The Discrete Fourier Expansion

In this lecture, we will explore the expansion of a function into an approximate Fourier series. That is, suppose we have a periodic function

$$f(\theta) = \sum_{k=-\infty}^{\infty} \hat{f_k} e^{ik\theta}$$

for

$$\hat{f_k} \triangleq \frac{1}{2\pi} \int_0^{2\pi} f(\theta) e^{-ik\theta} d\theta,$$

where we assume the coefficients $\hat{f_k}$ decay sufficiently fast, so that

$$\sum_{k=-\infty}^{\infty} |\hat{f_k}|.$$

(It's beyond the scope of this course, but this condition guarantees that the sum converges to f.) We will approximate the function by a finite-dimensional expansion

$$f(\theta) \approx \sum_{k=\alpha}^{\beta} \hat{f}_k^n e^{ik\theta}$$

where the approximate coefficients are calculated using the Trapezium rule:

$$\hat{f}_k^n = \frac{1}{2\pi} Q_n[f(\theta) e^{-ik\theta}]$$

for

$$Q_n[f] = \sum_{j=1}^{n} f(\theta_j).$$

Why not `quadgk` for calculating Fourier coefficients?

Why the Trapezium rule? We will see this is *not* an arbitrary choice: the Trapezium has important properties that lead to the robustness and speed of the approximation. To emphasize this point, let's consider an alternative: just use `quadgk`. The following sets up a function `f̂gk(k)` where each coefficient of `f` is approximated by `quadgk`:

In [105]:

```
f=θ->exp(cos(θ-0.1))

f̂gk=k->quadgk(θ->f(θ)*exp(-im*k*θ),0,2π)[1]/(2π)
```

Out[105]:

```
(anonymous function)
```

We can then evaluate the sum

$$\sum_{k=-8}^{8} \tilde{f}_k e^{ik\theta} :$$

where $\tilde{f}_k$ is calculated using `f̂gk`:

In [106]:

```
ret=0.

for k=-8:8
    ret += f̂gk(k)*exp(im*k*0.1)
end

ret-f(0.1)
```

Out[106]:

```
-1.1613424000245232e-8 + 0.0im
```

Unforunately, this very quickly breaks down, requiring an inexorbitant amount of time to evaluate:

In [108]:

```
@time f̂gk(-9.)
```

```
  8.623647 seconds (130.00 M allocations: 2.901 GB, 25.49% gc tim
e)
```

Out[108]:

```
3.4302835642957293e-9 + 4.3226999974636545e-9im
```

Trapezium rule for calculating Fourier coefficients

Instead of `quadgk`, we use the Trapezium rule:

In [109]:

```
function trap(f::Function,a,b,n)
    h=(b-a)/n
    x=linspace(a,b,n+1)

    v=f(x)
    h/2*v[1]+sum(v[2:end-1])*h+h/2*v[end]
end

trap(f::Function,n) = trap(f,0,2π,n)
```

Out[109]:

```
trap (generic function with 3 methods)
```

The following creates a function `f̂(k,n)` that returns the Trapezium rule approximation to the kth Fourier coefficient, $\hat{f}_k^n$.

In [41]:

```
f̂=(k,n)->trap(θ->f(θ).*exp(-im*k*θ),n)/(2π)
```

Out[41]:

```
(anonymous function)
```

We see for sufficiently large n, we recover the coefficients accurately:

In [110]:

```
f̂(-1,20)-f̂gk(-1)
```

Out[110]:

```
0.0 + 6.938893903907228e-17im
```

But this quickly scales up to to large n, for approximating

$$f(\theta) \approx \sum_{k=-\beta}^{\beta} \hat{f}_k^n e^{ik\theta}.$$

In [113]:

```
β=100

n=2β+1

ret=0.

for k=-β:β
    ret += f̂(k,n)*exp(im*k*0.1)
end


ret-f(0.1)
```

Out[113]:

```
4.707345624410664e-14 - 1.1832913578315177e-30im
```

We will create a routine called `dft` that returns the approximate Fourier coefficients

$$[\hat{f}^n_\alpha, \hat{f}^n_{\alpha+1}, \ldots, \hat{f}^n_\beta]^\top$$

In [114]:

```
function dft(f,α,β)
    n=β-α+1
    Complex128[
        trap(θ->f(θ).*exp(-im*k*θ),0.,2π,n)/(2π)    for k=α:β
        ]
end
```

Out[114]:

```
dft (generic function with 1 method)
```

We also create a routine that allows us to easily evaluate an approximate Fourier series, where `fc` is a `Vector` containing

$$[\hat{f}^n_\alpha, \hat{f}^n_{\alpha+1}, \ldots, \hat{f}^n_\beta]^\top$$

In [115]:

```
function fours(fc::Vector,α,β,θ)
    ret=0.+0.im

    for k=α:β
        ret += fc[k-α+1]*exp(im*k*θ)
    end
    ret
end
```

Out[115]:

```
fours (generic function with 1 method)
```

Thus we have the approximation:

In [118]:

```
fc=dft(f,-10,10)
fours(fc,-10,10,0.1)-f(0.1)
```

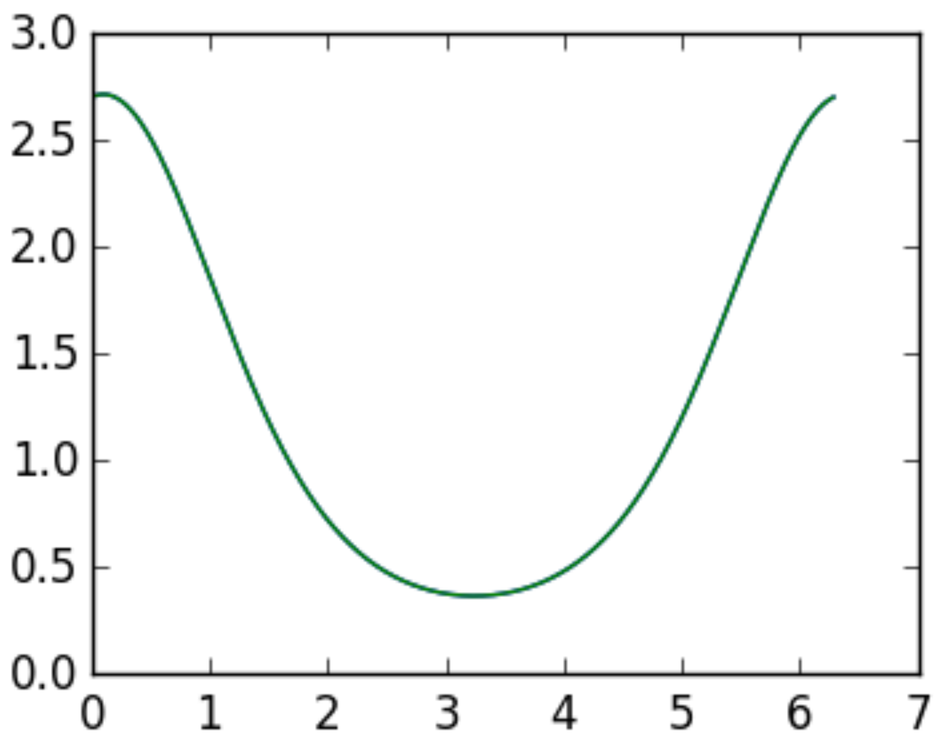
Out[118]:

```
-3.921707403264918e-11 + 0.0im
```

We can plot this approximate Fourier series:

In [120]:

```
α,β=-10,10
fc=dft(f,α,β)
using PyPlot

g=linspace(0.,2π,1000)

plot(g,real(map(θ->fours(fc,α,β,θ),g)))
plot(g,f(g));
```



Aliasing

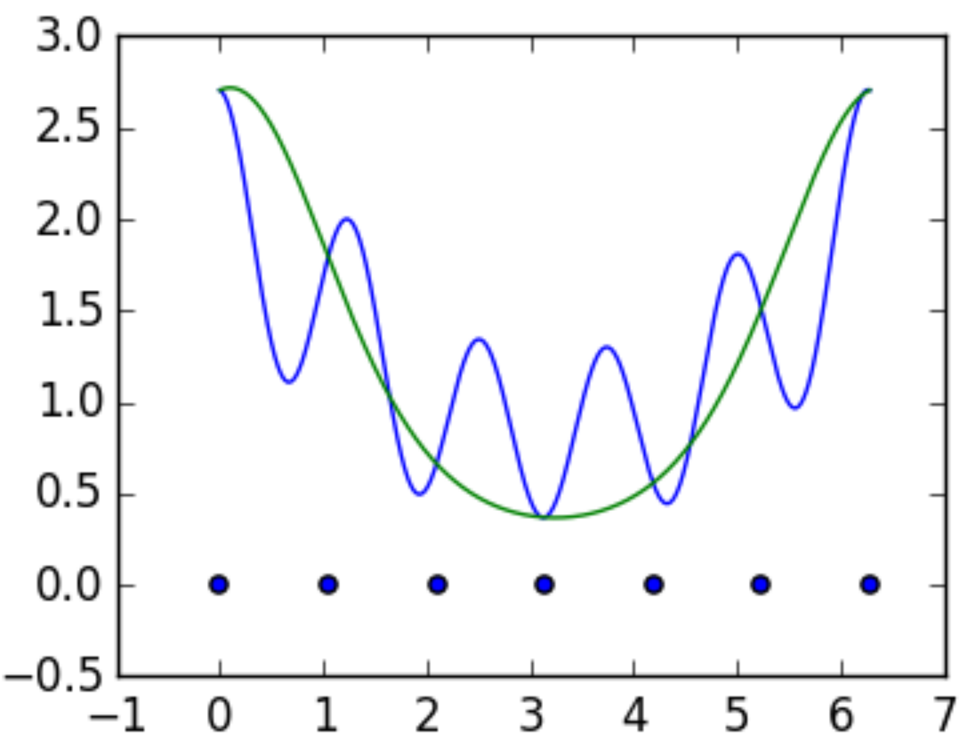
Aliasing is the observation that the discrete Fourier expansion interpolates the true function even when the α and β are not chosen to resolve the function. Here is an extreme example: let's take only positive terms:

$$f(\theta) \approx \sum_{k=0}^{n-1} \hat{f}_k^n e^{ik\theta}$$

Since the true function has both negative and positive Fourier coefficients, we cannot expect this to be accurate, as we see below:

In [122]:

```
α,β=0,5
f=θ->exp(cos(θ-0.1))
n=β-α+1



fc=dft(f,α,β)

g=linspace(0.,2π,1000)

plot(g,real(map(θ->fours(fc,α,β,θ),g)))
plot(g,f(g))

scatter(linspace(0.,2π,n+1),zeros(n+1));
```



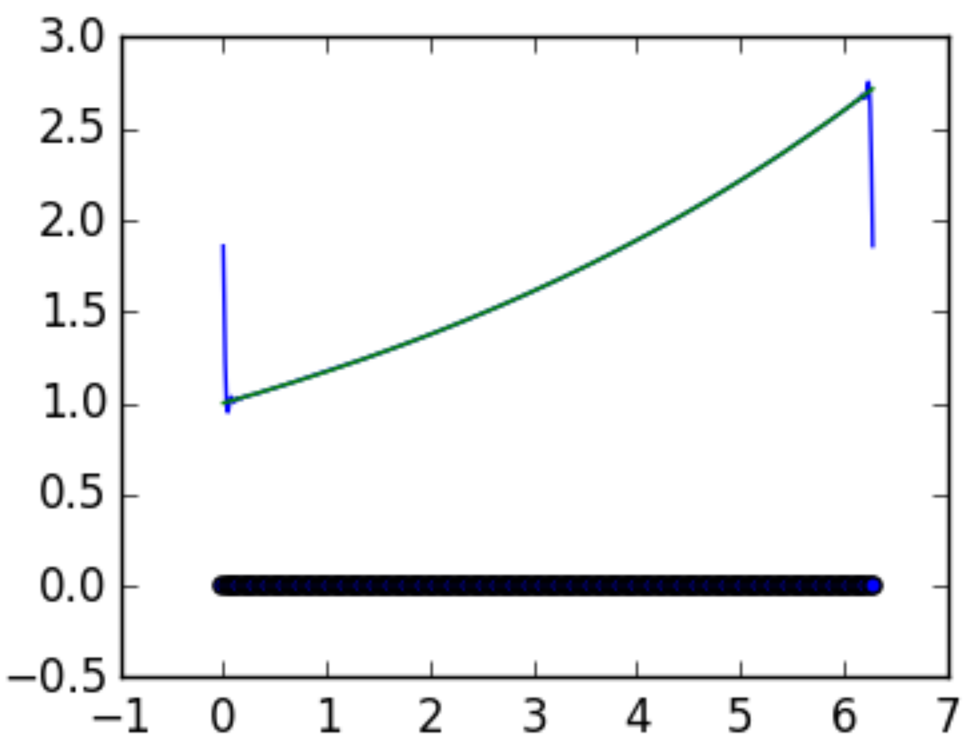
However, at each quadrature point θ_j, the blue curve equals the green curve exactly.

Gibb's Phenomenon

Gibb's phenomenon is the observation that when the function is not periodic, the approximate Fourier series overshoots:

In [124]:

```
α,β=-100,100

f=θ->exp(θ/(2π))

n=β-α+1



fc=dft(f,α,β)

g=linspace(0.,2π,1000)

plot(g,real(map(θ->fours(fc,α,β,θ),g)))
plot(g,f(g))

scatter(linspace(0.,2π,n+1),zeros(n+1));
```



Investigating this further is beyond the scope of the course.