Recitation 8

Recitation Instructor: Shivam Verma
Email: shivamverma@nyu.edu
Ph: 718-362-7836
Office hours: WWH 605 (2.50 - 4.50 pm, Tuesdays)

Brief Overview

- Overview of Lagrange interpolation
- Runge's phenomenon
- Using Chebyshev points
- DFT and FFT, MATLAB example
- MATLAB *interp1* example

Interpolation overview

- **What is interpolation?** Quite simply, interpolation is function approximation (using polynomials, in our case). It is notably different from *extrapolation*, since it only involves "constructing new data points within the range of a discrete set of known data points" [1].
    - For a general function f(x), if we approximate it with a polynomial, it is then easier (more economical) to store coefficients of the interpolating polynomial than the function itself.
    - We want to evaluate the function at an unknown point, using the given data.
- **Where is it used?**
    - Approximating functions (uses in physics, applied mathematics, Engineering etc.)
    - Numerical integration - finding integrals of complex functions numerically using quadrature rules
- Many **kinds of interpolation** methods. We will study only a few in this class.
    - **Linear**
    - **Polynomial**
    - **Trigonometric**
    - **Bezier**
    - **Spline**
    - **Bayesian methods**
- A very brief overview of interpolation is [1].

Lagrange Interpolation

You've been given the following (n+1) points or nodes:
$(x_0,y_0),(x_1,y_1),...,(x_n,y_n)$

Assuming these points are ordered, i.e. $x_0 < x_1 < ... < x_n$, interpolation involves finding out $f(x_k)$, where $x_k \in (x_0,x_n),$ i.e. an intermediate point. These points are known as the interpolation points or nodes.

Thus, we approximate the "function" of given points with:
$P_n(x) = a_n x^n + a_{n-1}x^{n-1} + ... + a_1 x + a_0$

Such that $P_n(x_i) = y_i,\ i = 0, 1, ..., n.$ The problem then becomes to find the coefficients of the interpolating polynomial!

From the equation(s) above, we have an $(n+1)\times(n+1)$ linear system:

$$\underbrace{\begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{n-1} & x_0^n \\ 1 & x_1 & x_1^2 & \dots & x_1^{n-1} & x_1^n \\ \vdots & \vdots & \vdots & \vdots & \vdots & \\ 1 & x_{n-1} & x_{n-1}^2 & \dots & x_{n-1}^{n-1} & x_{n-1}^n \\ 1 & x_n & x_n^2 & \dots & x_n^{n-1} & x_n^n \end{bmatrix}}_{\mathbf{V}} \underbrace{\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \\ a_n \end{bmatrix}}_{\vec{a}} = \underbrace{\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix}}_{\vec{b}}$$

The 'V' matrix is called Vandermonde matrix. This is one way of finding the interpolating coefficients. We can solve this as a linear system in $O(n^3)$, and then evaluate the polynomial at a point in $O(n)$ operations.
However, it is not used very often since it is computationally expensive (compared to our other solutions, at least!). See [3] for more details.

The other method is called Lagrange interpolation.

Definition 6.1 *Suppose that $n \geq 0$. Let x_i, $i = 0, \ldots, n$, be distinct real numbers, and y_i, $i = 0, \ldots, n$, real numbers. The polynomial p_n defined by*

$$p_n(x) = \sum_{k=0}^{n} L_k(x) y_k \,, \tag{6.6}$$

with $L_k(x)$, $k = 0, 1, \ldots, n$, defined by (6.4) when $n \geq 1$, and $L_0(x) \equiv 1$ when $n = 0$, is called the **Lagrange interpolation polynomial** *of degree n for the set of points $\{(x_i, y_i)\colon i = 0, \ldots, n\}$. The numbers x_i, $i = 0, \ldots, n$, are called the* **interpolation points**.

Where

$$L_k(x) = \prod_{\substack{i=0 \\ i \neq k}}^{n} \frac{x - x_i}{x_k - x_i} \,.$$

For illustration of method, see example 6.1 from book.

This method is faster than solving the Vandermonde matrix, since calculating the Lagrange interpolation polynomials takes $O(n^2)$ operations, as we don't necessarily have to find the coefficients. See [3] for details.

Runge's phenomenon

The error estimate of Lagrange interpolation is given by:

Theorem 6.2 *Suppose that $n \geq 0$, and that f is a real-valued function, defined and continuous on the closed real interval $[a, b]$, such that the derivative of f of order $n+1$ exists and is continuous on $[a, b]$. Then, given that $x \in [a, b]$, there exists $\xi = \xi(x)$ in (a, b) such that*

$$f(x) - p_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}\pi_{n+1}(x)\,, \tag{6.8}$$

where

$$\pi_{n+1}(x) = (x - x_0)\dots(x - x_n)\,. \tag{6.9}$$

Moreover

$$|f(x) - p_n(x)| \leq \frac{M_{n+1}}{(n+1)!}|\pi_{n+1}(x)|\,, \tag{6.10}$$

where

$$M_{n+1} = \max_{\zeta \in [a,b]} |f^{(n+1)}(\zeta)|\,.$$

We can see that for the error to approach zero, i.e.

$$\lim_{n\to\infty} \max_{x\in[a,b]} |f(x) - p_n(x)| = 0\,,$$

we require from (6.10) that:

$$\lim_{n\to\infty} \frac{M_{n+1}}{(n+1)!} \max_{x\in[a,b]} |\pi_{n+1}(x)| = 0\,,$$

Which depends on the interpolating points or nodes we choose.

There are many ways to choose these points:

1. **Equidistant nodes**

   For an interval $[a, b]$, $x_i = a + i(b-a)/n$

For these points, it can be proved that the numerator term increases faster than the denominator term as $n \to \infty$, and hence the relative error blows up for large n.

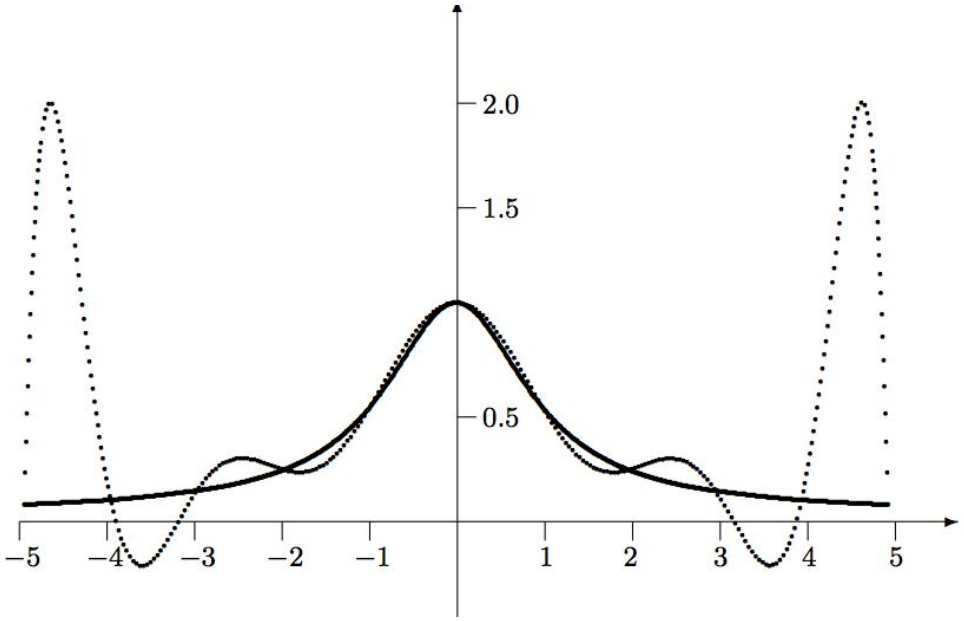
One way to see this is by looking at the condition number of the interpolation.

From Thm. 7.3 [2],

$$\kappa_{\text{abs}} = \Lambda_n := \max_{t \in [a,b]} \sum_{i=0}^{n} |L_{in}(t)|$$

The Λ_n coefficient is known as Lebesgue constant, and it gives us the absolute condition number of the polynomial interpolation.

For equidistant points, as $n \to \infty$, Λ_n increases exponentially and the 'max error' increases. This effect is seen near the edges and is known as Runge's phenomenon.



2. Chebyshev nodes

Chebyshev nodes are points which are the roots of Chebyshev polynomials. They're discussed below:

$x_i = cos(\frac{2i+1}{2n+2}\pi),\ i = 0, ..., n\,.$

For these special nodes, the Lebesgue constant is finite as n increases, and therefore we do not observe Runge's phenomenon.

An example below shows how the Lebesgue constant varies for both types of nodes.

n	Λ_n for equidistant nodes	Λ_n for Chebyshev nodes
5	3.106292	2.104398
10	29.890695	2.489430
15	512.052451	2.727778
20	10986.533993	2.900825

This can be seen by:

$$Equispaced\ points:\ \Lambda_N \sim 2^N/e\,N\log N.$$

$$Chebyshev\ points:\ \Lambda_N \sim const\log N.$$

Chebyshev polynomials satisfy the 3-term recurrence relation:

$T_k(x) = 2cos(x)\cdot T_{k-1}(x) - T_{k-2}(x),\ k = 2, 3, ...$

One form of Chebyshev polynomials is:

$T_k(x) = cos(k\cdot cos^{-1}(x))$

With $T_0(x) = 1,\ T_1(x) = x.$

See Ch. 6 of [2] for more detail.

DFT and FFT (Discrete and Fast Fourier Transform)

This is useful in finding the interpolation polynomial with Chebyshev nodes.

The DFT is given by:

$$X_k = \frac{1}{N}\sum_{n=0}^{N-1} x_n\, e^{-2\pi ikn/N}$$

It transforms a signal (or an input vector) from the time-domain to the frequency-domain. If we are given an input of N such points, then we can get the corresponding DFT in $O(N^2)$. This is because there are N-1 additions for each X_k, and N such elements.

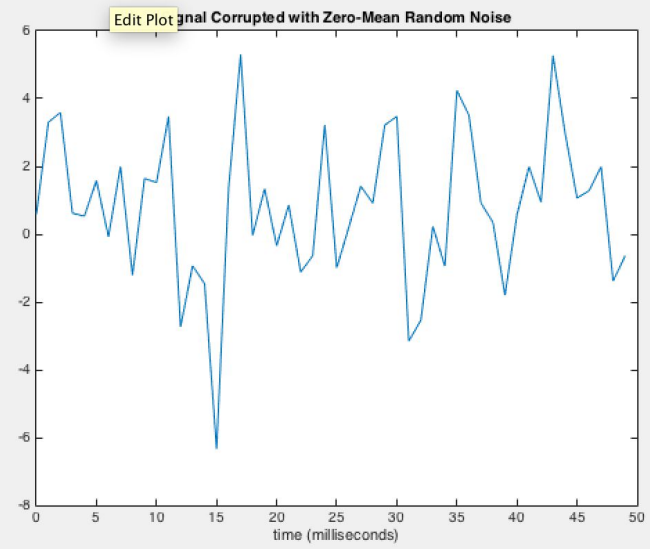
A Fast Fourier Transform (FFT) is a faster way to calculate the DFT. There are many algorithms for calculating the DFT, but they all have $O(N \cdot log(N))$. A nice video explanation of one such FFT algorithm is given by [9]. We will briefly talk about it in class.

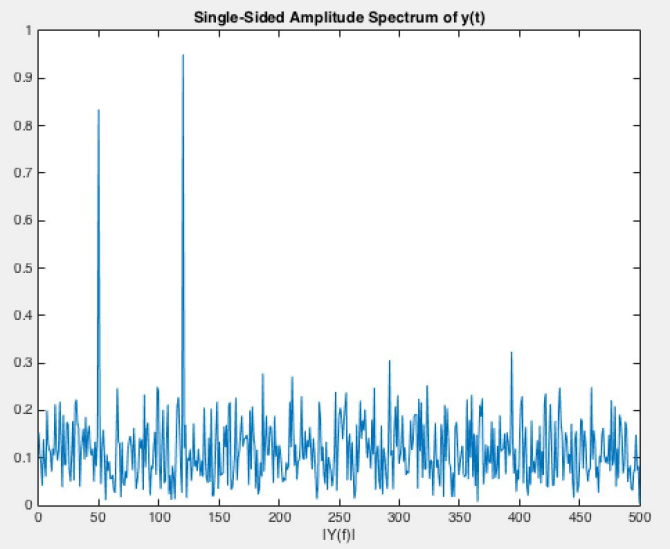
We can understand FFT better through a MATLAB example:

```
Fs=1000;    %sampling frequency
T=1/Fs;     %sampling time
L=1000;     %length of signal
t=(0:L-1)*T;    %time vector
%sum of a 50 Hz sinusoid and a 120 Hz sinusoid
x = 0.7*sin(2*pi*50*t) + sin(2*pi*120*t);
y = x + 2*randn(size(t));   %sinusoids plus noise
figure
plot(Fs*t(1:50),y(1:50))
title('Signal Corrupted with Zero-Mean Random Noise')
xlabel('time (milliseconds)')

NFFT = 2^nextpow2(L);   %next pow of 2 from length of y
Y = fft(y,NFFT)/L;
f = Fs/2*linspace(0,1,NFFT/2+1);
figure
%plot single-sided amplitude spectrum
plot(f,2*abs(Y(1:NFFT/2+1)))
title('Single-Sided Amplitude Spectrum of y(t)')
xlabel('|Y(f)|')
```

You should get the following plots:





As you can see, the (fast) fourier transform of the input signal gives us the correct frequencies. We can use the same idea to find the coefficients of the interpolating polynomial when it can be represented as a DFT.

Brief introduction to Barycentric interpolation formula

We will discuss notes given by Prof. Widlund, which are taken from Prof. Trefethen's textbook [8] (chapters 5, 15).

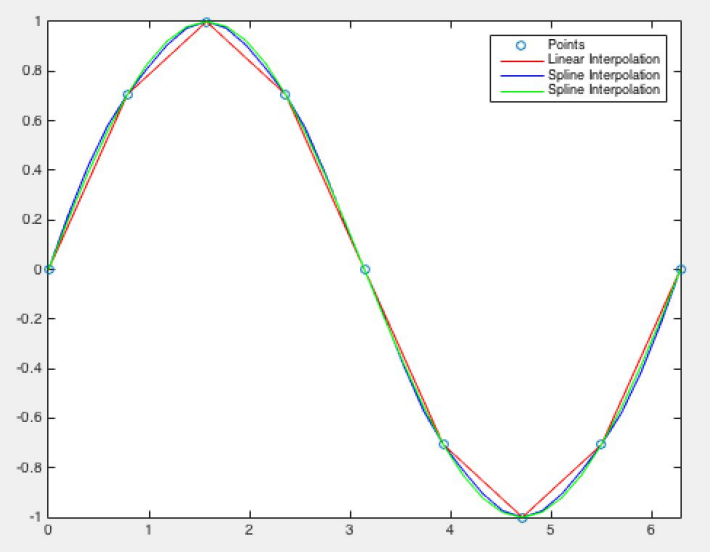
Interpolation in MATLAB

The MATLAB function *interp1* performs different kinds of interpolation, including linear, polynomial ('cubic') and spline. Below, we demonstrate these 3 types.

```
x = 0:pi/4:2*pi;
v = sin(x);
xq = 0:pi/16:2*pi;
figure
plot(x,v,'o');
xlim([0 2*pi]);
hold on
```

```
vq1 = interp1(x,v,xq); %linear interpolation
plot(xq,vq1,'r-');
vq2 = interp1(x,v,xq,'pchip'); %cubic interpolation
plot(xq,vq2,'b-');
vq3 = interp1(x,v,xq,'spline'); %spline interpolation
plot(xq,vq3,'g-');
legend('Points','Linear Interpolation','Spline Interpolation','Spline
Interpolation')
```

You should get the following plot once you run the above code in MATLAB.



See MATLAB documentation [5] and [6] for more info.

Note: Besides the textbook, I found chap. 6 of [2], as well as [7] and [8] to be very helpful for understanding interpolation.

Helpful links

1. https://en.wikipedia.org/wiki/Interpolation
2. Deuflhard and Hohmann et al. (Numerical Analysis in Modern Scientific Computing)
3. http://pages.cs.wisc.edu/~sifakis/courses/cs412-s13/lecture_notes/CS412_12_Feb_2013.pdf
4. http://www2.lawrence.edu/fast/GREGGJ/Math420/Section_3_1.pdf
5. http://www.mathworks.com/help/matlab/ref/interp1.html
6. http://my.math.wsu.edu/help/matlab/interp1.html
7. http://butler.cc.tut.fi/~piche/numa/lecture0708.pdf
8. https://people.maths.ox.ac.uk/trefethen/ATAP/
9. https://www.youtube.com/watch?v=EsJGuI7e_ZQ
10. An interactive guide to FT - http://betterexplained.com/articles/an-interactive-guide-to-the-fourier-transform/