Numerical Methods: Labwork 4 Report

Nguyễn Gia Phong–BI9-184

October 25, 2019

1 Curve Fitting Problems

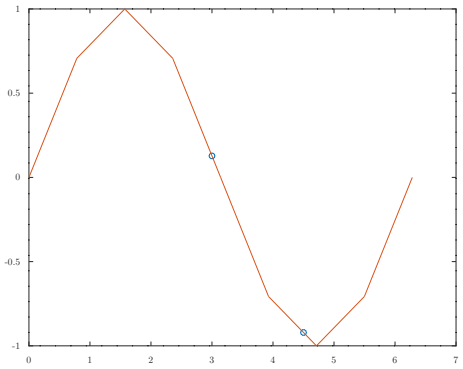
3. From the given table, we define the two vectors

```
octave> x = [0.00000 0.78540 1.57080 2.35620 ...
>            3.14159 3.92699 4.71239 5.49779 6.28319];
octave> fx = [0.00000 0.70711 1.00000 0.70711 ...
>             0.00000 -0.70711 -1.00000 -0.70711 0.00000];
```

(a) `f(3.00000)` and `f(4.50000)` can be interpolated by

```
octave> points = [3.00000 4.50000];
octave> linear = interp1 (x, fx, points)
linear =
   0.12748  -0.92080
```

To further illustrate this, we can then plot these point along with the linearly interpolated line: `plot (points, linear, "o", x, fx)`



(b) For convenience purposes, we define a thin wrapper around `interp1`

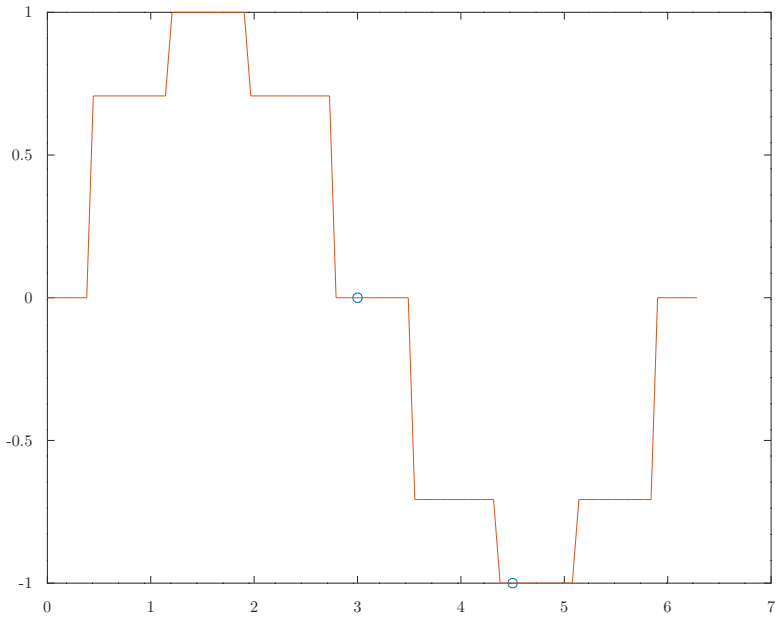
```
octave> interpolate = @(X, method) interp1 (
> x, fx, X, method, "extrap");
```

Anonymous function had to be used because named functions somehow do not support closure. Now we can use `interpolate (points, method)` to approximate `f(3.00000)` and `f(4.50000)` and obtain the table below

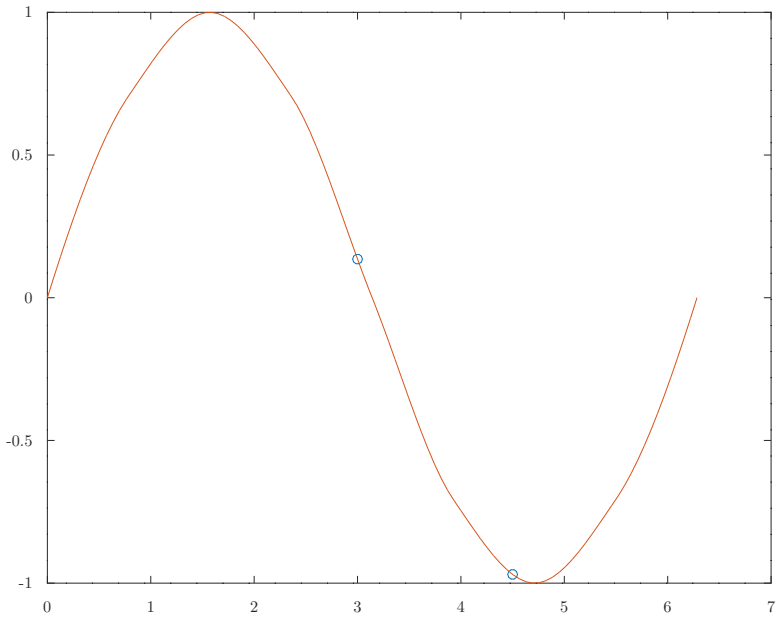
method	nearest	cubic	spline
f(3.00000)	0	0.13528	0.14073
f(4.50000)	-1	-0.96943	-0.97745

Next, we use some plots to better visualize these interpolation methods.

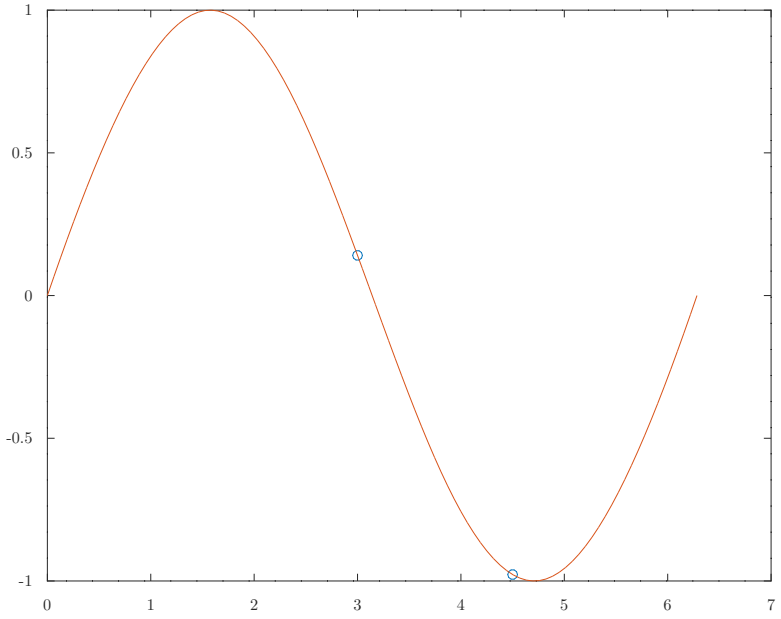
```
octave> interplot = @(mark, line, method) plot (
> mark, interpolate (mark, method), "o",
> line, interpolate (line, method));
octave> B = linspace (x(1), x(end));
octave> interplot (points, B, "nearest")
```



```
octave> interplot (points, B, "cubic")
```



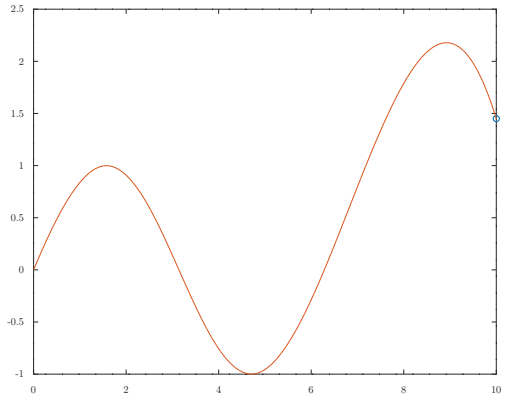

```
octave> interplot (points, B, "spline")
```



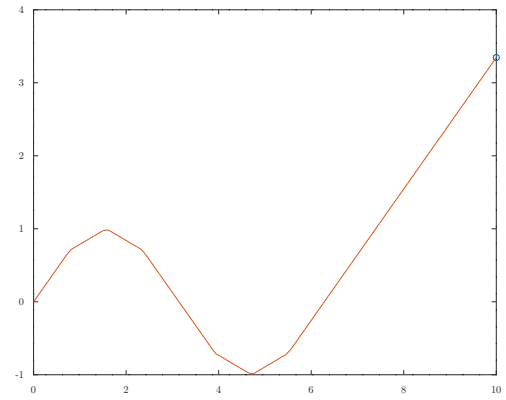
One can easily notice while `nearest` simply chooses the nearest neighbor, `cubic` and `spline` both try to *smoothen* the curve. This leads to the fact that `nearest`'s approximations strays from `linear`'s in the opposite dirrection when compared to the other two's. It also explains why `cubic`'s and `spline`'s results are quite close to each other.

(c) Since we are already extrapolating (by providing the `extrap` argument to `interp1`), interpolating for `f(10)` is rather straightforward:

```
octave> interpolate (10, "spline")
ans =  1.4499
octave> C = linspace (0, 10);
octave> interplot (10, C, "spline")
```

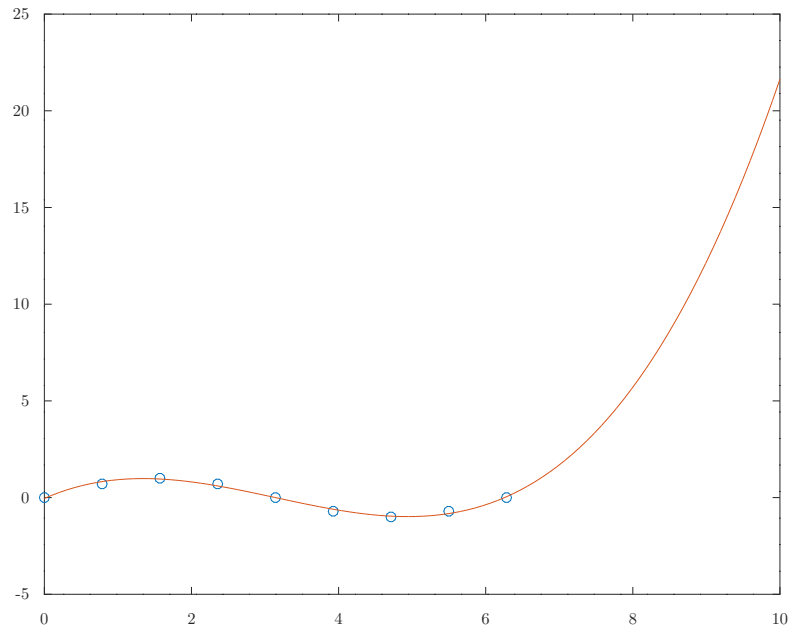



```
octave> interpolate (10, "linear")
ans =  3.3463
octave> interplot (10, C, "linear")
```



From the existing data, we can make a guess that `f` is a cubic function and regression fits quite well:

```
octave> p = polyfit (x, fx, 3)
p =
   0.084488  -0.796282   1.681694  -0.043870
octave> polyval (p, 10)
ans =  21.633
octave> plot (x, fx, "o", C, polyval (p, C))
```



In all these cases, due to the missing data, the value of `f` at 10 tends to go *wild*, i.e. far away from the given data in `fx`. If anything, the interpolated/extrapolated ones looks more harmonic, while regression simply fit the curve into the function of the given form. It is not obvious that either technique is better is this case, since the amount of given data is too small.