2. Fourier points and interpolants

```
ATAPformats
```

Any interval $[a, b]$ can be scaled to any other interval $[c, d]$. Since we shall be mainly dealing with periodic functions, most of the time, we shall just talk about $[-\pi, \pi]$. We will also make use of Chebfun's default interval $[-1, 1]$.

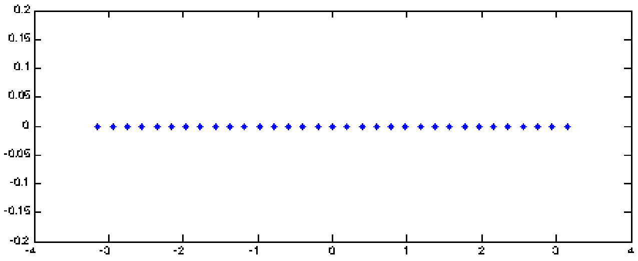
Let n be a positive integer:

```
n = 32;
```

Consider $n + 1$ equally spaced points $\{\theta_j\}$ from $-\pi$ to π:

$$\theta_j = -\pi + 2\pi j/n, \quad 0 \le j \le n.$$

```
tt = linspace(-pi, pi, n+1);
plot( tt, 0*tt, '.'), ylim([-.2, .2])
```

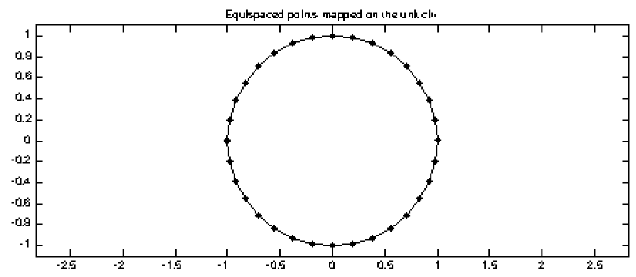


The equally spaced points in the θ-plane are the **Fourier points**. We can think of these as the arguments of $n + 1$ points $\{z_j\}$ on the unit circle in the complex plane. More precisely, if we consider the mapping,

$$z = e^{i\theta}, \tag{2.1}$$

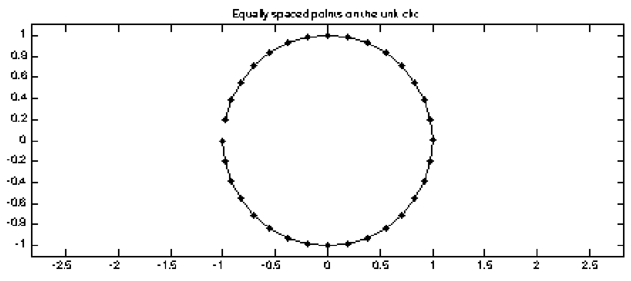
then the points θ_j are mapped to (n)th roots of unity lying in the closed unit circle, rotated through by an angle of $-\pi$ or π.

```
zz = exp(1i*tt);
hold off, plot(zz,'.-k'), axis equal, ylim([-1.1 1.1])
title('Equispaced points mapped on the unit circle')
```



We can see that the first and the last point are the same in complex plane: the mapping $z = e^{i\theta}$ wraps the interval $[-\pi, \pi]$ around the circle and the two ends meet, making $-\pi$ and and π the same points in the z-plane and therefore, to make sure that the points are distinct, we remove the last point. This is precisely what the Chebfun command `fourpts` does. It takes the argument n and the domain on which the points are to be distributed and returns exactly n equally spaced points such that no point is duplicated when mapped onto the unit circle in the complex plane.

```
tt = fourpts(n, [-pi, pi]);
zz = exp(1i*tt);
hold off, plot(zz,'.-k'), axis equal, ylim([-1.1 1.1])
title('Equally spaced points on the unit circle')
```



Let $\{f_j\}$, $0 \leq j \leq n-1$, be a set of numbers, which may or may not come from sampling a function $f(\theta)$ at the Fourier points. Then there exists a unique trigonometric polynomial of degree k or less, living on the interval $[-\pi, \pi]$, which interpolates these data, i.e., $p(\theta_j) = f_j$ for each j. By a trigonometric polynomial of **degree** k, we mean a linear combination of sines and cosines with the higest frequency being k or equivalently, a linear combination of copmlex exponentials with maximum frequencies being k and $-k$. Trigonometric polynomials which interpolate data take slighly different forms depending upon whether n, the number of interpolation points, is even or odd.

Let us first consider the case for odd number of points, $n = 2k+1$ for some non-negative integer k. Let P_k denote the vector space of trigonometric polynomials of degree k or lower, living on $[-\pi, \pi]$, spanned by the following set of n basis vectors:

$$\{1, \cos\theta, \sin\theta, \cos 2\theta, \ldots, \sin k\theta, \cos k\theta\}. \tag{2.2}$$

Equivalently, we can represent the vector space P_k by using a set of n complex exponentials of the form:

$$\{e^{-ik\theta}, \ldots, e^{-i\theta}, 1, e^{i\theta}, \ldots, e^{ik\theta}\}. \tag{2.3}$$

We trust that the reader already knows that a theorem gurantees the existence and uniqueness of the itnerpolant from the space P_k for any distinct set of n interpolation points.

If the number of interpolation points is even, i.e., $n = 2k$, for some positive integer k, the situation is somewhat complicated. Let us consider an example with $n = 2$ points. Let us say the interpolation points are $-\pi/2$ and $\pi/2$. If we assume our basis to be $\{1, \cos\theta\}$, we will not be able to solve the interpolation problem since $\cos(\pm\pi/2) = 0$ and the associated Vandermonde matrix is singular. However, the basis $\{1, \sin\theta\}$ will work in this case. This basis on the other hand will fail if we change our interpolation points to say $0, \pi$ since $\sin j\pi = 0$ for any integer j. But for this case the basis $\{1, \cos\theta\}$ will provide a unique solution to the interpolation problem. This observation leads to the following basis for the space P_k, when n is even:

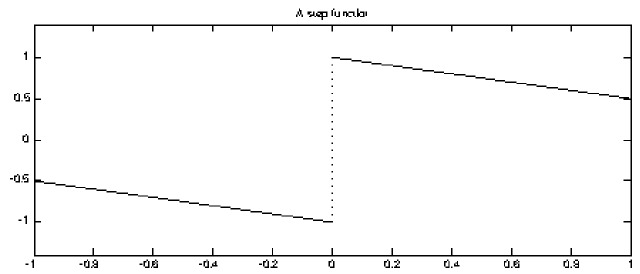
$$\{1, \cos\theta, \sin\theta, \ldots, \cos(k-1)\theta, \sin(k-1)\theta, \text{ one of } \cos k\theta \text{ or } \sin k\theta\}. \qquad (2.5)$$

With this little caveat for even n, the existence and uniqueness of the trigonometric interpolant from P_k for any distinct set of n interpolation points is guaranteed. In case the interpolation points are Fourier points, we call the trigonometric polynomial the **Fourier interpolant**.

One can construct sets of non-equispaced points for which trigonometric interpolants of periodic functions have terrible properties, as we shall see in later chpaters. Trigonometric interpolants of periodic functions through equally spaced points, however, are excellent. It is the uniform spacing and the periodic nature of the function being interpolated that makes the difference. The explanation of this fact has a lot to do with potential theory, a subject we shall introduce in a later chapter. Specifically, what makes equally spaced points effective is that each one experiences the same *force* by other points when placed on the circle. If we imagine these points as electrons constrained to move on the unit circle, their equilibrium position will be a set of equally spaced points on the unit circle.

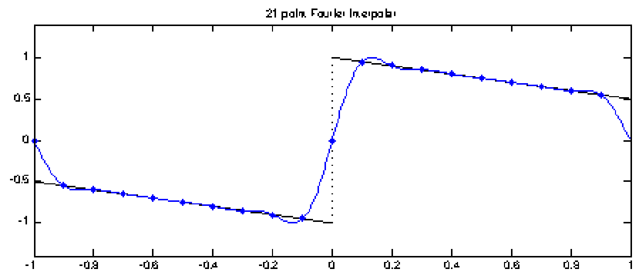
Chebfun is built on Chebyshev interpolants. For example, here is a certain step function:

```
x = chebfun('x');
f = @(x) sign(x) - x/2;
hold off, plot(f(x),'k'), ylim([-1.4 1.4])
title('A step function')
```



By calling `chebfun` with a second explicit argument of 20 and the `'periodic'` flag, we can construct the Fourier interpolant to f through 20 points:

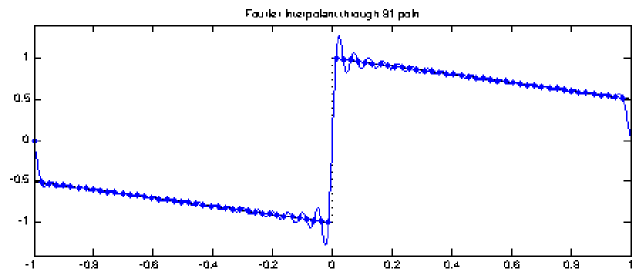
```
p = chebfun( f, 20, 'periodic');
hold on, plot(p,'.-'), ylim([-1.4 1.4])
title('21 point Fourier interpolant')
```



Notice the behaviour at the left end point. Chebfun does not interpolate the function at the left end point but interpolates the mean value $(f(1)+f(-1))/2$, which is 0 in this case.

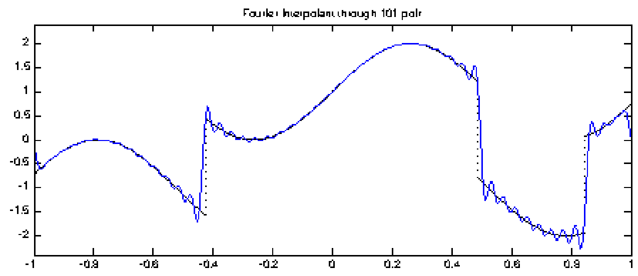
Similarly, here is the Chebyshev interpolant through 81 points:

```
hold off, plot(f(x),'k')
p = chebfun( f, 81, 'periodic');
hold on, plot(p,'.-'), ylim([-1.4 1.4])
title('Fourier interpolant through 81 points')
```



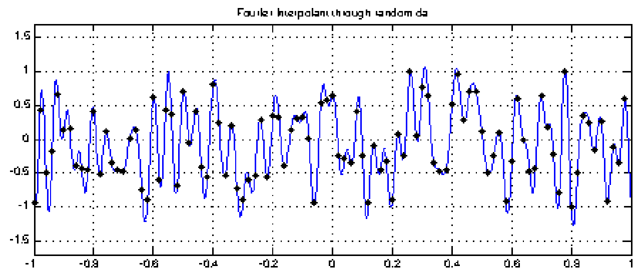
Here are a more complicated function and its Fourier interpolant in 101 points:

```
f = @(x) sin(6*x) + sign(sin((x + exp(2*x))));
hold off
plot(f(x), 'k');
p = chebfun(f, 101, 'periodic');
hold on, plot(p), ylim([-2.4 2.4])
title('Fourier interpolant through 101 points')
```



Another way to use the `chebfun` command with the `'periodic'` flag is by giving it an explicit vector of data rather than a function to sample, in which case it interprets the vector as data for a Fourier interpolant of the appropriate order. Here for example is the Fourier interpolant through 100 random data values in $[-1, 1]$:

```
p = chebfun(2*rand(100,1)-1, 'periodic');
hold off, plot(p,'-')
hold on, plot(p,'.k')
ylim([-1.7 1.7]), grid on
title('Fourier interpolant through random data')
```



This experiment illustrates how robust Fourier interpolation is. If we had taken a million points instead of 100, the result would not have been much different mathematically, though it would have been a mess to plot. We shall return to this figure in a later chapter.

For illustrations like these it is interesting to pick data with jumps or wiggles, and a later chapter discusses such interpolants systematically. In applications where trigonometric interpolants are most useful, however, the data will typically be smooth and periodic.

Summary of Chapter 2. *Trigonometric interpolants in non-equispaced points may have very poor approximation properties, but trigonometric interpolants in Fourier points, which are equally spaced, are excellent.*