Ateneo de Manila University

Ma195a15 Final project

# The Use of Chebyshev Polynomials for Approximating Functions

*Author:*
Wilhansen Joseph B. Li

*Supervisor:*
Dr. Roden David

September 27, 2010

# Chapter 1

# Introduction

## 1.1 Overview

Function approximation is just one of the many terms used to describe the process of compressing a large amount of data to a compact representation that can be used to:

1. Interpolate values between two data points,
2. Predict values outside of the subdomain given by the data set and
3. Create measures that can be used to describe the data

The broad term for this is "modeling". Other terms that do the almost the same thing include "statistical modeling", "data fitting", "data modeling", "parameter estimation", "series transforms", etc. Although they differ slightly in semantics, they generally follow the general idea. For example, statistical modeling may focus more on predicting data and describing them rather than interpolating values while function approximation is more interested in interpolating values between points given by the data set and doesn't really care about prediction. Also, their purposes might be different like how statistical measures are used to form conclusions while function approximations might be used to implement trancendental functions on a machine that can only perform arithmetic or how series transformations are used in signal processing to compress audio-visual data.

How modeling is done largely depends on the data set; no single model fits all kinds of data. For example, bernoulli events might be better modeled using the normal distribution but the normal distribution is not used to model the levels of a sound wave over time as the Fourier series or Haar wavelets would better model it.

The interest in this paper is the use of Chebyshev polynomials to approximate functions. Furthermore, various classes of mathematical functions will be analyzed in order to conclude which kinds of functions could best be modeled by Chebyshev polynomials. Scilab[1] source code will provided.

## 1.2 Vector Space of Functions

A vector space $\mathcal{F}$ can be created where each vector is a function; this is because functions have linearity: if $f(x), g(x) \in \mathcal{F}$ and $a, b \in \mathbb{R}$ then $(af + bg)(x) = af(x) + bg(x)$. Moreover, a set $\mathcal{G} \subseteq \mathcal{F}$ which restricts $\mathcal{F}$ to functions defined over a fixed interval $[a, b]$ is a subspace because domains of functions are not affected by linear combinations of functions.

Each vector in $\mathcal{F}$ has an uncountably infinite number of elements (i.e. the dimension of $\mathcal{F}$ is infinite). Consider $f(x) \in \mathcal{F}$ that's defined over $[0, 1]$; its elements, $\{a_0, a_1, \ldots, a_n\}$ are then: $a_0 = f(0), a_1 = f(1/n), a_2 = f(2/n), \ldots, a_n = f(1)$. However, because $f(x)$ is defined over a connected interval in $\mathbb{R}$, there is an infinite number of points in any interval so $n \to \infty$. Thus, $\mathcal{F}$ actually has an infinite dimension.

### 1.2.1 Magnitude of a function

The magnitude of a vector can be described by the square root of the summation of the square of its elements. For a function $f(x) \in \mathcal{F}$ which is defined over a domain $D = [a, b]$, we can define the magnitude as a Riemann sum which results in an integral:

$$\|f(x)\| = \lim_{n\to\infty} \sqrt{\sum_{i=0}^{n} \left( f\left(a + i\frac{b-a}{n}\right)\right)^2 \frac{b-a}{n}} = \lim_{\Delta x\to 0} \sqrt{\sum_{i=0}^{n} \left(f\left(a + i\Delta x\right)\right)^2 \Delta x} = \sqrt{\int_D (f(x))^2 dx} \quad (1.1)$$

Moreover, it can be confirmed the properties of vector spaces carry over:

$$\begin{aligned} \|cf(x)\| &= c\|f(x)\| && \text{(Homogeneity of degree 1)} \\ \|f(x) + g(x)\| &\leq \|f(x)\| + \|g(x)\| && \text{(Triangle inequality)} \\ \|f(x)\| &\geq 0 \text{ with } \|f(x)\| = 0 \iff f(x) = 0 && \text{(Positive Definiteness)} \end{aligned}$$

For the magnitudes of elements in $\mathcal{F}$ to be a field, we require that all magnitudes be finite (i.e. the integrals must converge). Moreover, elements in such set still forms a vector space since any linear combination of two convergent integrals is still convergent.

### 1.2.2 Inner Product of functions

An inner product space can be defined over $\mathcal{F}$. Just as how the magnitude of a function is defined by integration, the same will be done. The inner product is described to be the sum of the products of components between two vectors. If $f(x), g(x), h(x) \in \mathcal{F}$ are defined over $D$, then the inner product $\langle f, g\rangle$ is:

$$\langle f, g\rangle = \int_D f(x)g(x)dx \quad (1.2)$$

Note that properties of inner product spaces carry over:

$$\begin{aligned} \langle f, f\rangle &= \int_D f(x)f(x)dx = \int_D (f(x))^2 dx = \|f(x)\|^2 \\ \langle f, g\rangle &= \int_D f(x)g(x)dx = \int_D g(x)f(x)dx = \langle g, f\rangle \\ \langle cf, g\rangle &= \int_D cf(x)g(x)dx = c\int_D f(x)g(x)dx = c\langle f, g\rangle \\ \langle f + g, h\rangle &= \int_D (f(x) + g(x))h(x)dx = \int_D f(x)h(x)dx + \int_D g(x)h(x)dx = \langle f, h\rangle + \langle g, h\rangle \\ \langle f, g\rangle &\leq \|f(x)\|\|g(x)\| \qquad \text{(Cauchy-Schwarz Inequality)} \end{aligned}$$

Carrying over the definition in linear algebra, two functions $f(x), g(x)$ defined over $D$ is orthogonal if:

$$\langle f, g\rangle = \int_D f(x)g(x)dx = 0$$

### 1.2.3 Projection

The projection of the function $g$ onto $f$ is defined by

$$P_f(g) = \frac{\langle g, f\rangle}{\langle f, f\rangle}. \quad (1.3)$$

This gives a measure on how "close" the function $g$ is to $f$. The higher the value, the closer it is. Scaling the function $f(x)$ by $P_f(g)$, it will give the scaled $f(x)$ which best matches $g(x)$ (i.e. the least squares fit). That is, if $c \in \mathbb{R}$ then:

$$\min \|cf - g\|^2 = \|P_f(g)f - g\|^2 \quad (1.4)$$

The proof is as follows:

$$\begin{aligned}
\min \|cf - g\|^2 &\iff \frac{d}{dc}\int_D (cf(x) - g(x))^2 dx = 0 \\
\frac{d}{dc}\int_D (cf(x) - g(x))^2 dx &= \frac{d}{dc}\int_D \left[c^2 f^2(x) - 2cf(x)g(x) + g^2(x)\right] dx \\
&= \frac{d}{dc}\left[c^2 \int_D f^2(x)dx - 2c\int_D f(x)g(x)dx + \int_D g^2(x)dx\right] \\
&= 2c\int_D f^2(x)dx - 2\int_D f(x)g(x)dx = 0 \\
c &= \frac{\int_D f(x)g(x)dx}{\int_D f^2(x)dx} = \frac{\langle g, f\rangle}{\langle f, f\rangle} = P_f(g) \qquad \blacksquare
\end{aligned}$$

## 1.3 Orthogonal Basis Functions

Basis functions are basically a (possibly infinite) set of functions that can be used to approximate a function via a linear combination. That is, given a function $f(x)$, there exists numbers $\{c_1, c_2, \ldots, c_n\}$, called *coefficients*, over a field (usually $\mathbb{R}$) for a set of basis functions $\{T_1(x), T_2(x), \ldots, T_n(x)\}$ such that:

$$f(x) \approx c_1 T_1(x) + c_2 T_2(x) + \cdots + c_n T_n(x) = \sum_i c_i T_i(x) \tag{1.5}$$

If the set of basis functions is infinite, the function can still be approximated by a finite number of elements in that set without exceeding a certain error threshold.

If the basis functions $\{T_0(x), T_1(x), \ldots, T_n(x)\}$ over the domain $D$ is *orthogonal* then

$$\int_D T_i(x)T_j(x)w(x)dx = \begin{cases} 0 & i \neq j \\ \lambda_i \neq 0 & i = j \end{cases}$$

where $w(x)$ is the weight function that depends on the nature of the basis functions. Note that in this case, the inner product $\langle f, g\rangle$ is redefined as:

$$\langle f, g\rangle = \int_D f(x)g(x)w(x)dx. \tag{1.6}$$

If $\lambda_i = 1$ for all $i$ then it can be said that the basis is *orthonormal*.

This is analogous to orthogonal basis in the inner product space $\mathbb{R}^n$ where the inner product of two basis elements results in a zero if they're not the same or some non-zero value if they're the same. Intuitively, orthogonality means that the functions do not "overlap" when being projected so a coefficient, $c_i$ can be adjusted without causing "side effects" to other coefficients.

The process of determining such values of $c_i$ that can best approximate the function is called projection as discussed in 1.2.3 with the slight modification that in some classes of functions, an additional weighing function $w(x)$ is multiplied to the two product of two other functions prior to integrating (as in (1.6)). If $T_i(x)$ is orthogonal, the process of determining $c_i$ simplifies to as follows:

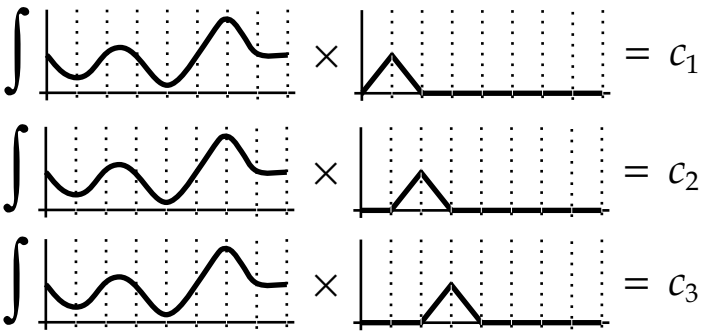
$$\begin{aligned}
f(x) &\approx c_1 T_1(x) + \cdots + c_n T_n(x) \\
f(x)T_i(x)w(x) &\approx c_1 T_1(x)T_i(x)w(x) + \cdots + c_i T_i(x)T_i(x)w(x) + \cdots + c_n T_n(x)T_i(x)w(x) \\
\int_D f(x)T_i(x)w(x)dx &\approx \int_D c_1 T_1(x)T_i(x)w(x)dx + \cdots + \int_D c_i T_i^2(x)w(x)dx + \cdots + \int_D c_n T_n(x)T_i(x)w(x)dx \\
\int_D f(x)T_i(x)w(x)dx &= 0 + \cdots + c_i\lambda_i + \cdots + 0
\end{aligned}$$

Thus:

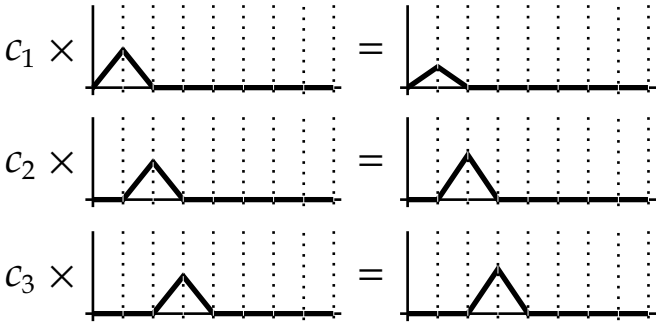
$$c_i = \frac{1}{\lambda_i} \int_D f(x) T_i(x) w(x) dx = \frac{1}{\langle T_i, T_i \rangle} \langle f, T_i \rangle = P_{T_i}(f). \tag{1.7}$$

Note that if the basis are not orthogonal, then there will be terms other than $c_i \lambda_i$ that will be non-zero causing some difficulties in determining $c_i$ because a system of linear equations will appear which requires another solving step.

To illustrate the projection process[1]:

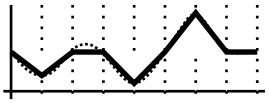



Scaling them we get




then finally summing the scaled functions:

$$\sum_i c_i T_i(x) =$$



[1] Illustrations from [3]

# Chapter 2

# Chebyshev Polynomials

## 2.1 Definition

Chebyshev polynomials arise as the solution to the Chebyshev differential equation:

$$(1-x^2)\frac{d^2y}{dx^2} - x\frac{dy}{dx} + n^2y = 0 \qquad n = 0, 1, 2, \ldots \tag{2.1}$$

let $x = \cos t$, the differential equation then becomes:

$$\begin{aligned}
(1-\cos^2 t)\left(\frac{d^2y}{dt^2}\left(\frac{dt}{dx}\right)^2 + \frac{dy}{dt}\frac{d^2t}{dx^2}\right) - \cos t\frac{dy}{dt}\frac{dt}{dx} + n^2y &= 0 \\
\sin^2 t\left(\frac{d^2y}{dt^2}\left(-\frac{1}{\sqrt{1-\cos^2 t}}\right)^2 + \frac{dy}{dt}\left(-\frac{\cos t}{(1-\cos^2 t)^{\frac{3}{2}}}\right)\right) - \cos t\frac{dy}{dt}\left(-\frac{1}{\sqrt{1-\cos^2 t}}\right) + n^2y &= 0 \\
\sin^2 t\left(\frac{d^2y}{dt^2}\left(-\frac{1}{\sin t}\right)^2 + \frac{dy}{dt}\left(-\frac{\cos t}{\sin^3 t}\right)\right) - \cos t\frac{dy}{dt}\left(-\frac{1}{\sin t}\right) + n^2y &= 0 \\
\sin^2 t\frac{1}{\sin^2 t}\frac{d^2y}{dt^2} - \sin^2 t\frac{\cos t}{\sin^3 t}\frac{dy}{dt} + \cos t\frac{1}{\sin t}\frac{dy}{dt} + n^2y &= 0 \\
\frac{d^2y}{dt^2} - \frac{\cos t}{\sin t}\frac{dy}{dt} + \frac{\cos t}{\sin t}\frac{dy}{dt} + n^2y &= 0 \\
\frac{d^2y}{dt^2} + n^2y &= 0
\end{aligned}$$

whose solution is given by [2]:

$$\begin{aligned}
y &= A\cos(nt) + B\sin(nt) \qquad |x| \le 1 \\
y &= A\cos(n\arccos x) + B\sin(n\arccos x) \qquad |x| \le 1 \\
y &= AT_n(x) + BU_n(x)
\end{aligned}$$

where $T_n(x)$ and $U_n(x)$ is defined as the Chebyshev polynomial of the first and second kind respectively with degree $n$. Both Chebyshev polynomials are in the domain $[-1, 1]$ and have their degree $n \in \mathbb{Z} \cup \{0\}$. There are two kinds of Chebyshev polynomials but the focus will be on the first kind (this will be implied whenever the kind is not specified).

The Chebyshev polynomial can be generated by the following recurrence relation [2]:

$$\begin{aligned}
T_0(x) &= 1 \\
T_1(x) &= x \\
T_{n+1}(x) &= 2xT_n(x) - T_{n-1}(x)
\end{aligned} \tag{2.2}$$

In Scilab, the command **chepol**(n,var) generates the Chebyshev polynomial of degree $n$. Alternatively, if the coefficient matrix of Chebyshev Polynomials $T_0(x)$ to $T_n(x)$ is desired, listing 2.1 is more useful.

Listing 2.1: Fast Generation of the Coefficient Matrix of Chebyshev Polynomials from $T_0(x)$ to $T_n(x)$

```
//Generates an (n+1) by (n+1) coefficient matrix of the Chebyshev polynomials
    from T_0 to T_n.
function T = che_coeff_gen(n)
  n = n + 1
  T = zeros(n,n)
  T(1,1) = 1
  if n < 2
    return
  end

  T(2,2) = 1

  for i = 3:n
    row = 2 * T(i-1,:)
    row = [0,row(1:(size(row,'c')-1))]
    T(i,:) = row - T(i - 2, :)
  end
endfunction
```

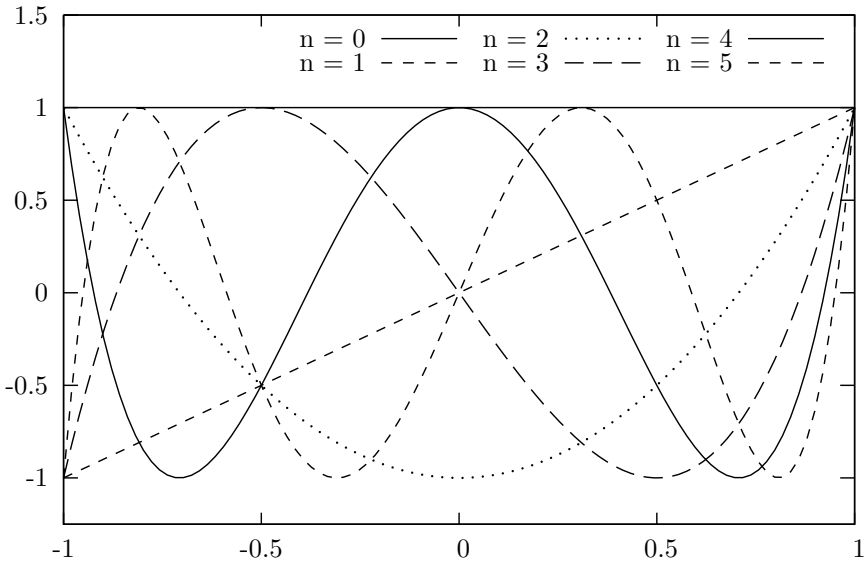



Figure 2.1: Graph of the first 6 Chebyshev Polynomials

A Chebyshev series of degree $n$ is the linear combination of Chebyshev polynomials from 0 to $n$. That is, if $S_n(x)$ is a Chebyshev series,

$$S_n(x) = \sum_{i=0}^{n} c_i T_i(x) \tag{2.3}$$

for some $c_i \in \mathbb{R}$. A smaller $n$ means that $S_n$ has a low degree while a larger $n$ means $S_n$ has a high degree.

### 2.1.1 Angle Domain (Angular Chebyshev)

The domain Chebyshev polynomial $T_n(x)$ can be transformed to the angle domain $T_n^\theta(t) = \cos(nt)$ with $t \in [0, \pi]$ which is similar to how the Chebyshev differential equation is simplified. This effectively transforms the Chebyshev polynomials into half of the Fourier series (the Fourier series includes the $\sin(x)$ functions[1]). Moreover, it should be noted that this transformation makes the distribution of points uniform. that is, while $T_n(x)$ has a non-unity weight function, $T_n^\theta(t)$ has no weight function (the weight function is 1) as seen in the next section.

The Chebyshev Polynomials in the angular domain may be referred as the Angular Chebyshev Polynomials but it will be more accurate to call them the Angular Chebyshev series instead since they are not polynomials anymore.

## 2.2 Orthogonality

The Chebyshev polynomial $T_n(x)$ are orthogonal with respect to the weight

$$w(x) = \frac{1}{\sqrt{1-x^2}}. \tag{2.4}$$

The angular Chebyshev series, however, do not need a weight function (i.e. $w(x) = 1$).

Orthogonality can be shown by integrating $T_n^\theta(x)$

$$\int_0^\pi T_m^\theta(t) T_n^\theta(t) dt = \int_0^\pi \cos(mt)\cos(nt)dt = \begin{cases} 0 & m \neq n \\ \pi & m = n = 0 \\ \frac{\pi}{2} & \text{otherwise} \end{cases} \tag{2.5}$$

Proof:

If $m = n = 0$:

$$\int_0^\pi \cos(mt)\cos(nt)dt = \int_0^\pi \cos(0)\cos(0)dt = \int_0^\pi dt = \pi.$$

If $m = n \neq 0$:

$$\begin{aligned} \int_0^\pi \cos(mt)\cos(nt)dt &= \int_0^\pi \frac{\cos((n+m)t) + \cos((n-m)t)}{2} dt \\ \int_0^\pi \frac{\cos((n+m)t) + \cos((n-m)t)}{2} dt &= \int_0^\pi \frac{\cos((n+m)t) + \cos(0)}{2} dt = \int_0^\pi \frac{\cos((n+m)t) + 1}{2} dt \\ &= \left[\frac{\sin((n+m)t)}{2(n+m)} + \frac{t}{2}\right]_0^\pi = \frac{\sin((n+m)\pi)}{2(n+m)} + \frac{\pi}{2} - \frac{\sin(0)}{2(n+m)} - \frac{0}{2} \end{aligned}$$

since $(n+m) \in \mathbb{Z}$, $\sin((n+m)\pi) = \sin(\pi) = 0$:

$$\int_0^\pi \cos(mt)\cos(nt)dt = \frac{0}{2(n+m)} + \frac{\pi}{2} = \frac{\pi}{2}.$$

[1] More information on the Fourier series is provided by [4]

If $n \neq m$:

$$\begin{aligned}\int_0^\pi \cos(mt)\cos(nt)dt &= \int_0^\pi \frac{\cos((n+m)t)+\cos((n-m)t)}{2}dt \\ &= \left[\frac{\sin((n+m)t)}{2(n+m)}+\frac{\sin((n-m)t)}{2(n-m)}\right]_0^\pi\end{aligned}$$

since $(n+m),(n-m) \in \mathbb{Z}$:

$$\int_0^\pi \cos(mt)\cos(nt)dt = 0. \qquad \blacksquare$$

Note that no weight function is needed. Transforming it to $T_n(x)$, $x = \cos t \Rightarrow -(1-x^2)^{-0.5}dx = dt$:

$$\int_{x(0)}^{x(\pi)} \frac{T_m(x)T_n(x)}{-\sqrt{1-x^2}}dx = \int_{-1}^{1} \frac{T_m(x)T_n(x)}{\sqrt{1-x^2}}dx = \int_{-1}^{1} T_m(x)T_n(x)w(x)dx = \begin{cases} 0 & m \neq n \\ \pi & m = n = 0 \\ \frac{\pi}{2} & \text{otherwise} \end{cases} \tag{2.6}$$

## 2.3 Expansion of $x^n$ in a Series of $T_n(x)$

The Chebyshev polynomial is defined by the recurrence relation:

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$$

Rewriting:

$$\begin{aligned} xT_n(x) &= \frac{1}{2}(T_{n+1}(x)+T_{n-1}(x)) \\ xT_1(x) &= x^2 \\ xT_0(x) &= T_1(x) = x \end{aligned} \tag{2.7}$$

This gives us a recurrence relation for obtaining the expansion of $x^n$ as a series of $T_n(x)$.

## 2.4 Rational Chebyshev Polynomials

Another variant of the Chebyshev polynomials, called Rational Chebyshev polynomials, is defined as follows:

$$R_n(x) = T_n\left(\frac{x-1}{x+1}\right) \tag{2.8}$$

with a weight function of:

$$w_R(x) = \frac{1}{(x+1)\sqrt{x}}. \tag{2.9}$$

All properties of $T_n(x)$ carry over, to $R_n(x)$.

## 2.5 Function Approximation

From 1.3, given a function $f(x)$ defined on $[-1, 1]$, there exist $\{c_0, c_1, \ldots, c_n\}$ such that

$$f(x) \approx \sum_i c_i T_i(x).$$

$c_i$ can be computed by

$$c_i = \frac{1}{\lambda_i} \int_{-1}^{1} f(x) T_i(x) w(x) dx \tag{2.10}$$

where

$$\lambda_i = \begin{cases} \pi & i = 0 \\ \dfrac{\pi}{2} & i = 1, 2, 3, \ldots \end{cases} \tag{2.11}$$

$T_i^\theta$ can be used if $f(x)$ is defined on $[0, \pi]$:

$$c_i^\theta = \frac{1}{\lambda_i} \int_{0}^{\pi} f(x) T_i^\theta(x) dx \tag{2.12}$$

# Chapter 3

# Computation

## 3.1 Montè Carlo Integration

When integrating functions, the black box model is assumed where no other information about the function is known other than the set out numerical outputs $\{y_0, y_1, \ldots, y_p\}$ given the inputs $\{x_0, x_1, \ldots, x_p\}$. In order to perform integration with this restriction, Montè Carlo integration has to be used.

Montè Carlo integration is based on the theory of probability[3]. A random variable is a mapping of any event (whether numerical or not) to a number. For example, a random variable for a coin toss $C(x)$ will return a 1 for a head and 0 for a tail. Similarly, any function $f(x)$ can be a random variable. A probability density function (pdf) $p(x)$ is any function with the following characteristics:

- $p(x) > 0 \quad \forall x \in D$
- $\int_D p(x)dx = 1.$

A pdf models how often a certain event appears, that is, how the events are distributed. Examples of these is the normal distribution (the "bell-shaped" curve), the uniform distribution and the binomial distribution. In statistics, $f(x) \sim p(x)$ means that the random variable $f(x)$ has the distribution of $p(x)$. The expected value of $f(x)$, written as $E[f(x)]$ is the average of the random variable which is defined as:

$$E[f(x)] = \int_D f(x)p(x)dx. \tag{3.1}$$

While this calculates the average value of the random variable, it is also possible to take the mean of a large number of random samples called $\hat{\mu}$ (an "estimator" for the actual mean $\mu$) of the random variable. This is shown to be consistent by the Law of Large Numbers thus converges to the actual expected value of the random variable for sufficiently many samples:

$$E[f(x)] \approx \hat{\mu} = \frac{1}{p+1}\sum_{i=0}^{p} f(x_i) \tag{3.2}$$

Thus, the integral of $f(x)$ can be approximated by:

$$\int_D f(x)dx = \int_D \frac{f(x)p(x)}{p(x)}dx = E\left[\frac{f(x)}{p(x)}\right] \approx \frac{1}{p+1}\sum_{i=0}^{p}\frac{f(x_i)}{p(x_i)}.$$

If we assume that the distribution is uniform over the interval $D = [a, b]$, we let $p(x) = (b-a)^{-1}$ thus:

$$\frac{1}{p+1}\sum_{i=0}^{p}\frac{f(x_i)}{p(x_i)} = \frac{1}{p+1}\sum_{i=0}^{p}(b-a)f(x_i) = \frac{b-a}{p+1}\sum_{i=0}^{p} f(x_i).$$

Here, $b-a$ is basically the "volume" of the domain of integration. For a one-dimensional case, it's the length of the interval.

Summarizing everything:

$$\int_a^b f(x)dx \approx \frac{b-a}{p+1}\sum_{i=0}^{p} f(x_i) \qquad \text{(The Montè Carlo Integrator)} \tag{3.3}$$

Listing 3.1: Montè Carlo Integration

```
//Integrates a function using the Monte-Carlo method.
//
//@param f The function to integrate
//@param x_arr The samples to evaluate
//@param volume The volume of integration
function y = mc(f, x_arr, volume)
  n = size(x_arr, '*')
  y = 0
  for i = 1:n
    y = y + f(x_arr(i))
  end
  y = (y * volume)/n
endfunction
```

Combining this with section 2.5, the projection over the Chebyshev polynomials can be approximated by:

$$c_n = \frac{1}{\lambda_n}\int_{-1}^{1} f(x)T_n(x)w(x)dx \approx \frac{2}{\lambda_n(p+1)}\sum_{i=0}^{p} f(x_i)T_n(x_i)w(x_i). \tag{3.4}$$

Listing 3.2: Generate a Vector of Chebyshev Polynomials from degree 0 to $n$

```
//Generates an (n+1) vertical vector containing the Chebyshev polynomials from
    T_0 to T_n.
function T = gen_che_arr(n)
  T = []
  for i = 1:n+1
    T = [T;chepol(i-1, 'x')]
  end
endfunction
```

Listing 3.3: Project a Series of Points to the Chebyshev Polynomials

```
//Projects a dataset to the Chebyshev Polynomials
//
//@param xs The x-values of the dataset.
//@param ys The y=values of the dataset.
//@param che_arr A vertical vector of Chebyshev Polynomials generated by
    gen_che_arr.
function a = che_proj(xs,ys,che_arr)
  function y = w(x)
    y = 1/sqrt(1-x*x)
  endfunction

  a = []
  n = size(che_arr, 'r')
  v = linspace(1, size(xs, '*'), size(xs, '*'))
```

```
  for k = 1:n

    function y = fun(j)
      y = ys(j)*horner(che_arr(k),xs(j))*w(xs(j))
    endfunction

    if k == 1
      a = [a, mc(fun, v, 2)/%pi]
    else
      a = [a, mc(fun, v, 2) * 2 /%pi]
    end

  end
endfunction
```

Note that there will be problems if the sample points land on $-1$ or 1 as this will cause divide-by-zero errors in $w(x)$ even if the improper integral $T_n(x)w(x)dx$ converges. A solution to this is to simply remove sample points very close to $-1$ or 1.

For $T_n^\theta$:

$$c_n = \frac{1}{\lambda_n}\int_0^\pi f(x)T_n^\theta(x)dx \approx \frac{\pi}{\lambda_n(p+1)}\sum_{i=0}^{p} f(x_i)T_n^\theta(x_i) \tag{3.5}$$

Listing 3.4: Project a Series of Points to the Angular Chebyshev Series

```
//Projects a dataset to a the angular Chebyshev series up to degree n
//
//@param xs The x-values of the dataset
//@param ys The y-values of the dataset
//@param n The degree of the angular Chebyshev series (i.e. from T_0^\theta(x)
    ,..., T_n^\theta(x))
function a = che_angle_proj(xs, ys, n)
  funcprot(0)
  a = []
  v = linspace(1,size(xs, '*'), size(xs, '*'))

  for k = 0:n

    function y = che_c(i)
      y = ys(i) * cos(k*xs(i))
    endfunction

    if k == 0
      a = [a, mc(che_c, v, 1)]
    else
      a = [a, mc(che_c, v, 2)]
    end

  end
endfunction
```

Since the Angular Chebyshev series are defined at all points, there are no numerical problems so the sample points on $T_n^\theta(x)$ could straightforwardly be evaluated.

## 3.2 Evaluation

Given the coefficients of the basis functions, the actual function can be approximated as described in 1.3.

Listing 3.5: Evaluate the Chebyshev Polynomials

```
//Evaluate a Chebyshev polynomial
//
//@param che_arr The vertical vector of Chebyshev polynomials generated by
    gen_che_arr.
//@param a_arr The horizontal vector of the coefficients of the Chebyshev
    polynomials generated by che_proj.
//@param x The point to evaluate.
function y = che_eval(che_arr, a_arr, x)
  y = 0
  n = size(che_arr, 'r')
  for i = 1:n
    y = y + a_arr(i) * horner(che_arr(i), x)
  end
endfunction
```

Listing 3.6: Evaluate the Angular Chebyshev Series

```
//Evaluate an angular Chebyshev series
//
//@param a The angular Chebyshev coefficients generated by che_angle_proj (a
    horizontal vector).
//@param x The point to evaluate
function y = che_angle_eval(a, x)
  n = size(a, '*')
  y = 0
  for i = 0:n-1
    y = y + a(i + 1) * cos(i * x)
  end
endfunction
```

## 3.3 Rational Polynomial Approximation

As a benchmark, the function $f(x)$ will also be approximated using rational polynomials. A rational polynomial is the ratio of two polynomial functions. Thus:

$$f(x) \approx \frac{g(x)}{h(x)} = \frac{a_0x^0 + a_1x^1 + \cdots + a_mx^m}{b_0x^0 + b_1x^1 + \cdots + b_nx^n} = \frac{\sum_i^m a_ix^i}{\sum_j^n b_jx^j}$$

Transposing the summation:

$$\sum_{j=0}^{n} f(x)b_jx^j = \sum_{i=0}^{m} a_ix^i$$

$$-\sum_{i=0}^{m} a_ix^i + \sum_{j=0}^{n} f(x)b_jx^j = 0. \tag{3.6}$$

Which can be seen as a dot product of two vectors:

$$\begin{bmatrix} -x^0 & -x^1 & \dots & -x^m & f(x)x^0 & f(x)x^1 & \dots & f(x)x^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_m \\ b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix} = 0$$

Suppose there are $p$ data points $(x_i, y_i)$ where $y_i = f(x_i)$. The equation now expands to:

$$\begin{bmatrix} -x_0^0 & -x_0^1 & \cdots & -x_0^m & y_0x_0^0 & y_0x_0^1 & \cdots & y_0x_0^n \\ -x_1^0 & -x_1^1 & \cdots & -x_1^m & y_1x_1^0 & y_1x_1^1 & \cdots & y_1x_1^n \\ -x_2^0 & -x_2^1 & \cdots & -x_2^m & y_2x_2^0 & y_2x_2^1 & \cdots & y_2x_2^n \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ -x_p^0 & -x_p^1 & \cdots & -x_p^m & y_px_p^0 & y_px_p^1 & \cdots & y_px_p^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_m \\ b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix} = 0.$$

However, this system will always have a trivial solution which is $b_0 = b_1 = \cdots = a_0 = a_1 = \cdots = 0$. To prevent this, fix $b_0 = 1$ which also prevents $h(x)$ from turning 0 $\forall x$. This transforms (3.6) into:

$$-\sum_{i=0}^{m} a_i x^i + \sum_{j=1}^{n} f(x) b_j x^j = -f(x) \tag{3.7}$$

Let:

$$A = \begin{bmatrix} -x_0^0 & -x_0^1 & \cdots & -x_0^m & y_0x_0^1 & y_0x_0^2 & \cdots & y_0x_0^n \\ -x_1^0 & -x_1^1 & \cdots & -x_1^m & y_1x_1^1 & y_1x_1^2 & \cdots & y_1x_1^n \\ -x_2^0 & -x_2^1 & \cdots & -x_2^m & y_2x_2^1 & y_2x_2^2 & \cdots & y_2x_2^n \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ -x_p^0 & -x_p^1 & \cdots & -x_p^m & y_px_p^1 & y_px_p^2 & \cdots & y_px_p^n \end{bmatrix} \quad x = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_m \\ b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \quad b = \begin{bmatrix} -y_0x_0^0 \\ -y_1x_1^0 \\ -y_2x_2^0 \\ \vdots \\ -y_px_p^0 \end{bmatrix} = \begin{bmatrix} -y_0 \\ -y_1 \\ -y_2 \\ \vdots \\ -y_p \end{bmatrix} \tag{3.8}$$

Thus the equation is now represented by $Ax = b$ where $x$ can be solved by the least-squares method (i.e. solving the system $A^TAx = A^Tb$).

Listing 3.7: Setup a Linear System for Rational Approximation

```
//Setup the system of equations for least-squares rational polynomial fitting.
//
//@param xs The x-values of the input.
//@param ys The y-values of the input (should be the same size as xs).
//@param num_degree The degree of the numerator of the approximated rational.
//@param den_degree The degree of the denominator of the approximated rational.
//
//@return A The coefficient matrix. (As in Ax = b)
//@return b The resultant vector. (As in Ax = b)
```

```
function [A, b] = rational_setup(xs, ys, numerator_degree, denominator_degree)
  m = max(0, numerator_degree) + 1
  n = max(0, denominator_degree) + 1
  p = size(xs, '*')

  A = zeros(p, n + m - 1)
  b = zeros(p, 1)

  for i = 1:p
    b(i) = -ys(i)
    A(i, 1) = -1
    x = xs(i)

    for j = 2:m
      A(i,j) = -x
      if j + m <= n + m
        A(i,j + m - 1) = ys(i) * x
      end
      x = x * xs(i)
    end
    for j = (2*m):(n + m - 1)
      A(i,j) = ys(i) * x
      x = x * xs(i)
    end
  end
endfunction
```

Listing 3.8: Approximate a Function by a Rational Polynomial

```
//Fit a dataset to a rational polynomial
//
//@param xs The x-values of the input.
//@param ys The y-values of the input (should be the same size as xs).
//@param num_degree The degree of the numerator of the approximated rational.
//@param den_degree The degree of the denominator of the approximated rational.
//
//@return R A rational polynomial (type 'rational') which best fits the dataset
    .
function R = rational_approx(xs, ys, num_degree, den_degree)
  [A, b] = rational_setup(xs, ys, num_degree, den_degree)
  x = A \ b
  num = poly(x(1:num_degree + 1), 'x', 'coeff')
  den = poly([1;x(num_degree + 2:num_degree + 1 + den_degree)], 'x', 'coeff')
  R = num/den
endfunction
```

## 3.4 Rational to Chebyshev Polynomial Conversion

Since Chebyshev Polynomials are polynomials, it is possible to express of $x^n$ with a series of Chebyshev polynomials to degree $n$. One method for directly calculating this is by the recurrence relation shown in 2.3. Another method for calculating all expansions from $x^0$ to $x^n$ given the Chebyshev Polynomials from $T_0(x)$ to $T_n(x)$ will be detailed here.

$T_i(x) = a_0x^0 + a_1x^1 + \cdots + a_ix^i$ which can be written as:

$$\begin{bmatrix} a_0 & a_1 & \cdots & a_i \end{bmatrix} \begin{bmatrix} x^0 \\ x^1 \\ \vdots \\ x^i \end{bmatrix} = \begin{bmatrix} T_i(x) \end{bmatrix}.$$

So the Chebyshev Polynomials with degrees from 0 to $n$ can be written as:

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ -1 & 0 & 2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \cdots & \cdots & \cdots & \cdots & 2^{n-1} \end{bmatrix} \begin{bmatrix} x^0 \\ x^1 \\ x^2 \\ \vdots \\ x^n \end{bmatrix} = \begin{bmatrix} T_0(x) \\ T_1(x) \\ T_2(x) \\ \vdots \\ T_n(x) \end{bmatrix}.$$

Let

$$P = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ -1 & 0 & 2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \cdots & \cdots & \cdots & \cdots & 2^{n-1} \end{bmatrix} \qquad x = \begin{bmatrix} x^0 \\ x^1 \\ x^2 \\ \vdots \\ x^n \end{bmatrix} \qquad T = \begin{bmatrix} T_0(x) \\ T_1(x) \\ T_2(x) \\ \vdots \\ T_n(x) \end{bmatrix}, \tag{3.9}$$

where $P$ is the coefficient matrix of the Chebyshev Polynomials $T_0(x)$ to $T_n(x)$ (which can be generated by listing 2.1); thus

$$Px = T.$$

Note that the $T_i(x)$ has a polynomial degree at most $i$ so the diagonals of $P$ are non-zero and all entries after $i$th column in row $i$ are zeroes. Moreover, if $i$ is odd, all even entries are zero and if $i$ is even, all odd entires are zero. Thus, $P$ is a sparse and lower-triangular matrix with non-zero trace which means that $P$ is non-singular. $P$ has an inverse (which is easily computed) so the expansion of $x^n$ in terms of the Chebyshev polynomials can be calculated by:

$$x = P^{-1}T. \tag{3.10}$$

This also provides us a way to express any polynomial function in terms of Chebyshev Polynomials. Given a polynomial function $f(x) = a_0x^0 + a_1x^1 + \cdots + a_nx^n$, a matrix $A$ can be constructed such that:

$$f(x) = Ax = \begin{bmatrix} a_0 & a_1 & \cdots & a_n \end{bmatrix} \begin{bmatrix} x^0 \\ x^1 \\ \vdots \\ x^n \end{bmatrix}.$$

$A$ is a row vector where the $i$th element corresponds to the coefficient of $x^i$. Premultiplying $A$ to (3.10):

$$f(x) = Ax = AP^{-1}T \tag{3.11}$$

Note that $A$ is a $1 \times (n+1)$ vector and $P$ is a $(n+1) \times (n+1)$ matrix so $AP^{-1}$ is a $1 \times (n+1)$ vector. Let

$$C = \begin{bmatrix} c_0 & c_1 & \cdots & c_n \end{bmatrix} = AP^{-1}. \tag{3.12}$$

Substituting this into (3.11),

$$f(x) = CT = c_0T_0(x) + c_1T_1(x) + \cdots + c_nT_n(x). \tag{3.13}$$

$C$ then contains the coefficients of the Chebyshev polynomials that correspond to the polynomial function. $C$ alternatively can be expressed in column vector form as follows:

$$\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix} = P^{-T}A^T = P^{-T} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} \tag{3.14}$$

which can now be arranged in the standard matrix form:

$$P^T \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix}. \tag{3.15}$$

If $f(x)$ is expressed as a rational function $g(x)/h(x)$, the procedure is to simply calculate the Chebyshev coefficients for $g(x)$ and $h(x)$ separately. Specifically, a coefficient matrix $P$ with degree $\max\{\deg(g), \deg(h)\}$ must be created and this matrix can be used for both $g(x)$ and $h(x)$. The row coefficient vectors $A_g$ and $A_h$ can then be created from the coefficients of the polynomials of $g(x)$ and $h(x)$ respectively. These matrices now may be plugged in to (3.12) to solve the Chebyshev coefficients of $g(x)$ and $h(x)$ separately. The Chebyshev coefficients of $g(x)$ and $h(x)$ is equivalently called the Chebyshev rational coefficients of $f(x)$ which is different from the rational Chebyshev in 2.4.

Listing 3.9: Convert Coefficients of Rational Polynomials to Chebyshev Rational Coefficients

```
//Converts rational coefficients to Chebyshev Rational coefficients.
//
//@param rational A list of rational (or just a rational) functions to convert.
//
//@return B A list of 2 by max(degree(numerator),degree(denominator)) matrices
    containing the converted coefficients. Note that even if the numerator and
    denominator vary in degree, the matrix will have the width of the max degree
    .
function B = rational_to_che(rational)
  if typeof(rational) == 'rational'
    rational = list(rational)
  end

  nCases = length(rational)
  maxDegree = 0
  for i = 1:nCases
    maxDegree = max(maxDegree, degree(rational(i).num), degree(rational(i).den)
        )
  end
  P = inv(che_coeff_gen(maxDegree))
  B = list()

  for i = 1:nCases
    num = [coeff(rational(i).num), zeros(1,maxDegree - degree(rational(i).num))
        ]
    den = [coeff(rational(i).den), zeros(1,maxDegree - degree(rational(i).den))
        ]
    B($ + 1) = [num * P; den * P]
  end
endfunction
```

Listing 3.10: Evaluation of Chebyshev Rationals

```
//Evaluates the Chebyshev Rational Polynomial at x
//
//@param che_arr An array of chebyshev polynomials generated by gen_che_arr.
//@param num_den_arr A 2 by size(che_arr, 'r') matrix containing the
    coefficients of the chebyshev polynomials.
//@param x The point of evaluation.
function y = che_rational_eval(che_arr, num_den_arr, x)
  num = 0
  den = 0

  n = size(che_arr, 'r')
  for i = 1:n
    z = horner(che_arr(i), x)
    num = num + num_den_arr(1,i) * z
    den = den + num_den_arr(2,i) * z
  end
  y = num / den
endfunction
```

# Chapter 4

# Analysis

## 4.1 Methodology

The process of evaluating the accuracy of Chebyshev polynomials involves:

1. Generating 30, 50 and 100 equally-spaced points on the interval $[0, \pi]$.
2. Using each set of points to approximate functions using the Montè Carlo Integration for Angular Chebyshev Series and Least Squares for Rational Polynomials.
3. Generating 200 random points $\{x_0, x_1, \ldots, x_{200}\}$ uniformly on the interval $[0, \pi]$.
4. Calculating the mean square error (MSE) using the randomly generated points with the formula:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(f(x_i) - \widehat{f}(x_i))^2 \tag{4.1}$$

   where $n = 200$, $x_i$ is the randomly-generated point, $f(x)$ is the actual function and $\widehat{f}(x)$ is the approximated function.

The approximating functions to be tested are as follows:

1. Rational:
   (a) Numerator degree = 4, denominator degree = 2
   (b) Numerator degree = 5, denominator degree = 5
   (c) Numerator degree = 5, denominator degree = 8
2. Angular Chebyshev Series
   (a) Up to degree 6.
   (b) Up to degree 10.
   (c) Up to degree 13.

## 4.2 Functions

The following functions will be used to benchmark:

1. Polynomials
   (a) $x$
   (b) $x^2 + 1$

(c) $\frac{1}{16}(231x^6 - 315x^4 + 105x^2 - 5)$ (Legendre Polynomial $P_6(x)$)
(d) $\frac{1}{2}(5x^3 + 3x^2 - 3x - 1)$ (Legendre Polynomials $P_2(x) + P_3(x)$)

2. Exponential Functions
   (a) $e^x$
   (b) $e^{x-\frac{\pi}{2}} - e^{-(x-\frac{\pi}{2})/128}$
   (c) $e^{x-\frac{\pi}{2}}(e^{x-\frac{\pi}{2}} + e^{-x+\frac{\pi}{2}})^{-1}$

3. Trigonometric Functions
   (a) $\sin(x)$
   (b) $\tan(\frac{x}{\pi} - .5)$
   (c) $\sin(5x)\cos(x)$
   (d) $\sin(10x)$ (High frequency oscillation)
   (e) $\sin(50x) + \cos(10x)$ (Combined high frequency oscillation)

4. Hyperbolic Functions
   (a) $\sinh(x - \frac{\pi}{2})$
   (b) $\tanh(x - \frac{\pi}{2})$
   (c) $\text{sech}(x - \frac{\pi}{2})$

Listing 4.1: Graphing and MSE Calculation of Function Approximation Methods

```
function y = poly_0(x)
  y = x
endfunction

function y = poly_1(x)
  y = x^2 - 1
endfunction

function y = poly_2(x)
  y = (231*x^6 - 315 * x^4 + 105 * x^2 - 5)/16
endfunction

function y = poly_3(x)
  y = (5 * x^3 + 3 * x^2 - 3 * x - 1)/2
endfunction

function y = exp_0(x)
  y = exp(x)
endfunction

function y = exp_1(x)
  y = exp(x - %pi/2) - exp(-(x - %pi/2)/128)
endfunction

function y = exp_2(x)
  y = exp(x - %pi/2) ./ (exp(x-%pi/2) + exp(-x + %pi/2))
endfunction

function y = trig_0(x)
```

```
  y = sin(x)
endfunction

function y = trig_1(x)
  y = tan(x/%pi - .5)
endfunction

function y = trig_2(x)
  y = sin(5*x) .* cos(x)
endfunction

function y = trig_3(x)
  y = sin(10*x)
endfunction

function y = trig_4(x)
  y = sin(50*x) + cos(10*x)
endfunction

function y = hyp_0(x)
  y = sinh(x - %pi/2)
endfunction

function y = hyp_1(x)
  y = tanh(x - %pi/2)
endfunction

function y = hyp_2(x)
  y = sech(x - %pi/2)
endfunction

//Calculates the Mean Square Error between two vectors.
function y = mse(y0, y1)
  y = 0
  n = min(size(y0,'*'), size(y1, '*'))
  for i = 1:n
    y = y + (y0(i) - y1(i))^2
  end
  y = y / n
endfunction

//
//Graphs the test functions against the function approximations and calculates
    the MSE.
//
//@param sampleSizesList A list of sample sizes
//@param nTestPoints The number of points to test agains for calculating the
    MSE.
//@param shouldPlot A boolean stating whether the functions should be plotted
    or not. When plotting the legend placement is set to interactive.
//
//@return results A 3-dimensional typed list containing the MSEs. The (i)(j)(k)
     entry corresponds to the k'th approximating function of the j'th test
    function with sampleSizesList(i) samples.
```

```
function results = start_analysis(sampleSizesList, nTestPoints, shouldPlot)
  funcprot(0)
  RATIONAL_N_DEG = [4, 5, 5]
  RATIONAL_D_DEG = [2, 5, 8]
  CHE_N_TERMS = [6, 10, 13]

  function results = analyze_func(f, fname, xs, xtest)
    ys = f(xs)
    rat_0 = rational_approx(xs, ys, RATIONAL_N_DEG(1), RATIONAL_D_DEG(1))
    rat_1 = rational_approx(xs, ys, RATIONAL_N_DEG(2), RATIONAL_D_DEG(2))
    rat_2 = rational_approx(xs, ys, RATIONAL_N_DEG(3), RATIONAL_D_DEG(3))

    che_0 = che_angle_proj(xs, ys, CHE_N_TERMS(1))
    che_1 = che_angle_proj(xs, ys, CHE_N_TERMS(2))
    che_2 = che_angle_proj(xs, ys, CHE_N_TERMS(3))

    ytest = f(xtest)
    output_rat_0 = horner(rat_0, xtest)
    output_rat_1 = horner(rat_1, xtest)
    output_rat_2 = horner(rat_2, xtest)
    output_che_0 = che_angle_eval(che_0, xtest)
    output_che_1 = che_angle_eval(che_1, xtest)
    output_che_2 = che_angle_eval(che_2, xtest)

    results = tlist(['constant', 'rat_0', 'rat_1', 'rat_2', 'che_0', 'che_1', '
        che_2'],...
    mse(ytest, output_rat_0),...
    mse(ytest, output_rat_1),...
    mse(ytest, output_rat_2),...
    mse(ytest, output_che_0),...
    mse(ytest, output_che_1),...
    mse(ytest, output_che_2))

    if shouldPlot
      f = scf()
      f.figure_name = fname
      f.auto_resize = "on"

      plot2d(xtest',...
      [output_rat_0' output_rat_1' output_rat_2' output_che_0' output_che_1'
          output_che_2' ytest'],...
      style=[2,3,4,5,6,7,1])
      e = gce()
      lgnd = legend(e.children, ["Rational 4/2"; "Rational 5/5"; "Rational 5/8"
          ;...
      "Angular Chebyshev 6"; "Angular Chebyshev 10"; "Angular Chebyshev 13";
          fname], 5)
    end

  endfunction

  results = list()

  xtest = gsort(rand(1, nTestPoints) * %pi, 'g', 'i')
```

```
  for sampleSize = sampleSizesList
    xs = linspace(0, %pi, sampleSize)
    res = tlist(['constant',...
    'poly_0', 'poly_1', 'poly_2', 'poly_3',...
    'exp_0', 'exp_1', 'exp_2',...
    'trig_0', 'trig_1', 'trig_2', 'trig_3', 'trig_4',...
    'hyp_0', 'hyp_1', 'hyp_2'],...
    analyze_func(poly_0, "Poly 1", xs, xtest),...
    analyze_func(poly_1, "Poly 2", xs, xtest),...
    analyze_func(poly_2, "Poly 3", xs, xtest),...
    analyze_func(poly_3, "Poly 4", xs, xtest),...
    analyze_func(exp_0, "Exp 1", xs, xtest),...
    analyze_func(exp_1, "Exp 2", xs, xtest),...
    analyze_func(exp_2, "Exp 3", xs, xtest),...
    analyze_func(trig_0, "Trig 1", xs, xtest),...
    analyze_func(trig_1, "Trig 2", xs, xtest),...
    analyze_func(trig_2, "Trig 3", xs, xtest),...
    analyze_func(trig_3, "Trig 4", xs, xtest),...
    analyze_func(trig_4, "Trig 5", xs, xtest),...
    analyze_func(hyp_0, "Hyp 1", xs, xtest),...
    analyze_func(hyp_1, "Hyp 2", xs, xtest),...
    analyze_func(hyp_2, "Hyp 3", xs, xtest))

    results($ + 1) = res
  end

endfunction

//
//Calculates the MSE of Chebyshev approximations of function f w.r.t. to the
    sample size and graphs them.
//
//@param nSampleSizesList An ascending list (not a vector) of integers
    containing the samples to be graphed. e.g. list(10, 20, 30, 40, 50, 60)
//@param nTestPoints The number of test points to be randomly generated and
    tested against for MSE calculation.
//@param testDegrees A list of integers of the Angular Chebyshev polynomials to
     be tested. i.e. list(3, 5) will test against T_0(x) to T_3(x) and T_0(x) to
     T_5(x).
//@param f The function to be analyzed.
//
//@return mseGraphs A length(nSampleSizesList) by length(testDegrees) matrix
    whose (i,j)th entry contains the MSE of Angular Chebyshev testDegrees(j)
    with sample size nSampleSizesList(i)
function mseGraphs = chebyshev_analysis(nSampleSizesList, nTestPoints,
    testDegrees, f)
  xtest = gsort(rand(1, nTestPoints) * %pi, 'g', 'i')
  ytest = f(xtest)
  mseGraphs = []

  for sampleSize = nSampleSizesList
    xs = linspace(0, %pi, sampleSize)
    ys = f(xs)
```

```
    res = []

    for deg = testDegrees
      che_approx = che_angle_proj(xs, ys, deg)
      output_che = che_angle_eval(che_approx, xtest)
      res = [res, mse(ytest, output_che)]
    end
    mseGraphs = [mseGraphs; res]
  end

  f = scf()

  f.figure_name = "Chebyshev MSE Accuracy"
  f.auto_resize = "on"

  leg_names = []
  for deg = testDegrees
    leg_names = [leg_names, "n = " + string(deg)]
  end

  x_axis = []
  for i = nSampleSizesList
    x_axis = [x_axis; i]
  end


  plot2d(x_axis, mseGraphs)
  e = gce()
  lgnd = legend(e.children, leg_names, 1)
  axes = gca()
  axes.x_label.text = "Samples"
  axes.y_label.text = "MSE"
endfunction
```

## 4.3 Results

Table 4.1 shows the mean square error calculations of the function approximation method with respect to specific functions at 30 and 100 samples. The 'a', 'b',... suffix of function classes corresponds to the enumeration in section 4.2. Values less than $1 \times 10^{-6}$ are rounded off to 0 while values exceeding 9999.0 and below 0.1000 are shown in scientific notation. In the interest of space, note the shorthand $m \oplus n = m \times 10^n$ and $m \ominus n = m \times 10^{-n}$. For example, $5.12 \oplus 3 = 5120$ and $1.28 \ominus 2 = .0128$. Figure 4.1 shows the graphs of notable cases with 50 sample points.

## 4.4 Remarks

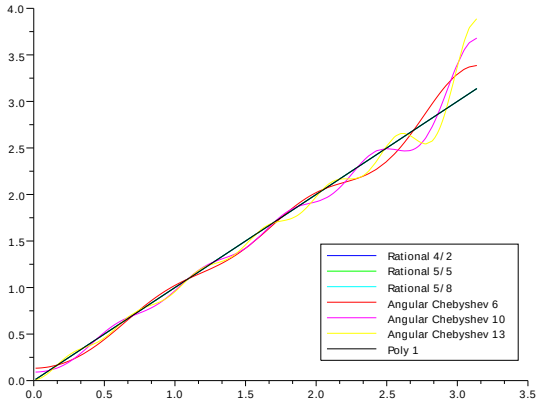
Angular Chebyshev series usually did worse than their rational counterparts as seen in table 4.1 and figure 4.1. For straight lines, the angular Chebyshev series were not very accurate because of their oscillatory nature. Moreover, as seen in 4.1a, 4.1b and 4.1e, they tend to offshoot at the edges of of the function. The angular Chebyshev series also did poorly for polynomials linearly independent of Chebyshev polynomials. However, for trigonometric functions, specifically for those with high frequencies, rational polynomials did a lot worse,

Table 4.1: Mean Square Errors of Function Approximation at Various Number of Samples with 200 Test Points

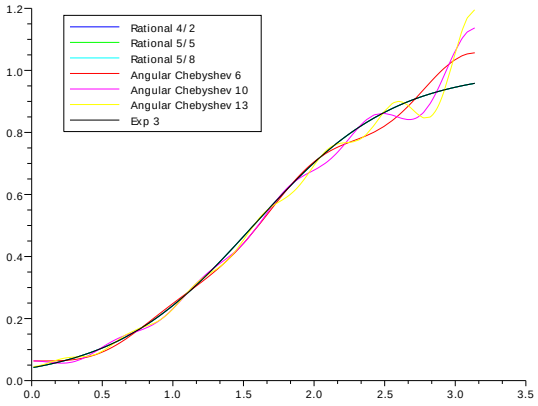
| Function | Ratl. a | | Ratl. b | | Ratl. c | | Cheb. a | | Cheb. b | | Cheb. c | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Inputs | 30 | 100 | 30 | 100 | 30 | 100 | 30 | 100 | 30 | 100 | 30 | 100 |
| Poly a | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 2.400⊖2 | 2.984⊖3 | 3.319⊖2 | 3.334⊖3 | 3.719⊖2 | 3.481⊖3 |
| Poly b | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.873⊖1 | 2.662⊖2 | 2.605⊖1 | 2.770⊖2 | 3.023⊖1 | 2.965⊖2 |
| Poly c | 1.756 | 1.393 | 0.128 | 8.356⊖2 | 22.10 | 18.37 | 4.887⊕5 | 1.401⊕5 | 5.555⊕5 | 8.273⊕4 | 5.946⊕5 | 7.011⊕4 |
| Poly d | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 19.07 | 3.169 | 25.53 | 2.891 | 28.441 | 2.888 |
| Exp a | 0.000 | 0.000 | 0.000 | 0.000 | 1.440⊖5 | 0.000 | 1.404 | 0.2419 | 1.849 | 0.2129 | 2.029 | 0.2072 |
| Exp b | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 4.050⊖2 | 8.240⊖3 | 5.276⊖2 | 6.442⊖3 | 6.228⊖2 | 6.499⊖3 |
| Exp c | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 2.148⊖3 | 2.035⊖4 | 3.051⊖3 | 2.809⊖4 | 3.433⊖3 | 3.117⊖4 |
| Trig a | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 8.837⊖4 | 3.907⊖4 | 7.079⊖4 | 1.587⊖4 | 6.257⊖4 | 1.063⊖4 |
| Trig b | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.440⊖3 | 2.220⊖4 | 2.318⊖3 | 2.206⊖4 | 3.388⊖3 | 2.949⊖4 |
| Trig c | 5.230 | 241.1 | 6.094 | 215.3 | 223.4 | 7.1411 | 8.343⊖2 | 8.269⊖2 | 5.111⊖3 | 4.973⊖3 | 1.832⊖3 | 1.512⊖3 |
| Trig d | 27.11 | 2.990 | 8.142 | 2.337 | 18.43 | 16.00 | 0.4590 | 0.4591 | 0.2203 | 0.2182 | 1.102⊖2 | 1.076⊖2 |
| Trig e | 4.218 | 3994.0 | 309.2 | 1.645 | 77.12 | 32.70 | 1.060 | 0.966 | 0.8460 | 0.4963 | 0.8708 | 0.4938 |
| Hyp a | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 2.771⊖2 | 5.779⊖3 | 4.184⊖2 | 4.305⊖3 | 6.050⊖2 | 5.272⊖3 |
| Hyp b | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 3.767⊖3 | 3.536⊖4 | 6.454⊖3 | 5.786⊖4 | 9.550⊖3 | 8.492⊖4 |
| Hyp c | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.452⊖3 | 1.709⊖4 | 1.902⊖3 | 1.924⊖4 | 2.276⊖3 | 2.214⊖4 |

Figure 4.1: Graph of Functions and their Approximations at 50 Linear Sample Points

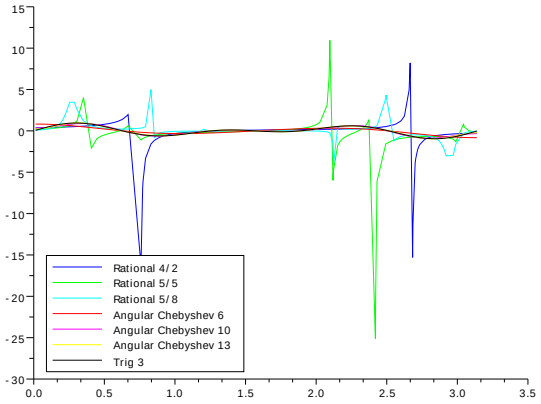
(a) Polynomial (a)



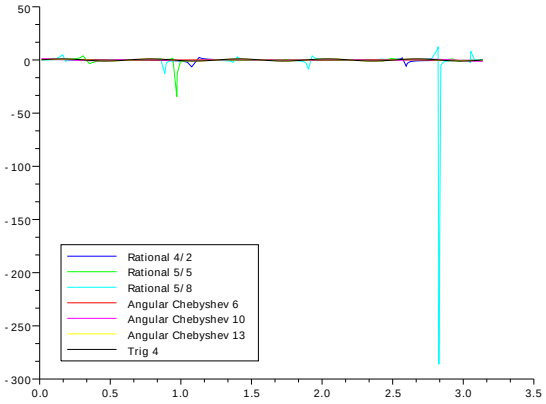

(b) Exponential (c)



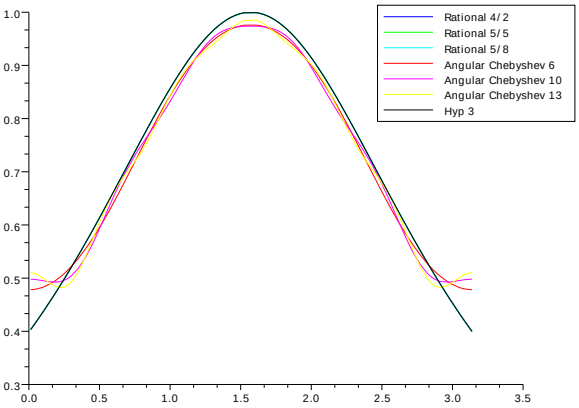

(c) Trigonometric (c)




(d) Trigonometric (d)



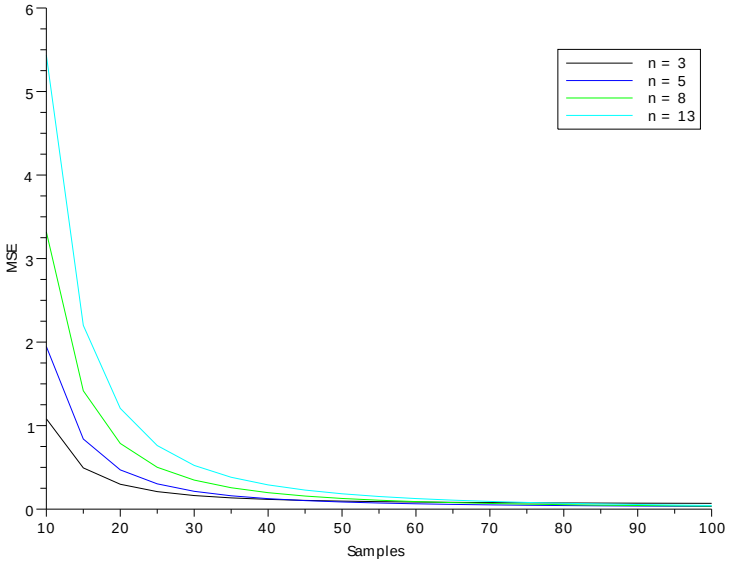

(e) Hypergeometric (c)



exhibiting numerical instability at some points (see figure 4.1c and 4.1d). Having higher-degree rational polynomials did not mitigate the instability but in fact, exacerbated it.

One thing to note about the Chebyshev series is that they are generally more stable than rational polynomials. While rational polynomials are more accurate in many cases, they exhibit instability on high-frequency functions, specifically, they exhibit sharp jumps at some points. In contrast, the Chebyshev series are more consistent in their errors with no sharp jumps though their MSE is higher than the rational polynomials'.

Figure 4.2: MSE v.s. Sample Size for Poly (a) on Angular Chebyshev Series with 200 Test Points




Also, another thing to note is that high degree Chebyshev series have higher MSEs then low degree Chebyshev series. However, with more sample points, high degree Chebyshev series benefit more than low degree Chebyshev series as seen in figure 4.2. In other words, high degree Chebyshev series are more sensitive to sample sizes than low degree ones.

# Bibliography


[1] Scilab Consortium. Scilab. Available from: `http://www.scilab.org` [cited September 15, 2010].

[2] Richard Culham. Chebyshev polynomials, March 2004. Available from: `http://www.mhtl.uwaterloo.ca/courses/me755/web_chap6.pdf` [cited September 3, 2010].

[3] Robin Green. Spherical harmonic lighting: The gritty details [online]. January 2003. Available from: `http://www.research.scea.com/gdc2003/spherical-harmonic-lighting.html` [cited September 3, 2010].

[4] Gilbert Strang. *Linear Algebra and Its Applications*. Thomson Learning, 3rd edition, 1988.