Description

The code in this script generates a list of shape functions and a list of their derivatives. The shape functions are **Lagrange polynomials** over the domain [-1, 1]. Each shape function will be equal to 1 at the x-coordinate that corresponds to it's index in the shape function array, and zero at all others.

For example, if 3 element nodes are requested, the domain [-1, 1] will be split into three evenly spaced x-values: [-1.0, 0.0, 1.0]. Then, three shape functions are created: [N_0, N_1, N_2]. The result of calling N_0 at x[0] will be 1, and will equal zero at the other two x-values, as shown:

- `N`$_0$`( x[0] ) = 1.0`
- `N`$_0$`( x[1] ) = 0.0`
- `N`$_0$`( x[2] ) = 0.0`

The functions are generated as a summation of lambda functions, which while are probably not the most efficient, provide us with an easy way to generalize the Lagrange polynomial, defined as:

$$N_i(x) = \prod_{j=1(j\neq i)}^{n} \frac{x - x_j}{x_i - x_j} = \frac{(x - x_1)(x - x_2)...(x - x_{i-1})(x - x_{i+1})...(x - x_n)}{(x_i - x_1)(x_i - x_2)...(x_i - x_{i-1})(x_i - x_{i+1})...(x_i - x_n)}$$

and it's derivative, defined as:

$$N_i'(x) = \frac{\sum_{k=0}^{n} \prod_{l=0,l\neq k}^{n} (x - x_l)}{\prod_{k=0,k\neq j}^{n} (x_j - x_k)}$$

Code

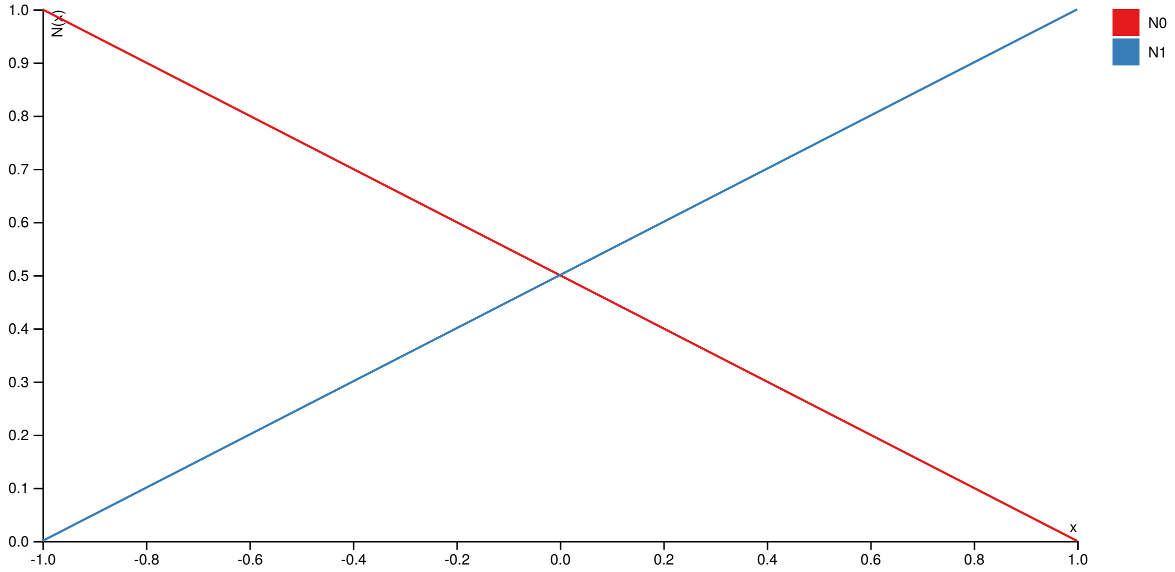
The complete code, shown below, can also be found on **Github**

```
import matplotlib.pyplot as plt

def shape_functions ( num_element_nodes ):

    # Create xi array
    xi = [ -1. + float(i) * 2. / float( num_element_nodes - 1 )  for i in range( num_element_nodes ) ]

    # Create N and dN arrays
    N = []
    dN = []

    for i in range( num_element_nodes ):

        # Create shape function
        f = []

        for j in range( num_element_nodes ):

            if i != j:

                f.append( lambda _xi, _i=xi[i], _j=xi[j]: ( _xi - _j ) / ( _i - _j ) )

        N.append( lambda _xi, _f=f: reduce( lambda a,b: a*b, map( lambda c: c(_xi), _f ) ) )


        # Create derivative of shape function
        mf = []

        for j in range( num_element_nodes ):

            if i != j:

                m = 1.0 / ( xi[ i ] - xi[ j ] )
                f = []

                for k in range( num_element_nodes ):

                    if k != i and k != j:

                        f.append( lambda _xi, _i=xi[ i ], _k=xi[ k ]: ( _xi - _k ) / ( _i - _k ) )

                    else:

                        f.append( lambda _xi: 1.0 )

                mf.append( ( m, f ) )

        dN.append( lambda _xi, _mf=mf: reduce( lambda a,b: a+b, [ _m * reduce( lambda c,d: c*d, map( lambda __f: __f(_xi)

    return N, dN, xi

def plot_shape_functions ( shape_fns ):

    npoints = 250
    x = [ -1 + i * 2. / ( npoints- 1 )  for i in range( npoints ) ]

    for f in shape_fns:
        plt.plot( x, [ f( i ) for i in x ] )

    plt.show()

def plot_f_df ( f, df ):

    npoints = 250
    x = [ -1 + i * 2. / ( npoints- 1 )  for i in range( npoints ) ]

    plt.plot( x, [ f( i ) for i in x ] )
    plt.plot( x, [ df( i ) for i in x ] )
    plt.show()


if __name__ == '__main__':
    N, dN, xi = shape_functions( 3 )
    plot_shape_functions( N )
    # plot_f_df( N[0], dN[0] )
```

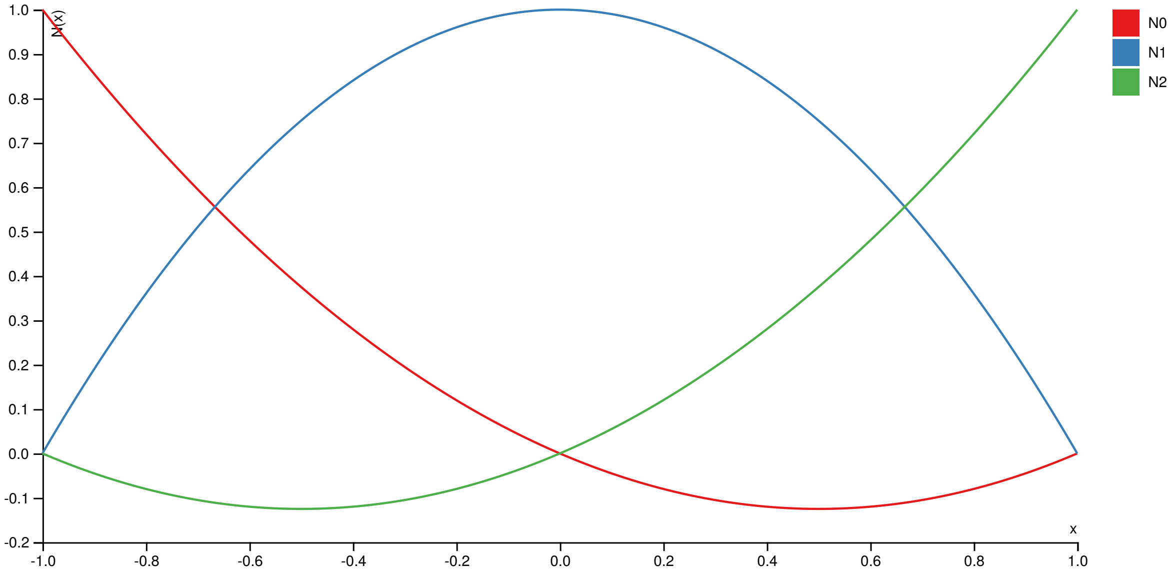
Sample Plots

The following plots show a few examples of the functions that are generated by the script.

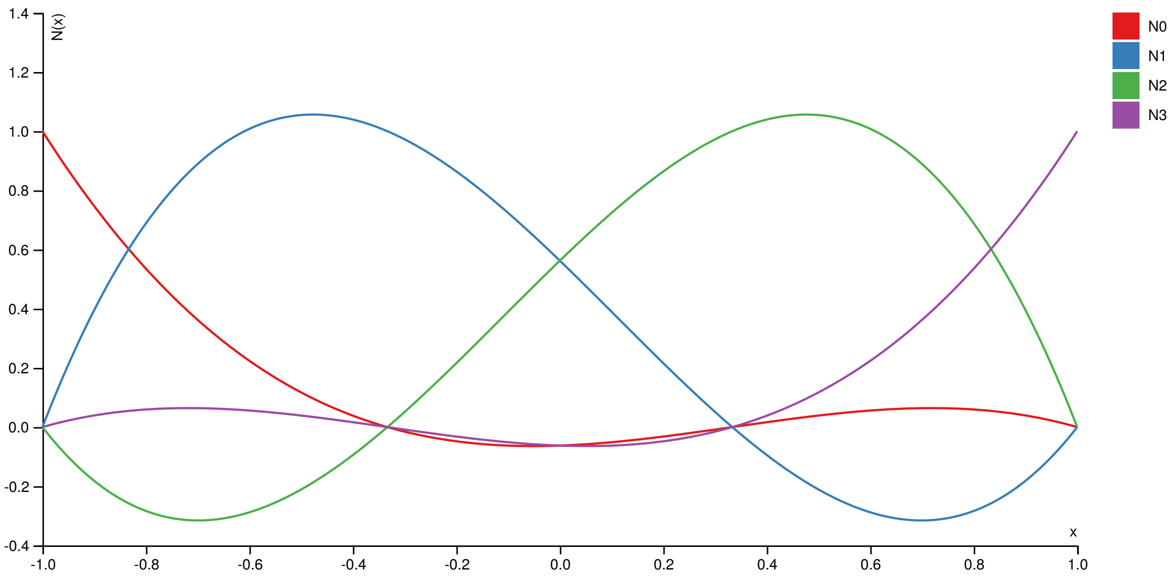
2 element nodes



3 element nodes



4 element nodes


N(x)
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
-0.2
-0.4
-1.0
-0.8
-0.6
-0.4
-0.2
0.0
0.2
0.4
0.6
0.8
1.0
x
N0
N1
N2
N3