Polynomial interpolation: the Newton form

We have seen that, given $d+1$ points $(x_0, y_0), (x_2, y_2), \ldots, (x_d, y_d)$ with distinct x_i, there exists a unique interpolating polynomial $p(x)$ of degree at most d.

The **Lagrange form** of this polynomial is $p(x) = \sum_{j=0}^{d} y_j \frac{\prod_{i\neq j}(x-x_i)}{\prod_{i\neq j}(x_j-x_i)}$.

The **Newton form** instead expresses the polynomial in the form

$$p(x) = c_0 + c_1(x-x_0) + c_2(x-x_0)(x-x_1) + c_3(x-x_0)(x-x_1)(x-x_2) + \ldots$$

Comment. Note that the Newton form of a polynomial can be thought of as a generalization of Taylor expansion. Indeed, we get Taylor expansion around $x=c$ if we choose all x_i to be equal to c (of course, for the purposes of interpolation, the x_i will be different).

Example 71. Determine the minimal polynomial interpolating the points $(-3,-1), (-1,5), (0,8), (2,-1)$.

Solution. (Lagrange) The interpolating polynomial in Lagrange form is:

$$\begin{aligned} p(x) &= -1\frac{(x+1)x(x-2)}{(-3+1)(-3)(-3-2)} + 5\frac{(x+3)x(x-2)}{(-1+3)(-1)(-1-2)} + 8\frac{(x+3)(x+1)(x-2)}{(0+3)(0+1)(0-2)} - 1\frac{(x+3)(x+1)x}{(2+3)(2+1)2} \\ &= -\frac{1}{2}x^3 - 2x^2 + \frac{3}{2}x + 8 \end{aligned}$$

Solution. (Newton, direct approach) The interpolating polynomial in Newton form is

$$p(x) = c_0 + c_1(x+3) + c_2(x+3)(x+1) + c_3(x+3)(x+1)x.$$

We use the four points to solve for the coefficients c_i:

$$\begin{aligned} (-3,-1) &: c_0 = -1 \\ (-1,5) &: \underbrace{c_0}_{-1} + 2c_1 = 5 \quad \Longrightarrow \quad c_1 = 3 \\ (0,8) &: \underbrace{c_0}_{-1} + 3\underbrace{c_1}_{3} + 3c_2 = 8 \quad \Longrightarrow \quad c_2 = 0 \\ (2,-1) &: \underbrace{c_0}_{-1} + 5\underbrace{c_1}_{3} + 15\underbrace{c_2}_{0} + 30c_3 = -1 \quad \Longrightarrow \quad c_2 = -\frac{1}{2} \end{aligned}$$

Hence, $p(x) = -1 + 3(x+3) - \frac{1}{2}(x+3)(x+1)x = -\frac{1}{2}x^3 - 2x^2 + \frac{3}{2}x + 8$.

Comment. Note how, by design of the Newton form, the equation for each point engaged one additional coefficient c_j, allowing us to solve for c_j (without having to combine several equations).

In particular, note how we are building intermediate interpolating polynomials:

- $c_0 + c_1(x+3) = -1 + 3(x+3)$ interpolates $(-3,1), (-1,6)$.
- $c_0 + c_1(x+3) + c_2(x+3)(x+1) = -1 + 3(x+3)$ interpolates $(-3,1), (-1,6), (0,3)$.

Newton's divided differences

Next, we observe an alternative way of computing the coefficients c_j in the Newton form

$$p(x) = c_0 + c_1(x-x_0) + c_2(x-x_0)(x-x_1) + c_3(x-x_0)(x-x_1)(x-x_2) + \dots \tag{1}$$

for interpolating a function f at $x = x_0, x_1, x_2, \dots$.

Example 72. Determine the first few coefficients. Below, we will use the following notation for these coefficients: $c_0 = f[x_0]$, $c_1 = f[x_0, x_1]$, $c_2 = f[x_0, x_1, x_2]$, ...

Solution. For brevity, we write $y_j = f(x_j)$.

- Using (x_0, y_0): $p(x_0) = c_0 \overset{!}{=} y_0$
  $f[x_0] = c_0 = y_0$
- Using (x_1, y_1): $p(x_1) = c_0 + c_1(x_1 - x_0) \overset{!}{=} y_1$
  $f[x_0, x_1] = c_1 = \dfrac{y_1 - y_0}{x_1 - x_0}$
  Note that the coefficient $f[x_0, x_1]$ is a **divided difference** (the slope of the line through the two points).
- Using (x_2, y_2): $p(x_2) = c_0 + c_1(x_2 - x_0) + c_2(x_2 - x_0)(x_2 - x_1) \overset{!}{=} y_2$
  $f[x_0, x_1, x_2] = c_2 = \dfrac{y_2 - y_0 - \frac{y_1 - y_0}{x_1 - x_0}(x_2 - x_0)}{(x_2 - x_0)(x_2 - x_1)} \overset{\text{check!}}{=} \dfrac{\frac{y_2 - y_1}{x_2 - x_1} - \frac{y_1 - y_0}{x_1 - x_0}}{x_2 - x_0} = \dfrac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0}$
  The coefficient $f[x_0, x_1, x_2]$ is what we call a **divided difference of order 2**.

Definition 73. Define $f[x_0, x_1, \dots, x_n]$ to be the coefficient of x^n (the highest power of x) in the minimal-degree polynomial interpolating f at $x = x_0, x_1, \dots, x_n$.

Important. In other words, $f[x_0, x_1, \dots, x_n]$ is the coefficient c_n in the Newton form (1).
$f[x_0, x_1, \dots, x_n]$ is called a **divided difference of order n** of the function f because of the recursive relation illustrated in the previous example, which is proven in general in the next theorem.
Note that, by definition, $f[x_0, x_1, \dots, x_n]$ does not depend on the order of the points.

Theorem 74. The divided differences $f[x_0, x_1, \dots, x_n]$ are recursively determined by $f[a] = f(a)$ as well as the relation

$$f[P, a, b] = \frac{f[P, b] - f[P, a]}{b - a},$$

where P is a set of points.

For instance. With $P = x_1, \dots, x_{n-1}$ and $a = x_0$, $b = x_n$, the recursive relation becomes

$$f[x_0, \dots, x_n] = \frac{f[x_1, \dots, x_n] - f[x_0, \dots, x_{n-1}]}{x_n - x_0}.$$

Proof. Suppose that $P=\{x_0,\dots,x_n\}$ and that

$$p_0(x)=c_0+c_1(x-x_0)+c_2(x-x_0)(x-x_1)+\dots+c_n(x-x_0)(x-x_1)\cdots(x-x_{n-1})$$

is the interpolating polynomial for $x_0,\dots,x_n$. Then

$$\begin{aligned} p_a(x) &= p_0(x)+f[P,a](x-x_0)(x-x_1)\cdots(x-x_{n-1})(x-x_n),\\ p_b(x) &= p_0(x)+f[P,b](x-x_0)(x-x_1)\cdots(x-x_{n-1})(x-x_n) \end{aligned}$$

are the interpolating polynomials for $x_0,\dots,x_n,a$ and $x_0,\dots,x_n,b$, respectively. The claim follows if we can show that

$$p_{a,b}(x)=p_a(x)+\cdot\frac{f[P,b]-f[P,a]}{b-a}(x-x_0)(x-x_1)\cdots(x-x_{n-1})(x-x_n)(x-a)$$

is the interpolating polynomial for $x_0,\dots,x_n,a,b$. By construction, it interpolates $x=x_0,\dots,x_n,a$. To see that it also interpolates $x=b$, note that

$$\begin{aligned} p_{a,b}(b) &= p_a(b)+\cdot(f[P,b]-f[P,a])\boxed{(b-x_0)(b-x_1)\cdots(b-x_{n-1})(b-x_n)}\\ &= p_0(b)+f[P,a]\boxed{(b-x_0)\cdots(b-x_{n-1})(b-x_n)}+\cdot(f[P,b]-f[P,a])\boxed{(b-x_0)\cdots(b-x_n)}\\ &= p_0(b)+f[P,b]\boxed{(b-x_0)\cdots(b-x_{n-1})(b-x_n)}\\ &= p_b(b)=f(b). \end{aligned}$$

□

(Newton form using divided differences)

The Newton form of the polynomial $p(x)$ interpolating f at $x=x_0,x_1,\dots$ is

$$p(x)=c_0+c_1(x-x_0)+c_2(x-x_0)(x-x_1)+c_3(x-x_0)(x-x_1)(x-x_2)+\dots,$$

where the coefficients $c_n=f[x_0,x_1,\dots,x_n]$ can be computed using the triangular scheme:

	$f[\cdot]$	$f[\cdot,\cdot]$	$f[\cdot,\cdot,\cdot]$	$f[\cdot,\cdot,\cdot,\cdot]$	…
x_0	$f[x_0]$				
		$f[x_0,x_1]=\frac{f[x_1]-f[x_0]}{x_1-x_0}$			
x_1	$f[x_1]$		$f[x_0,x_1,x_2]=\frac{f[x_1,x_2]-f[x_0,x_1]}{x_2-x_0}$		
		$f[x_1,x_2]=\frac{f[x_2]-f[x_1]}{x_2-x_1}$		$f[x_0,x_1,x_2,x_3]=\dots$	
x_2	$f[x_2]$		$f[x_1,x_2,x_3]=\frac{f[x_2,x_3]-f[x_1,x_2]}{x_3-x_1}$		…
		$f[x_2,x_3]=\frac{f[x_3]-f[x_2]}{x_3-x_2}$		…	
x_3	$f[x_3]$	…			…

Note that the coefficients $c_n=f[x_0,x_1,\dots,x_n]$ needed for the Newton form appear at the top edge of the triangle (in the shaded cells).

Example 75. Determine the minimal polynomial interpolating the points $(-3,-1),(-1,5),(0,8),(2,-1)$.

Solution. (Newton, direct approach; again, for comparison) The interpolating polynomial in Newton form is

$$p(x) = c_0 + c_1(x+3) + c_2(x+3)(x+1) + c_3(x+3)(x+1)x.$$

We use the four points to solve for the coefficients c_i:

$$\begin{aligned}
(-3,-1) &: c_0 = -1 \\
(-1,5) &: \underbrace{c_0}_{-1} + 2c_1 = 5 \quad \Longrightarrow \quad c_1 = 3 \\
(0,8) &: \underbrace{c_0}_{-1} + 3\underbrace{c_1}_{3} + 3c_2 = 8 \quad \Longrightarrow \quad c_2 = 0 \\
(2,-1) &: \underbrace{c_0}_{-1} + 5\underbrace{c_1}_{3} + 15\underbrace{c_2}_{0} + 30c_3 = -1 \quad \Longrightarrow \quad c_2 = -\frac{1}{2}
\end{aligned}$$

Hence, $p(x) = -1 + 3(x+3) - \frac{1}{2}(x+3)(x+1)x = -\frac{1}{2}x^3 - 2x^2 + \frac{3}{2}x + 8$.

Solution. (Newton, divided differences)

	$f[\cdot]$	$f[\cdot,\cdot]$	$f[\cdot,\cdot,\cdot]$	$f[\cdot,\cdot,\cdot,\cdot]$
-3	-1			
		$\frac{5-(-1)}{-1-(-3)} = 3$		
-1	5		$\frac{3-3}{0-(-3)} = 0$	
		$\frac{8-5}{0-(-1)} = 3$		$\frac{-\frac{5}{2}-0}{2-(-3)} = -\frac{1}{2}$
0	8		$\frac{-\frac{9}{2}-3}{2-(-1)} = -\frac{5}{2}$	
		$\frac{-1-8}{2-0} = -\frac{9}{2}$		
2	-1			

Accordingly, reading the coefficients from the top edge of the triangle (as shaded above), the Newton form is

$$p(x) = -1 + 3(x+3) - \frac{1}{2}(x+3)(x+1)x = -\frac{1}{2}x^3 - 2x^2 + \frac{3}{2}x + 8,$$

in agreement with what we had computed earlier.

Example 76. (homework) Determine the minimal polynomial interpolating $(0,-1),(2,1),(3,8)$.

Solution. (Lagrange) The interpolating polynomial in Lagrange form is:

$$\begin{aligned} p(x) &= -1\frac{(x-2)(x-3)}{(0-2)(0-3)} + 1\frac{(x-0)(x-3)}{(2-0)(2-3)} + 8\frac{(x-0)(x-2)}{(3-0)(3-2)} \\ &= -\frac{1}{6}(x-2)(x-3) - \frac{1}{2}x(x-3) + \frac{8}{3}x(x-2) \\ &= 2x^2 - 3x - 1 \end{aligned}$$

Solution. (Newton, direct approach) The interpolating polynomial in Newton form is

$$p(x) = c_0 + c_1(x-0) + c_2(x-0)(x-2).$$

We use the three points to solve for the coefficients c_i:

- $(0,-1)$: $c_0 = -1$.
- $(2,1)$: $\underbrace{c_0}_{-1} + 2c_1 = 1$, so that $c_1 = 1$.
- $(3,8)$: $\underbrace{c_0}_{-1} + 3\underbrace{c_1}_{1} + 3c_2 = 8$, so that $c_2 = 2$.

Hence, $p(x) = -1 + 1(x-0) + 2(x-0)(x-2) = 2x^2 - 3x - 1$.

Solution. (Newton, divided differences)

$$\begin{array}{lccc} 0\colon & -1 & & \\ & & \frac{1-(-1)}{2-0} = 1 & \\ 2\colon & 1 & & \frac{7-1}{3-0} = 2 \\ & & \frac{8-1}{3-2} = 7 & \\ 3\colon & 8 & & \end{array}$$

Accordingly, reading the coefficients from the top edge of the triangle, the Newton form is

$$p(x) = -1 + 1(x-0) + 2(x-0)(x-2) = 2x^2 - 3x - 1.$$

Example 77. (homework) Repeat the previous example with the additional point $(1,-2)$.

Solution. (Newton, divided differences) Notice how only the shaded entries are new.

$$\begin{array}{lcccc} 0\colon & -1 & & & \\ & & \frac{1-(-1)}{2-0} = 1 & & \\ 2\colon & 1 & & \frac{7-1}{3-0} = 2 & \\ & & \frac{8-1}{3-2} = 7 & & \frac{2-2}{1-0} = 0 \\ 3\colon & 8 & & \frac{5-7}{1-2} = 2 & \\ & & \frac{-2-8}{1-3} = 5 & & \\ 1\colon & -2 & & & \end{array}$$

Since the point $(1,-2)$ is on the graph of $2x^2 - 3x - 1$, we obtained the same final polynomial. If we had added a point not on the graph, then we would have found a degree 3 polynomial interpolating the total of four points.

Important comment. This is a considerable advantage for many practical purposes since often one does not know a priori how many interpolation points to use.

Example 78. Python numpy and scipy are powerful scientific libraries for Python. While numpy provides core functionality, scipy implements more specialized routines such as interpolation.

```
>>> from numpy import linspace, pi, sin
```

```
>>> from scipy import interpolate
```

Comment. We previously used the sin function available in the math Python standard library. The numpy library offers its own sin function with additional features. For instance, try sin([1,2]). This evaluates $\sin(x)$ at both $x=1$ and $x=2$. On the other hand, this results in an error with the sin function not from numpy.

Let us interpolate $f(x)=\sin(x)$ using 3 points, namely $x_0=0$, $x_1=\frac{\pi}{2}$, $x_2=\pi$. We begin by making lists of the x and y values as follows:

```
>>> xpoints = [0, pi/2, pi]
```

```
>>> ypoints = [sin(x) for x in xpoints]
```

Comment. As pointed out in the previous comment, we can even simply use ypoints = sin(xpoints). (The result would be a numpy array instead of a standard list but, for basic purposes, these behave alike. The numpy library introduces and uses arrays for additional features and performance for scientific computations.)

Let us check that xpoints and ypoints hold the expected values:

```
>>> xpoints
```

```
[0, 1.5707963267948966, 3.141592653589793]
```

```
>>> ypoints
```

```
[0.0, 1.0, 1.2246467991473532e-16]
```

We now ask scipy to create the interpolating polynomial:

```
>>> poly = interpolate.lagrange(xpoints, ypoints)
```

The resulting polynomial can be evaluated at any other point (such as $x=\pi/4$) and we can access its coefficients (which tell us that the polynomial is approximately $-0.41x^2+1.27x$):

```
>>> poly(pi/4)
```

```
0.75
```

```
>>> sin(pi/4)
```

```
0.7071067811865475
```

```
>>> poly.coeffs
```

```
[-0.40528473456935116, 1.2732395447351628, 0.0]
```

Homework. Show that the exact interpolation polynomial is $\frac{4}{\pi}x-\frac{4}{\pi^2}x^2$.

Finally, let us plot the sine function together with the polynomial interpolation. In the code below we use matplotlib, a powerful and widely used plotting library.

```
>>> import matplotlib.pyplot as plt
```

```
>>> xplot = linspace(0, pi, 100)
```

```
>>> plt.plot(xpoints, ypoints, 'o', xplot, sin(xplot), '-', xplot, poly(xplot), ':')
```

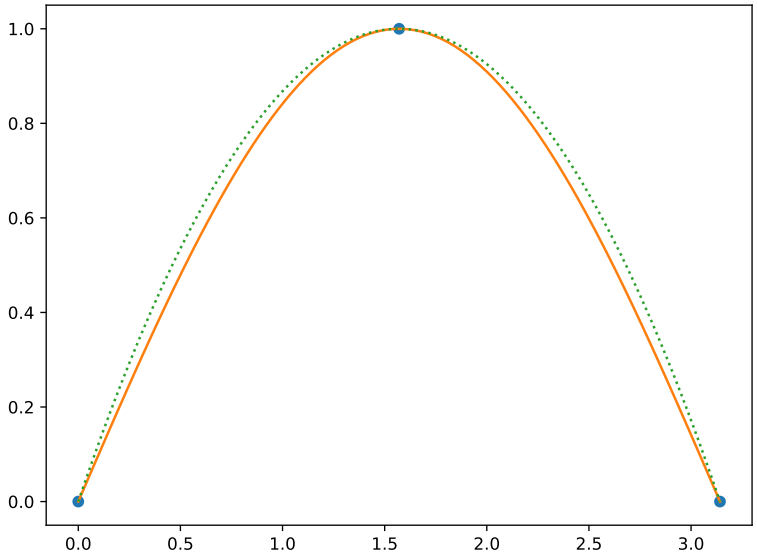
```
>>> plt.show()
```

Comment. Note that we are making three plots in one line here (namely, we plot the three points, we plot sine, and we plot the polynomial interpolation).

To plot just sine, simplify the plot command to plt.plot(xplot, sin(xplot), '-'). The '-' connects the 100 points (with x-coordinates from xplot) by a line. Replace it, for instance, with 'r-.' to get a red dotted line.

https://matplotlib.org/stable/tutorials/introductory/pyplot.html

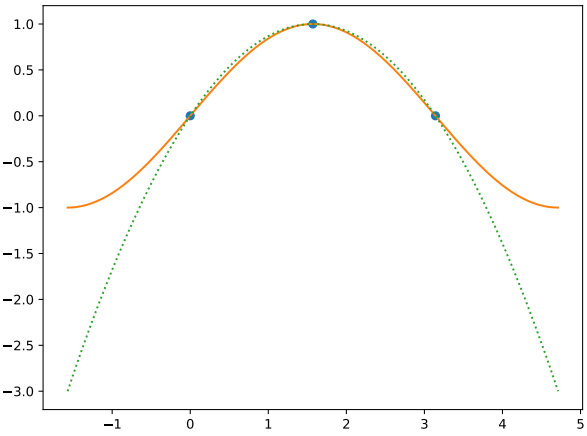
The resulting output should look as follows:

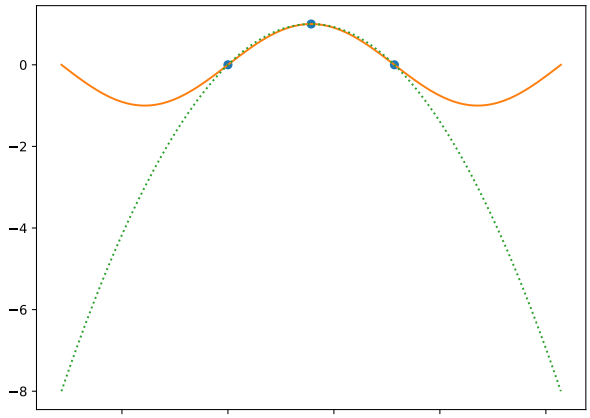


This shows pretty decent interpolation on the interval $[0, \pi]$.

Which function is which?! (You can tell from the fact that we dotted one graph or from the plots below.)

On the other hand, here are the same plots on $\left[-\frac{\pi}{2}, \frac{3\pi}{2}\right]$ and $[-\pi, 2\pi]$:





Homework. Adjust our code above (only the line `linspace(0, pi, 100)` needs to be changed) to produce these two plots.

As we can see (and as we probably expected), the polynomial interpolation does not approximate the sine function outside the interval $[0, \pi]$.

Comment. Given the three interpolation points $0, \pi/2, \pi$, an attempt to approximate the function at values much less than 0 or much larger than π (that is, outside of the range of our data) is typically referred to as **extrapolation**.

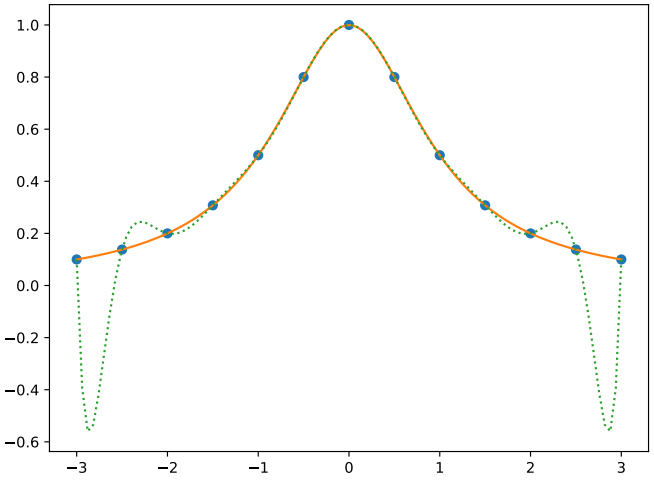
Python assignment #3

The goal of this assignment is to observe **Runge's phenomenon**.

https://en.wikipedia.org/wiki/Runge%27s_phenomenon

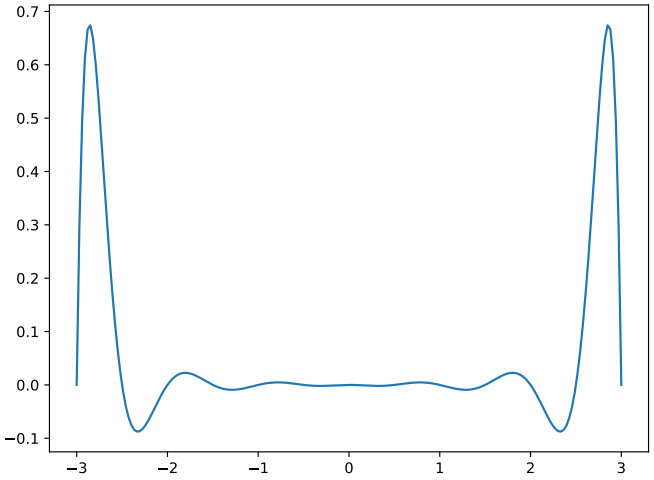
Example 79. We wish to interpolate the function $f(x) = \frac{1}{1+x^2}$ at the equally spaced points $x_0, x_1, \ldots, x_{12} = -3, -2.5, -2, \ldots, 2.5, 3$. Let $p(x)$ be the resulting polynomial interpolation. Proceed as we did in Example 78.

(a) Plot both $p(x)$ and $f(x)$ on the interval $[-3, 3]$. The output should look as follows:



Observe the problematic oscillations towards the boundaries of the interval $[-3, 3]$.

(b) Plot the error $f(x) - p(x)$ on the interval $[-3, 3]$. The output should look as follows:



(c) What is $p\left(\frac{1}{3}\right)$? For comparison, what is $f\left(\frac{1}{3}\right)$?

Solution. $p\left(\frac{1}{3}\right) \approx 0.9019025435658377$ and $f\left(\frac{1}{3}\right) = 0.9$.

(d) Using 100 equally spaced points, approximate the maximum error on the interval $[-3, 3]$ if $f(x)$ is approximated by $p(x)$.

If `poly` is the interpolating polynomial and if you defined a function called `f`, then this error is:

```
>>> max([abs(f(x)-poly(x)) for x in linspace(-3,3,100)])
```

Example 80. Repeat the previous example using instead the points

$$x_j = 3\cos\left(\frac{2j+1}{26}\pi\right), \quad j = 0, \ldots, 12.$$

Note that this is the same number of points as before but that they are no longer equally spaced.

After you have submitted your code on Replit, send me an email with the following:

(a) The approximation of the maximum error when using equally spaced points.

(b) The approximation of the maximum error when using the not equally spaced points.

(c) A sentence or two on what you observe when comparing the plots of the errors $f(x) - p(x)$.

Finally, a pointer and a hint:

- First run the code corresponding to Example 78 to make sure that things are working.

  Initially, the last two lines (responsible for plotting) are commented. That allows you to interact with the code in the Console. For instance, run the code; then enter `poly(pi/4)` into the Console to see the value of the interpolating polynomial at $\pi/4$ (this value should be 0.75).

  On the other hand, to see the plot, uncomment those two lines. If you now run the code, the command `plt.show()` opens a little output window with the plot. We then need to click the **Stop** button after we are done admiring the plot.

- We should not enter the x-coordinates of our interpolation points by hand. Instead, check out the following example:

```
>>> [-2 + 0.5*j for j in range(9)]

   [-2.0, -1.5, -1.0, -0.5, 0.0, 0.5, 1.0, 1.5, 2.0]
```

  Comment. You can also try `linspace(-2, 2, 9)` or `arange(-2, 2.5, 0.5)` (both of these must be imported from `numpy`) for a similar result.