\end{aligned}$$

where K is a constant. But the a_n coefficients are the parameters in this model, and now recall that the general solution is to solve the simultaneous equations $\partial C/\partial a_0 = 0$, $\partial C/\partial a_1 = 0$, and so on. Writing out $f = a_0\psi_0(x) + a_1\psi_1(x) + \dots$ in the middle term, and differentiating, we obtain the equations

$$\partial C/\partial a_0 \;=\; 2a_0 - 2\int_0^\infty \psi_0\, f_o\, dx = 0 \tag{5.7}$$

$$\partial C/\partial a_1 \;=\; 2a_1 - 2\int_0^\infty \psi_1\, f_o\, dx = 0 \tag{5.8}$$

and so on. Each equation is independent and gives the general result

$$a_n = \int_0^\infty \psi_n(x) f_o(x) dx. \tag{5.9}$$

But this is exactly what we wrote in Eq. (5.3)! So we see both that the approximation is a least-squares approximation, and that the coefficients are independent. Note that this result follows for *continuous* functions $f_0(x)$; things are a little different when the function $f_o(x)$ is a discrete set of observed data points.

5.2 Generate data

The next step is to test our fitting technique on a well-behaved, simple data set. To do this, we will generate random data drawn from from a complex normal distribution and obtain an approximation for the probability density function of the *squared magnitude* of this data.

Open your B1/Matlab/FittingData directory. You will find the following code for generating the data. Examine this code, paying attention to functions such as `randn` and `hist`. The `mu` and `sigma2` input parameters can be used to alter the mean and variance of the underlying Gaussian samples. The `nsamp` and `nbins` parameters specify, respectively, the total number of randomly generated samples and the number of data points that are synthesised from this data. Notice the line in the code that normalises the data. This is important since the data should approximate a probability density function, and thus the integral over the support should be one.

```
%EXP_DATA generates exponentially distributed data,
% normalises it, and plots a histogram, which
% is an approximation to the density function
% associated with the underlying distribution.
%
% The histogram data and the bin locations are
% returned as vectors.
%
% input
%
```

```
%   sigma2      scales the Gaussian data (scalar)
%   mu          offsets the Gaussian data (cplx scalar)
%   nsamp       number of data samples (scalar)
%   nbins       number of bins in the histogram (scalar)
%
% output
%
%   fo          histogram points (vector)
%   x           histogram bin locations (vector)
%
% jpc 21/10/13
function [ fo, x ] = exp_data( sigma2, mu, nsamp, nbins )

% Synthesise complex normal data
%   mean = 0
%   variance per dimension = sigma2/2
W = sqrt(sigma2/2)*(randn(1,nsamp) + 1i*randn(1,nsamp))...
    + mu;

% Calculate the power
%   This should be exponentially distributed.
%   The mean should be approx. sigma2
X = abs(W).^2;

% Generate a histogram with nbins
%   The elements of fo are the number
%    of samples of X that fall in each bin.
%   The vector x corresponds to the
%    bin locations.
[fo,x] = hist(X,nbins);     % histogram
fo = fo/trapz(x,fo);        % normalise; fo is a density func

% Plot the data and save
ystr = sprintf('Exponentially distributed data (mean = %g)',sigma2);
figure
set(gca,'FontSize',18);
plot(x,fo,'kx','LineWidth',1.2)
xlabel('x');
ylabel(ystr);
grid on
print('-depsc',sprintf('exp_data_%g.eps',sigma2));
```

Task:

- Run the code for different input values to explore how it works. Include any noteworthy observations in your report.

5.3 Fitting to data

Complete the following task:

Task:

- Construct a Matlab function that creates a Laguerre fit to a supplied set of input data. The function should support the following as inputs:
  1. a vector of data values,
  2. a vector of 'x' values defining the locations of those data points,
  3. a scalar defining the order of the fit,
  4. and a scalar defining α.
- Run the code in Code Box A below. What does the data in `fo` look like? Is it decaying/increasing? How quickly (describe qualitatively)?

```
% Code Box A
>> sigma2 = 2; mu = 0;
>> nsamp = 1e4; nbins = 100;
>> [fo, x] = exp_data(sigma2, mu, nsamp, nbins);
```

Now we can construct a fit to the set of data generated by the `exp_data` function given above. But first, it is useful to have an idea of what we expect our fit to look like.

5.3.1 Hypothesis

Let's delve into the mathematics a bit. Our code for synthesising the data basically generates samples of random variables and constructs an empirical density function. For the scenario given above, `sigma2 = 2` and `mu = 0`, which corresponds to the random variables Y and Z each being (real) Gaussian distributed with zero mean and unit variance. Now define the random variable

$$W = Y + iZ \tag{5.10}$$

where $i = \sqrt{-1}$. This variate is complex Gaussian distributed. Squaring the magnitude of W produces

$$X = |W|^2 = Y^2 + Z^2. \tag{5.11}$$

You should remember from the A1 paper on probability and statistics that the sum of the squares of Gaussian random variables, each having a mean of zero and all having identical variances, is chi-squared distributed, so X is a chi-squared random variable with two degrees of freedom, or in even simpler terms, it is an *exponentially* distributed random variable with an expected value of 2. Thus, we expect that a fit of

$$f(x) = \frac{1}{2}e^{-x/2} \tag{5.12}$$

will accurately model the density of the data stored in `fo`. This is a nice result since it gives us something to compare with when we construct a fit based on Laguerre functions.

It is also worth noting that a basic parameterised fit can be constructed by simply computing the sample mean of the data ξ and setting

$$f(x) = \frac{1}{\xi}e^{-x/\xi}. \tag{5.13}$$

5.3.2 Laguerre fit

From the discussion above, it should be relatively clear that we want to choose an orthonormal basis that will result in an exponential fit to our data. Observing the general definition of Laguerre polynomials/functions, it should be obvious that we want to set $\alpha = 0$ and only consider the 0th order fit. (Why?)

Task:

- Construct a 0th order Laguerre fit to the data generated in the previous task. What is the coefficient a_0? Is this what you expected? Write down the equation for this fit.
- Estimate the mean of the data by executing the code in Code Box B below. What is this code doing? Why does this estimate the mean?
- Construct a parameterised fit to the data in `fo` using the estimated mean and Eq. (5.13).
- Compare the parameterised fit and the Laguerre fit by plotting them on the same graph as the data `fo`. Which is better? Quantify the 'goodness of fit' by estimating the error $\int_0^\infty (f(x) - f_o(x))^2 dx$.

```
% Code Box B
>> estimated_mean = trapz(x,x.*fo)
```

5.4 Another example

The contrived example that we treated above turned out to rather nicely align with the first order Laguerre function with parameter $\alpha = 0$. Such good fortune rarely occurs in practice. Sometimes, the structure of the data is such that a higher order expansion is required in order to accurately represent the underlying data. Let us consider an example.

Task:

- Run the code in Code Box C below.
- Generate Laguerre fits derived from the basis sets corresponding to $\alpha = 0$ and $\alpha = 2$. Use expansions of different orders up to 5.
- Plot your results and discuss. Which α value is better for a given order? Why? How could you have predicted this?

```
% Code Box C
>> sigma2 = 1/3; mu = 1 + sqrt(-1);
>> nsamp = 1e4; nbins = 100;
>> [fo, x] = exp_data(sigma2, mu, nsamp, nbins);
```

Some of my results can be seen on the next page.

5.5 Scaling and offsetting

We now come to the tricky matters of distance scaling and distance offsets.

The Laguerre polynomials as defined above are *canonical* in the sense that they are scaled and centred such that the squared norm of $L_0^{(\alpha)}(x)$ (i.e., the inner product with itself) is one. However, data in real life is seldom this well behaved! When faced with such an issue, you can take one of two courses of action: 1) ignore the anomaly and generate a Laguerre fit as if everything were just fine (a.k.a. the 'head in the sand' approach), or 2) try to condition the observed data or your chosen basis such that the fit is more easily achieved. Only choose option 1 if you feel lucky.

Let's focus on option 2. Suppose your function of observations is defined as

$$\hat{f}_o(x) = \mathcal{C}_o((x - \xi)/\sigma) \tag{5.14}$$

where ξ and σ are offset and scale parameters, respectively, and $\mathcal{C}_o$ denotes some canonical function. By writing this equation out, we can already see that one solution

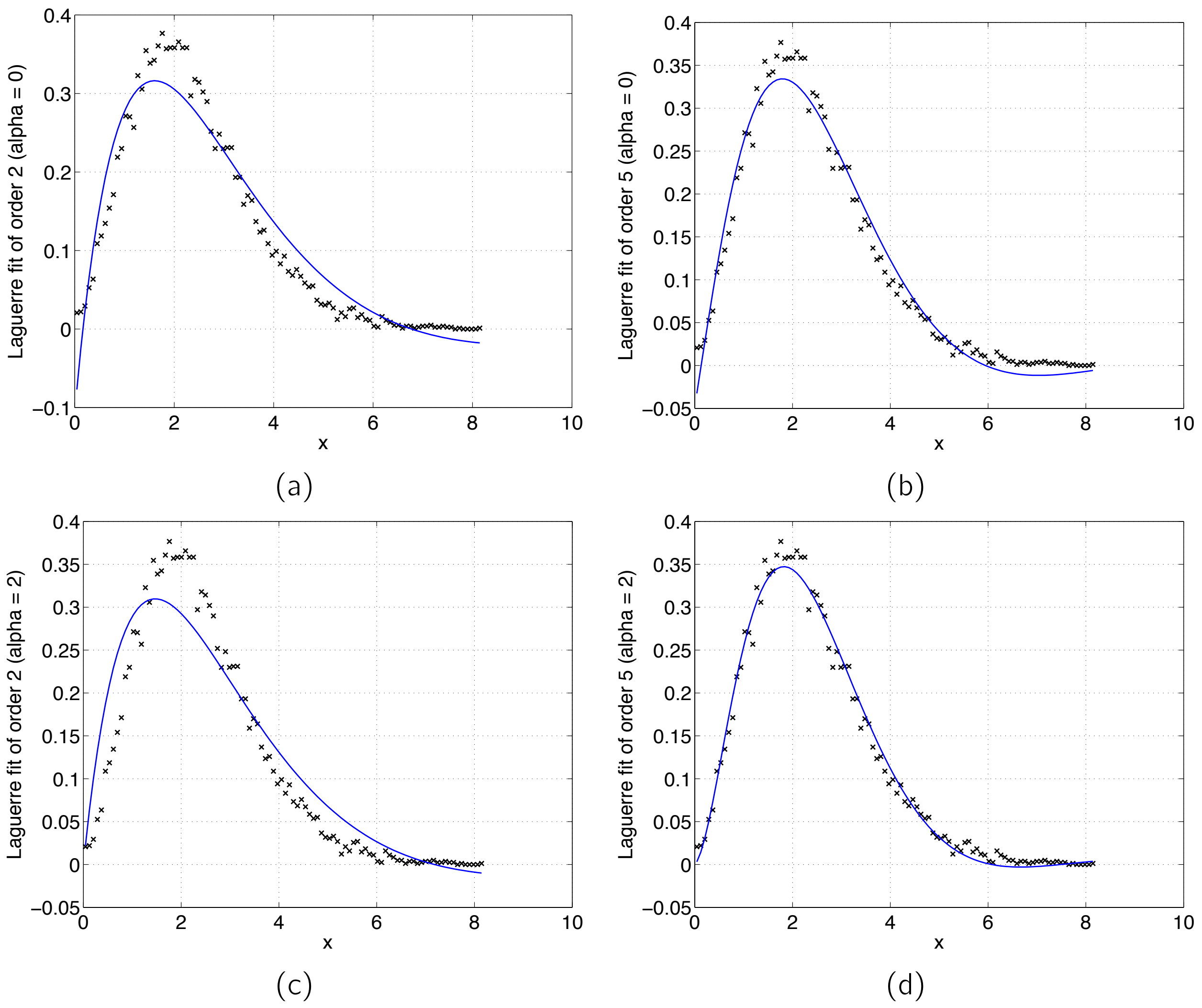


Figure 5.1: Laguerre fits: (a) $\alpha = 0$, 2nd order; (b) $\alpha = 0$, 5th order; (c) $\alpha = 2$, 2nd order; (d) $\alpha = 2$, 5th order;.

when faced with data that is scaled and offset like this is simply to redefine the domain of the data by the offset/scaled domain. Another solution would be to construct shifted and scaled versions of the basis functions used to fit to the data.

An example of the first solution — i.e., the method of redefining the domain of the data — goes something like this. If the locations of the observed data are defined by the points $\{x_i\}$, then we make the substitution

$$x_i \leftarrow \frac{x_i - \xi}{\sigma}. \tag{5.15}$$

We then fit the data in the usual way (i.e., we calculate the a_n coefficients) using the new x_i values and transform back when we want to display the results in the original coordinate system. The resulting basis expansion will be given by

$$f(x) = a_0\psi_0((x-\xi)/\sigma) + a_1\psi_1((x-\xi)/\sigma) + \ldots \tag{5.16}$$

Task:

- Run the code in Code Box D below.
- Generate Laguerre fits derived from the basis set corresponding to $\alpha = 0$. Use expansions of different orders up to 5.
- Plot your results. Are any of these fits suitable? Deduce whether or not you would be able to obtain a 'good' fit for this data by using some arbitrary order of expansion; explain your reasoning. (**Hint:** Pay particular attention to the origin.)
- Now rescale the location data by making the assignment shown in Code Box E below, and generate Laguerre fits as you did above using the scaled location vector `xscaled` rather than `x`. Why is there a mystical factor of 2 included in the assignment? (Try the assignment with and without the prefactor of 2 to see what happens.)
- Plot your results using the original location values stored in `x` and discuss your findings.
- Write down the equation for the new 0th order fit. How does this compare to a parameterised fit vis-à-vis Eq. (5.13)?

```
% Code Box D
>> sigma2 = 6; mu = 0;
>> nsamp = 1e4; nbins = 100;
>> [fo, x] = exp_data(sigma2, mu, nsamp, nbins);
```

```
% Code Box E
>> xscaled = 2*x/sigma2;
```

5.6 Summary

In this section, you

- explored the link between orthogonal functions and the method of least squares optimisation;
- generated empirical distributions of randomly generated data;
- wrote a Matlab function that creates a Laguerre fit for a set of data;

- explored the role that α plays in obtaining a suitable fit;
- studied solutions for treating scaled and offset data.

6 Fitting to a Real Data Set

In this final section, you will fit a mathematical model to a real data set[1] in order to examine the statistical behaviour of a wireless communication channel. You have acquired all of the skills necessary for this undertaking in the previous sections and in your B1 lectures.

Your (hypothetical) group leader will be keen to understand what methodology you adopt as well as why you adopt it. Be sure to include this information in your report. You are of course free to rewrite code as you wish.

On with the task ...

Task:

- Navigate to your B1/Matlab/FittingData folder and open the Matlab file `fit_measured_data.m`. The code can be seen below.
- Study the code and complete it.
- In your report, provide a critical analysis of your methodology, referring to key sections of your code as well as the mathematical background where appropriate. Be concise, but thorough.

```
%FIT_MEASURED_DATA generates a
% Laguerre fit to a set of measured
% channel data.  The fit mathematically
% describes the probability density
% of the attenuation (squared) of a
% measured wireless channel.
%
% jpc 23/10/13
function [ a ] = fit_measured_data( maxorder, alpha )

% Load the measured data.
% Two vectors will appear
% in the workspace:
%   xi is a vector of data point
%     locations;
```

[1]The data used in this section was obtained by Vit Sipal and David Edwards of the Oxford Communications Research Group.

```
%   fo is a vector of data points.
load('measured_data.mat')

% Plot the data.
% This will help you to
% visualisation what the fit
% should look like.
ystr = sprintf('Density data points');
figure
set(gca,'FontSize',18);
plot(xi,fo,'kx','LineWidth',1.2)
xlabel('x');
ylabel(ystr);
title('Histogram points for measured data');
grid on

%%% Your code here %%%
```

In order to give you something to aim for, I was able to obtain the following fit using only a 2nd order basis expansion. As you can see, the error is quite small (approximately 2.1×10^{-3}) even for this simple model. Be sure to quantify the error of your design in your report.

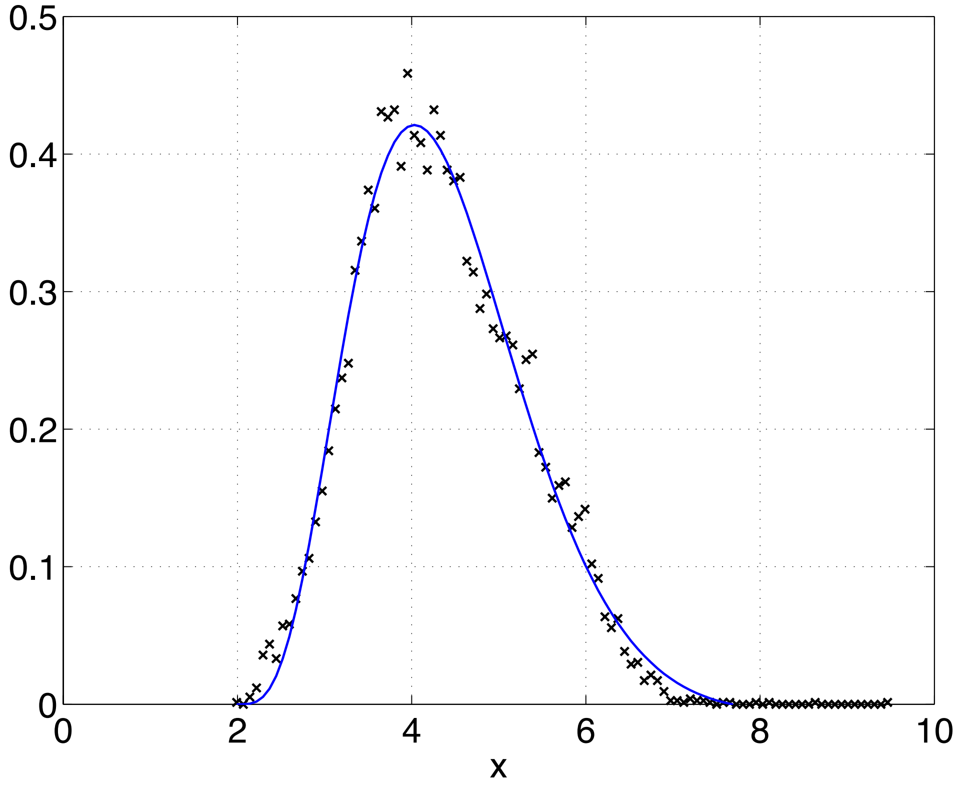


Figure 6.1: A 2nd order Laguerre fit to the measured data.

6.1 Final thoughts

This concludes the project. When writing your report, pay close attention to the questions asked in each of the tasks. Remember to provide a clear but concise account of your methodology, justifying your assumptions and approach as you see fit.

As a final thought, now that you have generated an appropriate fit to the measured data, what conclusions can you draw about the distribution of the scattering process associated with the Wi-Fi received signal power data?