Applications of the FFT

1. Spectral Analysis
   - convolutions with the FFT
   - filtering periodic data
   - removing low amplitude noise

2. Image Processing
   - images are matrices
   - red, green, blue intensities
   - blurring and deblurring images

MCS 472 Lecture 11
Industrial Math & Computation
Jan Verschelde, 2 February 2024

Applications of the FFT

1. Spectral Analysis
   - convolutions with the FFT
   - filtering periodic data
   - removing low amplitude noise

2. Image Processing
   - images are matrices
   - red, green, blue intensities
   - blurring and deblurring images

the DFT convolution theorem

$$
\begin{tikzcd}
(x, y) \arrow[r, "\star"] \arrow[d, "\text{DFT}"'] & x \star y \\
(\widehat{x}, \widehat{y}) \arrow[r, "\bullet"] & \widehat{x} \bullet \widehat{y} \arrow[u, "\text{iDFT}"']
\end{tikzcd}
$$

- The inverse discrete Fourier transform (iDFT)
- applied to the componentwise product $\widehat{x} \bullet \widehat{y}$
- of the discrete Fourier transforms (DFTs) $\widehat{x}$ and $\widehat{y}$,
- respectively of x and y, equals the convolution $x \star y$.

the convolution theorem applied to a filter with the FFT

$$
\begin{tikzcd}
u \arrow[r] \arrow[d, "\mathrm{FFT}"'] & \boxed{h} \arrow[r, "\star"] \arrow[d, "\mathrm{FFT}"'] & y = h \star u \\
\widehat{u} & \widehat{h} \arrow[r, "\bullet"] & \widehat{y} = \widehat{h} \bullet \widehat{u} \arrow[u, "\mathrm{iFFT}"']
\end{tikzcd}
$$

With the FFT, the convolution of two n-vectors is $O(n \log(n))$.

the convolution theorem applied to a filter with the FFT

$$
\begin{tikzcd}
u \arrow[r] \arrow[d, "\text{FFT}\ O(n\log(n))"'] & \boxed{h} \arrow[r, "\star", "O(n^2)"'] \arrow[d, "\text{FFT}\ O(n\log(n))"'] & y = h \star u \\
\widehat{u} & \widehat{h} \arrow[r, "\bullet", "O(n)"'] & \widehat{y} = \widehat{h} \bullet \widehat{u} \arrow[u, "\text{iFFT}\ O(n\log(n))"']
\end{tikzcd}
$$

With the FFT, the convolution of two n-vectors is $O(n\log(n))$.

Applications of the FFT

1. Spectral Analysis
   - convolutions with the FFT
   - filtering periodic data
   - removing low amplitude noise
2. Image Processing
   - images are matrices
   - red, green, blue intensities
   - blurring and deblurring images

filtering periodic data

The $O(n \log(n))$ time of the Fast Fourier Transform allows for the online filtering of data.

We distinguish between three types of filters:

1. *low pass:* only low frequencies pass,
2. *high pass:* only high frequencies pass, and
3. *band pass:* only frequencies within a band pass.

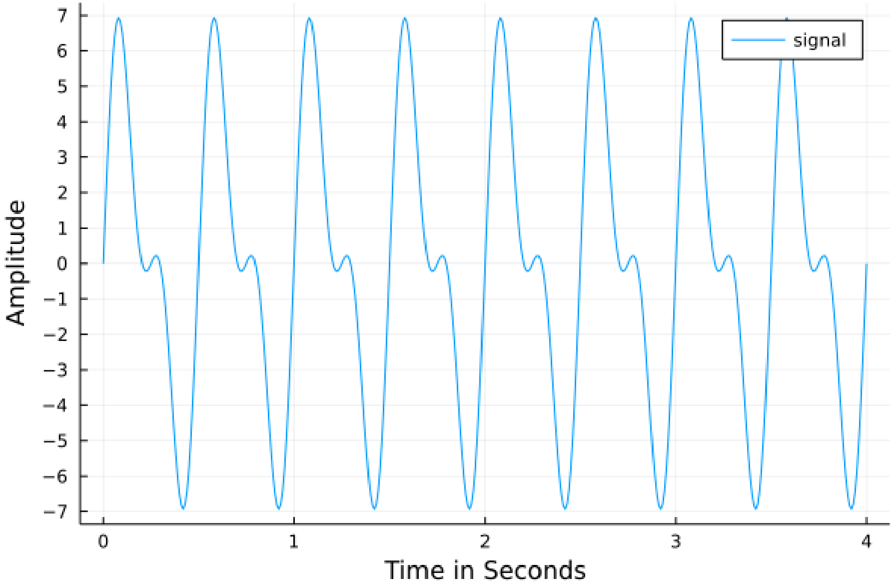
Filtering with the FFT in three steps:

1. transform the input data to the frequency domain,
2. remove the components of unwanted frequencies, and
3. transform the filtered data to the time domain.

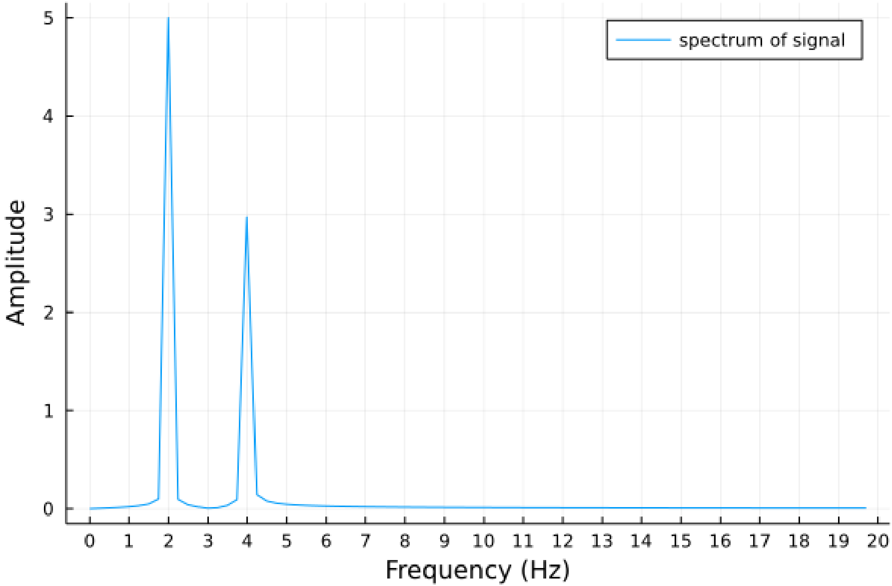
an experiment in Julia

```
using Plots
using FFTW

dt = 0.01
t = 0:dt:4
y = 3*sin.(4*2*pi*t) + 5*sin.(2*2*pi*t)
plot(t, y, yticks=-7:1:7, label="signal"
     xlabel="Time in Seconds", ylabel="Amplitude")

F = fft(y)
n = length(y)/2
amps = abs.(F)/n
freq = [0:79]/(2*n*dt)
plot(freq, amps[1:80], xticks=0:1:20,
     label="spectrum of signal",
     ylabel="Amplitude", xlabel="Frequency (Hz)")
```

amplitude versus time



amplitude versus frequency



Applications of the FFT

1. Spectral Analysis
   - convolutions with the FFT
   - filtering periodic data
   - removing low amplitude noise

2. Image Processing
   - images are matrices
   - red, green, blue intensities
   - blurring and deblurring images

adding low amplitude noise

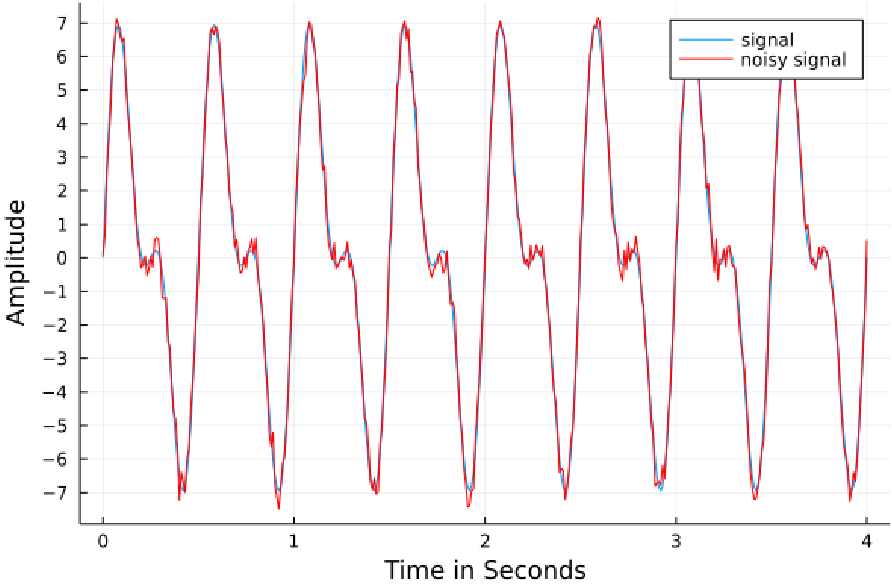
Normally distributed noise of magnitude 0.3 is added.

```
ynoise = y + 0.3*randn(length(y))
```

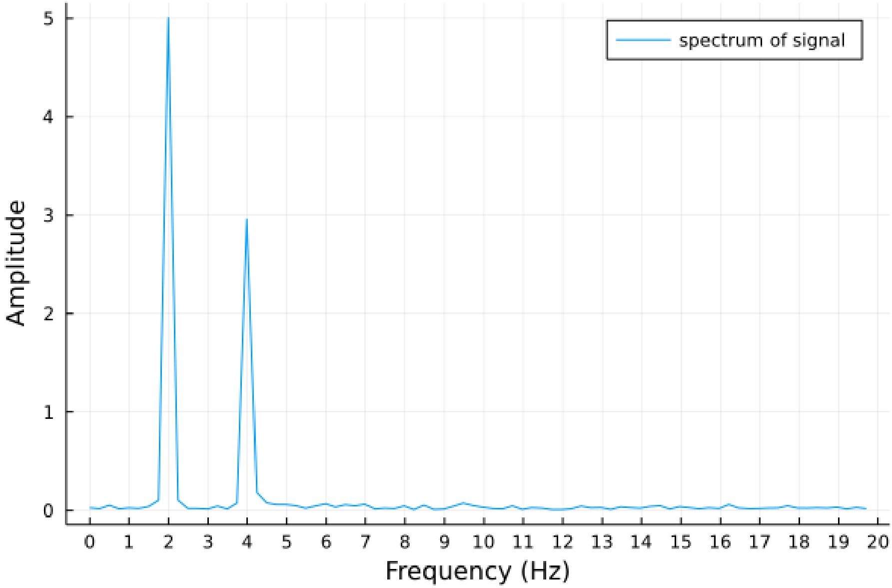
Code to make the plot:

```
dt = 0.01
t = 0:dt:4
y = 3*sin.(4*2*pi*t) + 5*sin.(2*2*pi*t)
plot(t, y, yticks=-7:1:7, label="signal"
     xlabel="Time in Seconds", ylabel="Amplitude")
plot!(t, ynoise, color="red", label="noisy signal")
```

the noisy signal



the spectrum of the noisy signal



removing the low amplitudes

Observe on the spectrum of the noisy signal:
the noisy appears for all frequencies, but at low amplitude.

Code to remove the low amplitudes:

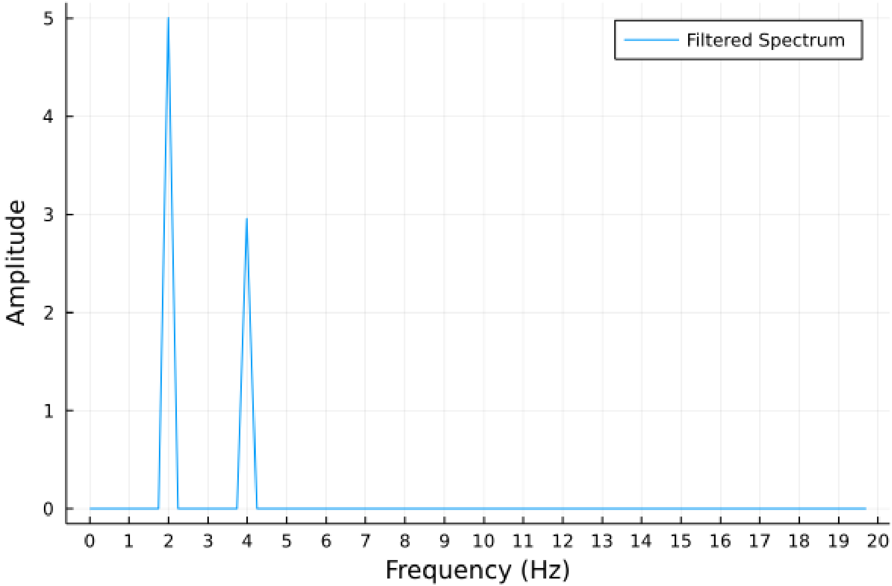
```
filteredF = [x*Int(abs(x) > 50) for x in F];
```

All numbers in magnitude less than 50 are replaced by zero.

The semicolon (`;`) suppresses the output.

Finding the good threshold requires inspecting the numbers.

the filtered spectrum



applying the inverse FFT

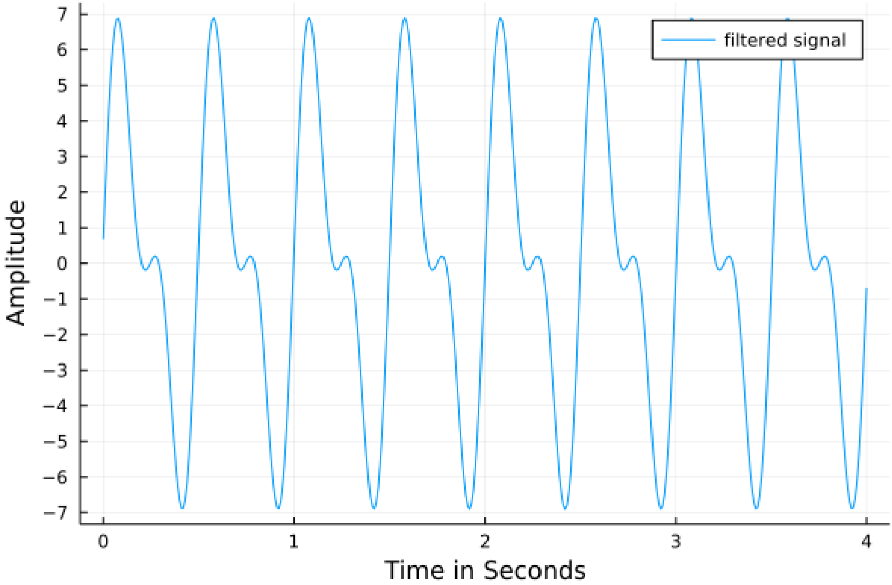
To reconstruct the signal, we apply the inverse FFT.

```
yfiltered = ifft(filteredF)
```

To plot the filtered signal,
we plot only the real part of the output of the `ifft`.

```
plot(t, real(yfiltered),
     yticks=-7:1:7,
     label="filtered signal",
     xlabel="Time in Seconds",
     ylabel="Amplitude")
```

the filtered signal



filtering unwanted frequencies

With the FFT and iFFT we can remove unwanted frequencies: in the spectrum, set the amplitudes for the unwanted frequencies to zero.

Exercise 1:
Make a signal with three components:

1. the first sine has amplitude 5 and runs at 2Hz,
2. the second sine has amplitude 3 and runs at 8Hz, and
3. the third sine has amplitude 1 and runs at 16Hz.

Use this signal to demonstrate the application of the FFT for three types of filters:

1. Low pass: remove all frequencies higher than 6Hz.
2. High pass: remove all frequencies lower than 10Hz.
3. Band pass: keep the frequencies between 6Hz and 10Hz.

Applications of the FFT

1. Spectral Analysis
   - convolutions with the FFT
   - filtering periodic data
   - removing low amplitude noise

2. Image Processing
   - images are matrices
   - red, green, blue intensities
   - blurring and deblurring images

a familiar image

images are matrices of RGB codes

```
using Images

A = load("buildingsky.png")

size(A)
```

The `load` displays the image.

The output of `size(A)` is $(570, 855)$,
so `A` has 570 rows and 855 columns.

selecting the middle of the image

```
m1 = Int(size(A,1)/2)
m2 = Int(round(size(A,2)/2))

Amiddle = A[m1-20:m1+20,m2-20:m2+20]
```

Applications of the FFT

1. Spectral Analysis
   - convolutions with the FFT
   - filtering periodic data
   - removing low amplitude noise
2. Image Processing
   - images are matrices
   - red, green, blue intensities
   - blurring and deblurring images

Red, Green, Blue intensities

Let us look at one element of the matrix.

```
a = A[1,1]; typeof(a)
```

shows `RGB{N0f8}`.
The `RGB` type stores the Red, Green, Blue intensities.

The output of `dump(a)` is

```
RGB{N0f8}
  r: N0f8
    i: UInt8 0x81
  g: N0f8
    i: UInt8 0xd3
  b: N0f8
    i: UInt8 0xfb
```

computing with intensities

The components of `a` are `a.r`, `a.g`, and `a.b`,
for the red, green, and blue intensities.

Convert components to floats, and we can compute with the intensities:

```
agray = RGB{Float32}((Float32(a.r)
                    + Float32(a.g)
                    + Float32(a.b))/3)
```

Averaging the intensities lead to a grayscale picture.

converting to grayscale

```
Agray = zeros(size(A,1), size(A,2))

Agray = Matrix{RGB{Float32}}(Agray)

for i=1:size(A,1)
    for j=1:size(A,2)
        a = A[i,j]
        b = RGB{Float32}((Float32(a.r)
                        + Float32(a.g)
                        + Float32(a.b))/3)
        Agray[i,j] = b
    end
end
```

the grayscale picture

a matrix of floats

In the grayscale image, all intensities are the same.

The matrix of RGB codes is the converted as below:

```
C = zeros(size(Agray,1), size(Agray,2))

for i=1:size(Agray,1)
    for j=1:size(Agray,2)
        C[i,j] = Agray[i,j].r
    end
end
```

The grayscale matrix is used for the remaining computations.

If we want to work with colors, then we can work with three different matrices, one matrix of each intensity.

making images greener

Exercise 2:

Take an image, for example our familiar picture,

and give the operations to increase the green intensities by 10%.

Applications of the FFT

1. Spectral Analysis
   - convolutions with the FFT
   - filtering periodic data
   - removing low amplitude noise

2. Image Processing
   - images are matrices
   - red, green, blue intensities
   - blurring and deblurring images

blurring images for safe transmission

Let X be an m-by-n matrix, representing an image.

- B is a blur matrix.
- Compute $Y = B \star X$ to blur the image stored in X, the $\star$ is the matrix-matrix multiplication.

  Because of the blurring Y is safe for transmission.
- To deblur the image, do $X = B^{-1} \star Y$.

Two computational problems:

1. The matrix-matrix multiplication $\star$ costs $O(n^3)$.
2. Computing the inverse B^{-1} also costs $O(m^3)$.

two dimensional Fourier transforms

X is an m-by-n matrix.

- Let ω_m be the mth primitive root: $\omega_m^m - 1 = 0$.
- Let ω_n be the nth primitive root: $\omega_n^n - 1 = 0$.

Then the discrete Fourier transform of X is $\widehat{X}$ with entries

$$\begin{aligned} \widehat{x}_{i,j} &= \sum_{p=0}^{m-1} \left(\sum_{q=0}^{n-1} x_{p,q} \omega_n^{q\,j} \right) \omega_m^{p\,i} \\ &= \sum_{q=0}^{n-1} \left(\sum_{p=0}^{m-1} x_{p,q} \omega_m^{p\,j} \right) \omega_n^{q\,i} \end{aligned}$$

The rowwise and columnwise formulas are the same
because of the distributive properties of addition with multiplication.

processing pictures of round objects

Exercise 3:

In processing medical images, the images are of round objects, e.g.: brain scans produced by Magnetic Resonance Imaging (MRI).

Search the literature for methods on the applications of the FFT on data that is not scanned on a grid that is not rectangular, but polar.

apply the FFT and use componentwise operations

Let X be the image and B the blur matrix.

$$
\begin{tikzcd}
(B, X) \arrow[d, "\text{FFT}"'] & X \\
\widehat{Y} = \widehat{B} \bullet \widehat{X} \arrow[r, "\text{send } \widehat{Y}"] & \widehat{X} = \widehat{Y} / \widehat{B} \arrow[u, "\text{iFFT}"']
\end{tikzcd}
$$

Matrix-matrix multiplications and inverse computations are avoided:

- The $\bullet$ is the componentwise multiplication, and
- the $/$ is the componentwise division.

blurring an image

```
B = randn(size(C,1), size(C,1))
Y = B*C
D = zeros(size(Y,1), size(Y,2))
D = Matrix{RGB{Float32}}(D)
for i=1:size(Y,1)
    for j=1:size(Y,2)
        a = Y[i,j]
        if a < 0
            a = 0.0
        end
        if a > 1
            a = 1.0
        end
        D[i,j]= RGB{Float32}(a,a,a)
    end
end
```

the blurred grayscale picture

extending the blur matrix

For the componentwise multiplication $\widehat{Y} = \widehat{B} \bullet \widehat{X}$,
the matrix B needs to be of the same size as X.

We extend B with ones on the diagonal and zeros off the diagonal:

```
BB = zeros(size(C,2), size(C,2))
BB[1:size(B,1),1:size(B,2)] = B
for i in size(B,1)+1:size(BB,1)
    BB[i,i] = 1.0
end
```

blurring with FFT and componentwise products

```
using FFTW

fftB = fft(BB)
fftC = fft(C)
fftY = zeros(size(C,1), size(C,2))
fftY = Matrix{Complex{Float64}}(fftY)
for i=1:size(C,1)
    for j=1:size(C,2)
        fftY[i,j] = fftB[i,j]*fftC[i,j]
    end
end
```

The `fftY` holds the blurred image, safe for transmission.

componentwise divisions

The received data is `fftY`.

To compute the original images, we first do $\widehat{X} = \widehat{Y}/\widehat{B}$, as below:

```
fftX = zeros(size(C,1), size(C,2))
fftX = Matrix{Complex{Float64}}(fftY)

for i=1:size(C,1)
    for j=1:size(C,2)
        fftX[i,j] = fftY[i,j]/fftB[i,j]
    end
end
```

application of the inverse FFT

```
X = ifft(fftX)

deblurred = zeros(size(X,1), size(X,2))
deblurred = Matrix{RGB{Float32}}(deblurred)

for i=1:size(X,1)
    for j=1:size(X,2)
        x = Float32(real(X[i,j]))
        deblurred[i,j] = RGB{Float32}(x,x,x)
    end
end
```

summary and bibliography

Two applications of the Fast Fourier Transform (FFT) were presented.
See the posted Jupyter notebooks.

The main reference for this lecture is:

- Charles R. MacCluer:
  Industrial Mathematics. Modeling in Industry, Science, and Government. Prentice Hall, 2000.

  We ended Chapter 4.

- Timothy Sauer: *Numerical Analysis*,
  second edition, Pearson, 2012.

  Chapter 10 deals with the discrete Fourier transform.

summary of the last six lectures

In the past six lectures, we provided a computational overview of signal processing and filter design.

1. The z-transform of a sequence shows the grow or decay factors.
2. Linear, time invariant, and causal filters are determined entirely by the impulse response, or the coefficients of its transfer function.
3. Bode plots show the amplitude gain and phase shift of the evaluated transfer function.
4. The Discrete Fourier Transform (DFT) turns convolutions into componentwise products.
5. The Fast Fourier Transform (FFT) executes the DFT in $O(n \log(n))$ time.
6. Applications of the FFT include the removal of low amplitude noise; low pass, high pass, band pass filters; and image blurring.