# Numerical Analysis Team 2.2 Project Documentation

## About

Our project uses Python 3.5+ code and the following dependencies (please note that the project will not run with Python 2.7):

* numpy
* scipy
* sympy
* matplotlib

These dependencies gave us access to robust mathematical algorithms and data structures, but this unfortunately comes at a cost… Installation kind of sucks. Detailed instructions for installation will follow.

A few things to note; we have built the application to either run through a GUI interface, or you can call the algorithms directly if you’re comfortable using Python in this way. Instructions will also follow.

## Installation

### The easy way

The easy way simply installing Anaconda, a Python distribution for data science:

<https://www.continuum.io/downloads>

This will install Python 3.5, Numpy, Scipy, Sympy, and Matplotlib all for you. You can see what packages it will install here:

<https://docs.continuum.io/anaconda/pkg-docs>

### The hard way

We have placed the dependencies for Windows You’ll need Python 3.5 and, run the following commands in the install directory in the following order:

**If you’re on Windows**

pip install numpy-1.11.3+mkl-cp35-cp35m-win\_amd64.whl

pip install scipy-0.19.0-cp35-cp35m-win\_amd64.whl

pip install matplotlib-2.0.0-cp35-cp35m-win\_amd64.whl

pip install mpmath

pip install sympy

**If you’re on a Mac/Linux OS**

python setup.py install

If that still doesn’t work, reach out to one of us and we’ll try to help you get up and running.

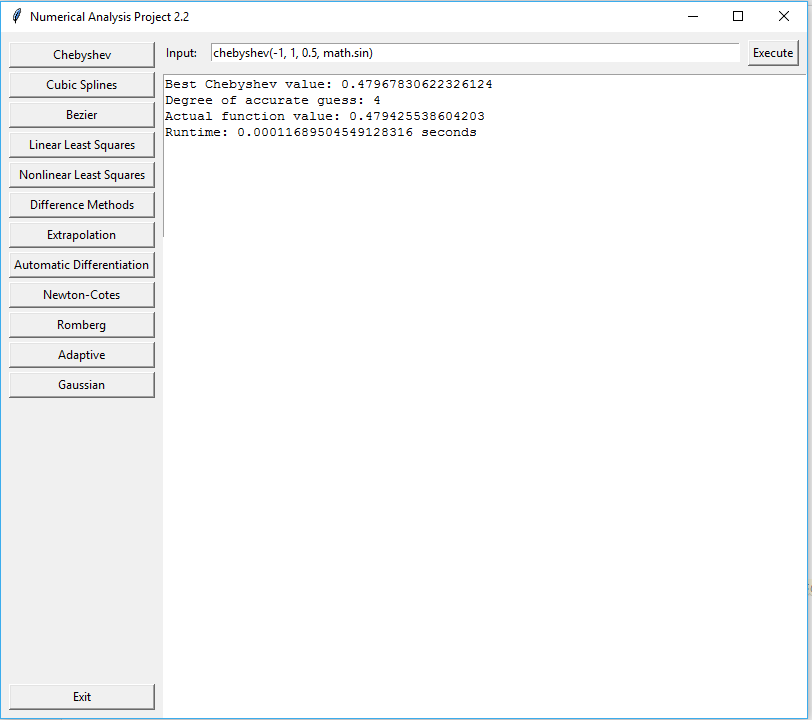
# Numerical Analysis Team 2.2 Project Documentation

## Algorithm Documentation- Here you are going to find the necessary documentation to run all our algorithms, along with various information on what will work and not work from the test questions. We will try to be as explicit as possible, but please do not hesitate to reach out to us if you have any questions about our program.

Section 1: Interpolation

We have implemented 3 separate algorithms in order to assist you in interpolating your data. Please understand that our functions will be limited in scope, and you must input data as prescribed and anything not explicitly stated here or in our GUI examples will not work.

Chebyshev

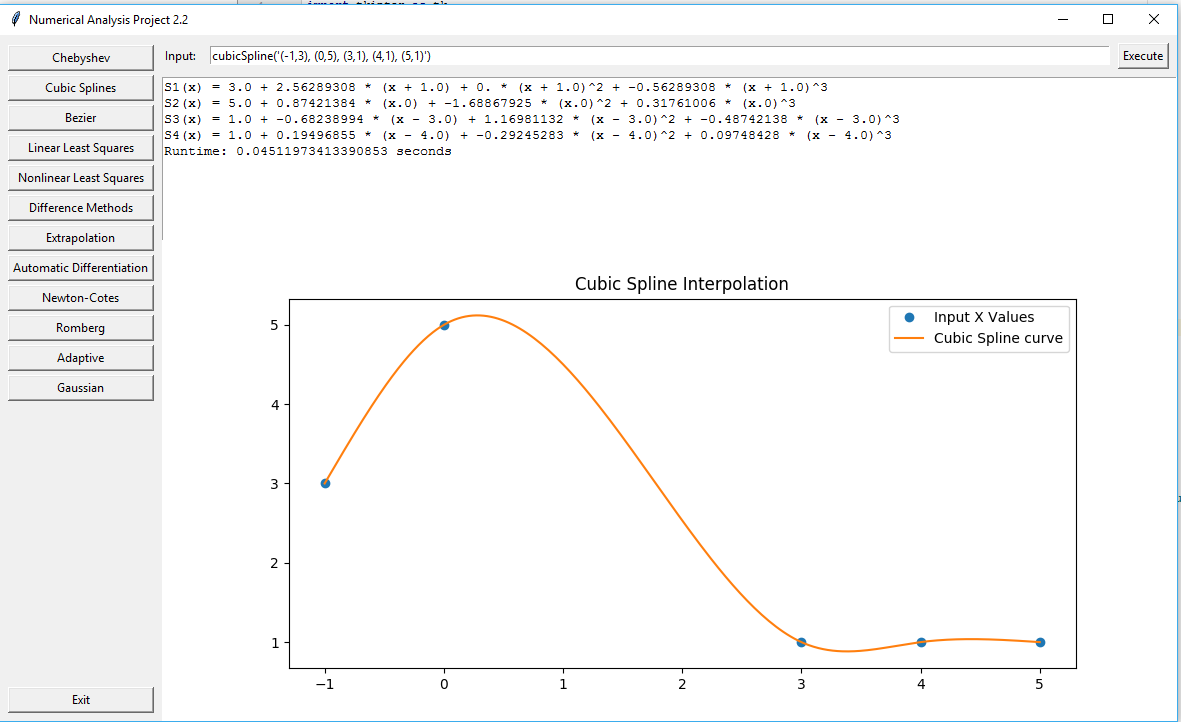


Runs the Chebyshev algorithm up to 30 times, increasing degree n until the guess is sufficiently close. Outputs the calculated Chebyshev value, the degree of the polynomial where the best guess was calculated and the actual value from the function. You MUST enter a function that can be evaluated, because func is not an optional argument.

**Example usage**: chebyshev(-1, 1, 0.5, math.sin)

Advanced functions can be input as example: lambda x: (math.sin(x) - math.cos(x))

Cubic Splines



Takes a string of points in the string form: '(-1,3), (0,5), (3,1), (4,1), (5,1)' and optionally, the graph resolution. Prints the cubic spline functions and displays an interpolated line plot below.

Example usage: cubicSpline('(-1,3), (0,5), (3,1), (4,1), (5,1)')

or cubicSpline('(-1,3), (0,5), (3,1), (4,1), (5,1)', resolution=2) for a low resolution graph.

Bezier



Takes a series of points in the form: [[1,0,6,2],[1,-1,0,1],[1,1,6,0]] and outputs the Bezier spline's knots and control points based on the input coordinates. It can also take in a series of **2-dimensional** points in order to get the Bezier interpolation equations. If you input points it will also output a graph.

Example usage: bezier([[1,0,6,2],[1,-1,0,1],[1,1,6,0]], 4), bezier([(1,0),(2,0),(0,3),(0,1)],4)

Adaptive

Automatic Differentiation

Bezier

Cubic Splines

Difference Methods

Extrapolation

Gaussian Integration

Romberg Integration

Adaptive Integration

Linear Least Squares

Nonlinear Least Squares

Newton-Cotes