Example 84. 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)

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

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

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

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

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 86. 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), ':')
```

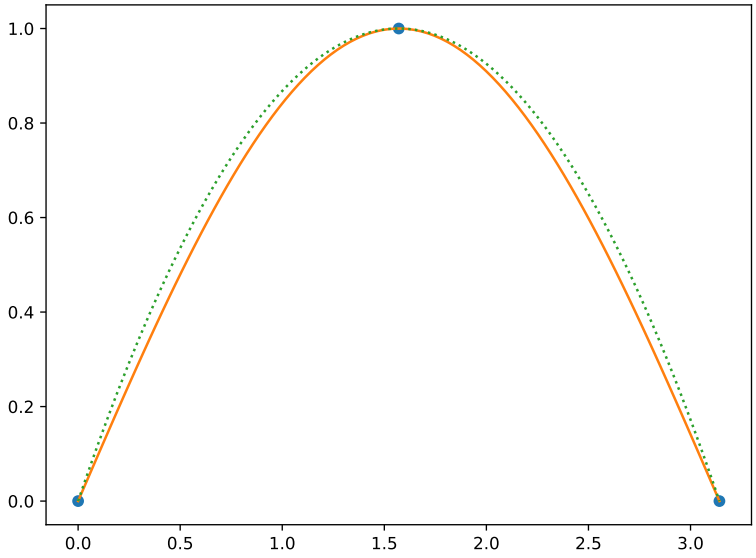
```
>>> plt.show()
```

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

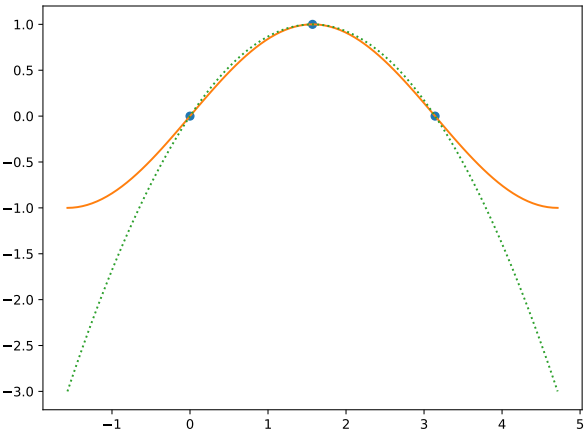
The resulting output should look as follows:

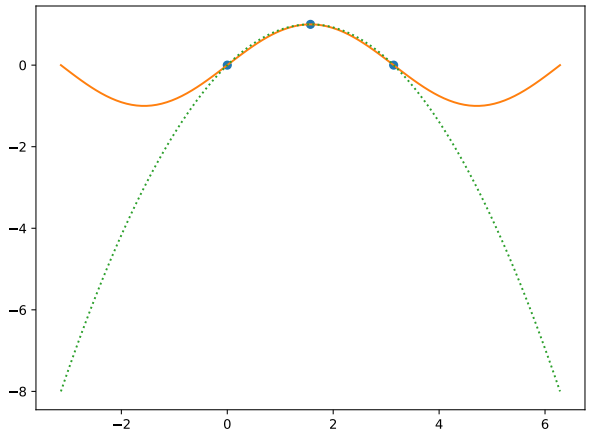


This shows pretty decent interpolation on the interval $[0, \pi]$.

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 well 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**.