Digital signal processing workbook

Andrey Akinshin

Contents

Module 1

Introduction

This document is a workbook for the "Digital signal processing" Coursera course (https://www.coursera.org/course/dsp). It uses the R language and the knitr package for calculation and plot rendering.

Help code:

```
library("ggplot2")

## Loading required package:  methods

library("grid")
library("gridExtra")
library("phonTools")
# Discrete signal plot
dsplot <- function(x, y, xlab="", ylab="") {
  d <- data.frame(x = x, y = y, y0 = rep(0, length(y)))
  p <- ggplot() +
    geom_segment(data=d, mapping=aes(x=x, y=y0, xend=x, yend=y), size=1, color="red") +
    geom_point(data=d, mapping=aes(x=x, y=y), size=2, shape=21, fill="red") +
    scale_x_continuous(name=xlab) +
    scale_y_continuous(name=ylab)
  p
}
# Continuous signal plot
csplot <- function(x, y, xlab="", ylab="") {
  d <- data.frame(x = x, y = y)
  p <- ggplot(data=d, aes(x=x, y=y)) +
    geom_line(color="red") +
    scale_x_continuous(name=xlab) +
    scale_y_continuous(name=ylab)
  p
}
# dft plot
dftplot <- function(x, xlab="", onlyhalf=F) {
  n <- 1:length(x)
  if (onlyhalf)
    n <- 1:(length(x)/2)
  X <- fft(x)
  grid.arrange(dsplot(n, round(Re(X),4),  ylab="Re(X[n])", xlab=xlab),
               dsplot(n, round(Im(X),4),  ylab="Im(X[n])", xlab=xlab), ncol=2)
  grid.arrange(dsplot(n, round(Mod(X),4), ylab="Mod(X[n])", xlab=xlab),
```

```
              dsplot(n, round(Arg(X),4), ylab="Arg(X[n])", xlab=xlab), ncol=2)
}
# Discrete-Time Fourier Transform
dtft <- function(x) {
  func <- function(w) {
    sum(x * exp(-1i * w * (1:length(x))))
  }
  Vectorize(func)
}
# Discrete-Time Fourier Transform plot
dtftplot <- function(x, xlab="", ylab="") {
  func <- dtft(x)
  t <- seq(-pi, pi, by=0.01)
  value <- func(t)
  grid.arrange(csplot(t, round(Re(value),4),  ylab="Re(X[n])",  xlab=xlab),
               csplot(t, round(Im(value),4),  ylab="Im(X[n])",  xlab=xlab), ncol=2)
  grid.arrange(csplot(t, round(Mod(value),4), ylab="Mod(X[n])", xlab=xlab),
               csplot(t, round(Arg(value),4), ylab="Arg(X[n])", xlab=xlab), ncol=2)
}
```

Module 2

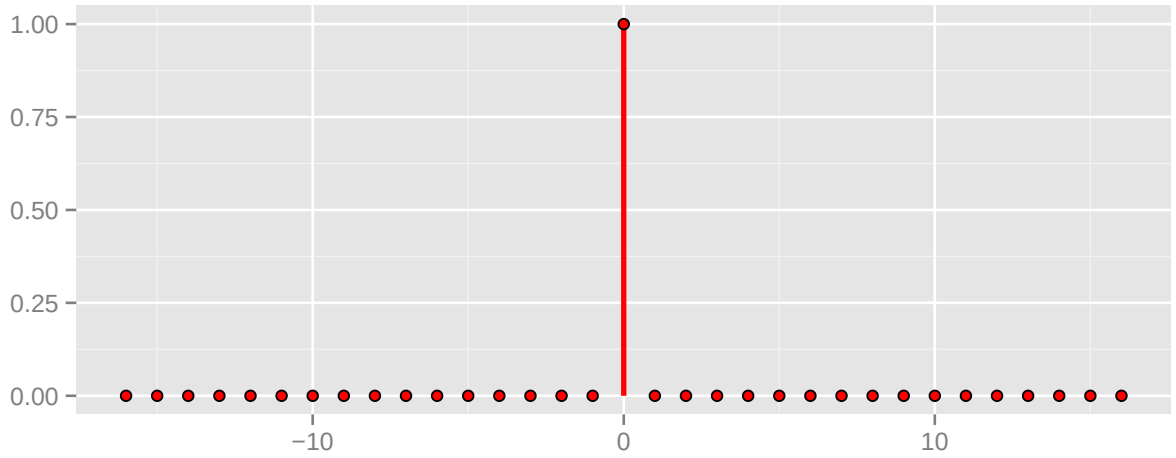
Discrete-time (DT) signals

2.1 Discrete-time signals

2.1.1 The delta signal

$$x[n] = \delta[n]$$

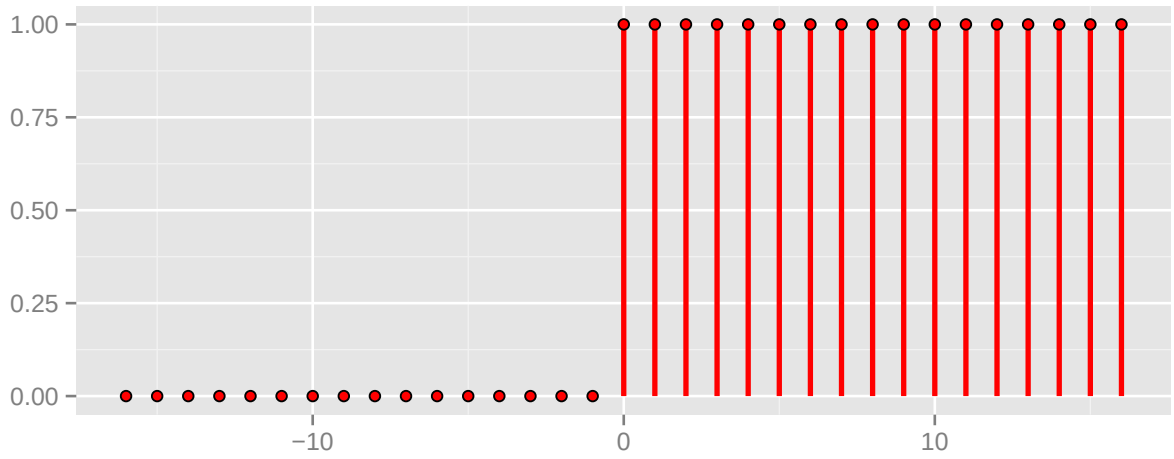
```
n <- -16:16
x <- ifelse(n == 0, 1, 0)
dsplot(n, x)
```



2.1.2 The unit step

$$x[n] = 1$$

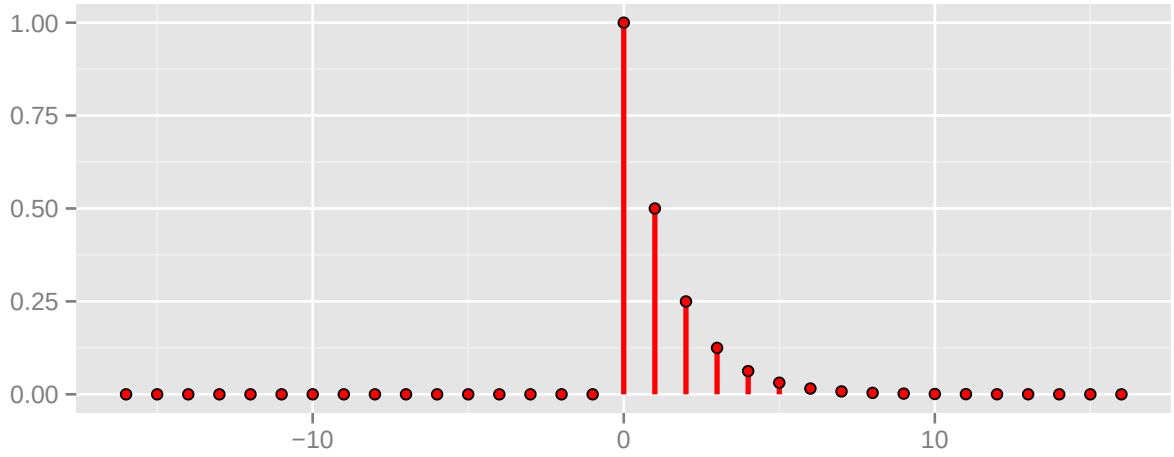
```
n <- -16:16
x <- ifelse(n >= 0, 1, 0)
dsplot(n, x)
```



2.1.3 The exponential decay

$$x[n] = |a|^n u[n], \quad |a| < 1$$

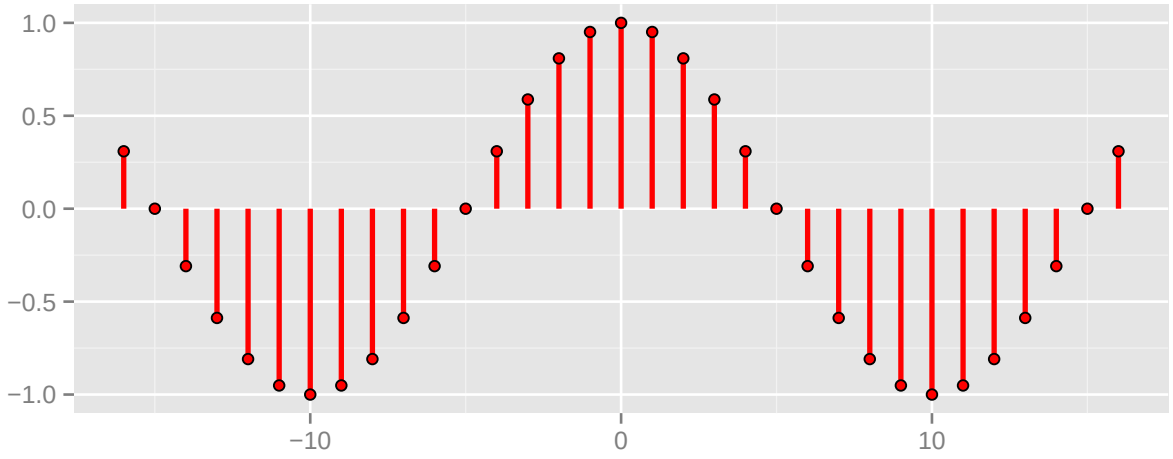
```
a <- 0.5
n <- -16:16
x <- ifelse(n >= 0, abs(a)^n, 0)
dsplot(n, x)
```



2.1.4 The sinusoid

$$x[n] = sin(\omega_0 n + \theta)$$

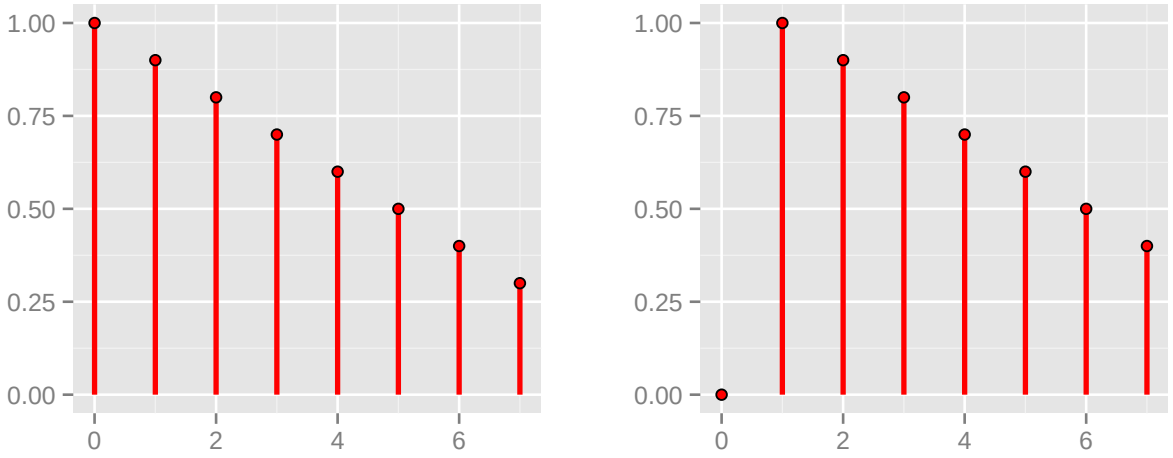
```
omega0 <- pi/10; theta <- pi/2
n <- -16:16
x <- sin(omega0 * n + theta)
dsplot(n, x)
```



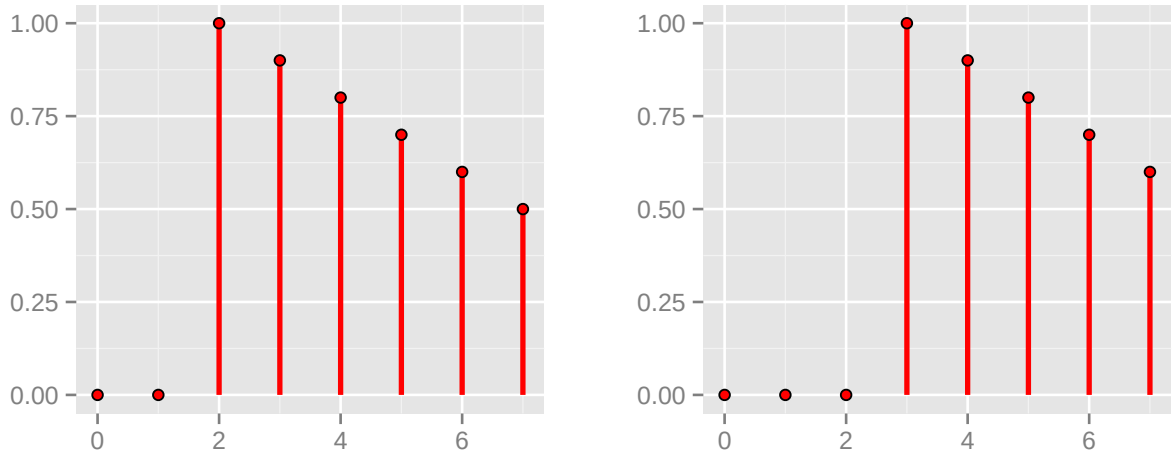
2.1.5 Shift of a finite-length: finite-support

$$\overline{x}[n] = x[n - z]$$

```
shift <- function(x, z) c(rep(0, z), x[1:(length(x)-z)])
n <- 0:7
x1 <- 1-n*0.1
x2 <- shift(x1, 1)
x3 <- shift(x1, 2)
x4 <- shift(x1, 3)
grid.arrange(dsplot(n, x1), dsplot(n, x2), ncol=2)
```



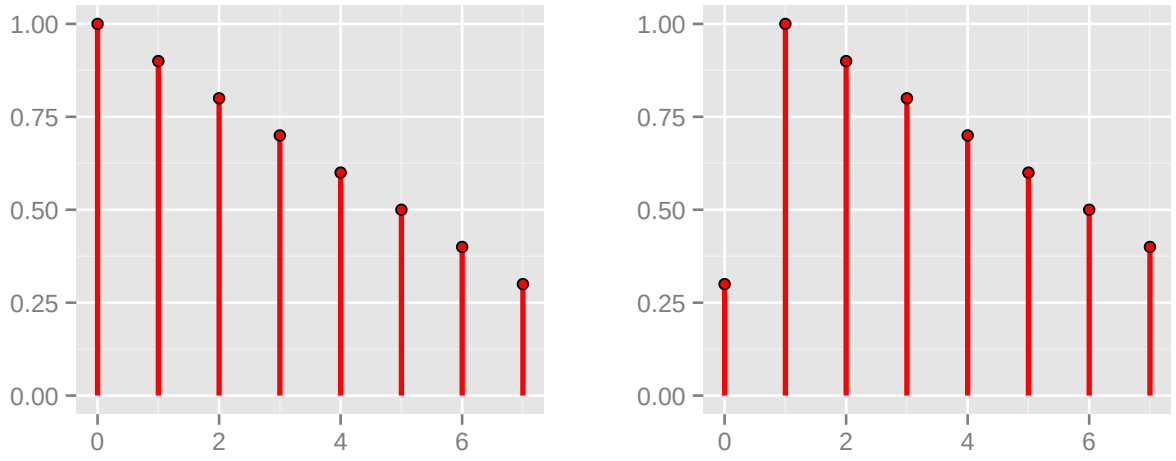
```
grid.arrange(dsplot(n, x3), dsplot(n, x4), ncol=2)
```



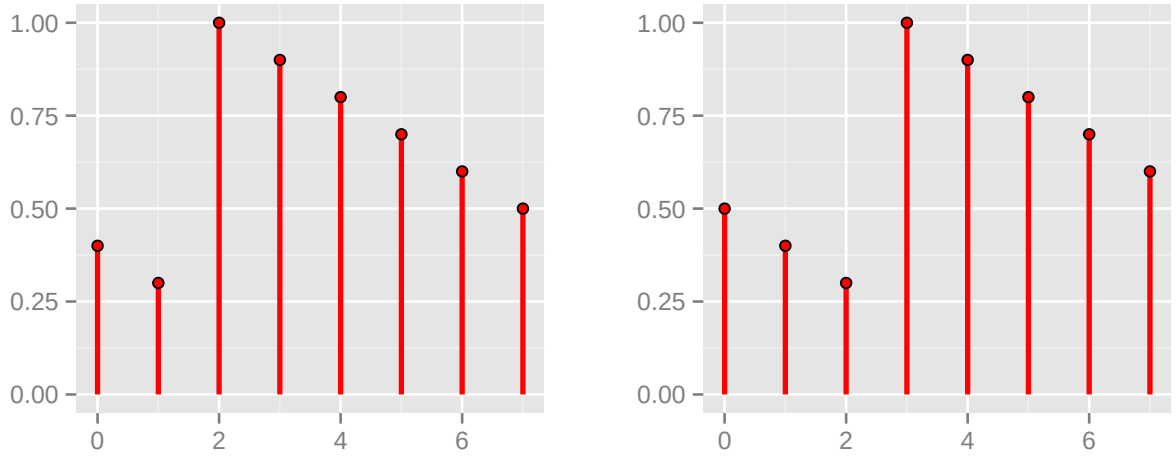
2.1.6 Shift of a finite-length: periodic extension

$$\tilde{x}[n] = x[n - z]$$

```
shift <- function(x, z) c(x[(length(x)-z+1):length(x)], x[1:(length(x)-z)])
n <- 0:7
x1 <- 1-n*0.1
x2 <- shift(x1, 1)
x3 <- shift(x1, 2)
x4 <- shift(x1, 3)
grid.arrange(dsplot(n, x1), dsplot(n, x2), ncol=2)
```




```
grid.arrange(dsplot(n, x3), dsplot(n, x4), ncol=2)
```



2.1.7 Energy and power

General formulas:

$$E_x = \sum_{n=-\infty}^{\infty} |x[n]|^2$$

$$P_x = \lim_{N\to\infty} \frac{1}{2N+1}$$

Periodic signals:

$$E_{\tilde{x}} = \infty$$

$$P_{\tilde{x}} = \sum_{n=0}^{N-1} |\tilde{x}[n]|^2$$

2.2 The complex exponential

$$x[n] = Ae^{j(\omega n+\phi)} = A\big(\cos(\omega n+\phi) + j\sin(\omega n+\phi)\big)$$

$$cos(\omega n+\phi) = a\cos(\omega n) + b\sin(\omega n), \quad a = \cos\phi,\ b = \sin\phi$$

$$Re\{e^{j(\omega n+\phi)}\} = Re\{e^{j\omega n}e^{j\phi}\}$$

$$e^{j\alpha} = \cos\alpha + j\sin\alpha$$

$$\text{Rotation:} \quad z' = ze^{j\alpha}$$

$$\text{Signal:} \quad x[n] = e^{j\omega n}; \quad x[n+1] = e^{j\omega}x[n]$$

$$\text{Signal:} \quad x[n] = e^{j\omega n+\phi}; \quad x[n+1] = e^{j\omega}x[n], \quad x[0] = e^{j\phi}$$

$$e^{j\omega n} \text{ peridoic} \Leftrightarrow \omega = \frac{M}{N}2\pi,\ M, N \in \mathbb{N}$$

$$e^{j\omega} = e^{j}(\omega + 2k\pi), \forall k \in \mathbb{N}$$

2.3 The Karplus-Strong Algorithm

Main equation:

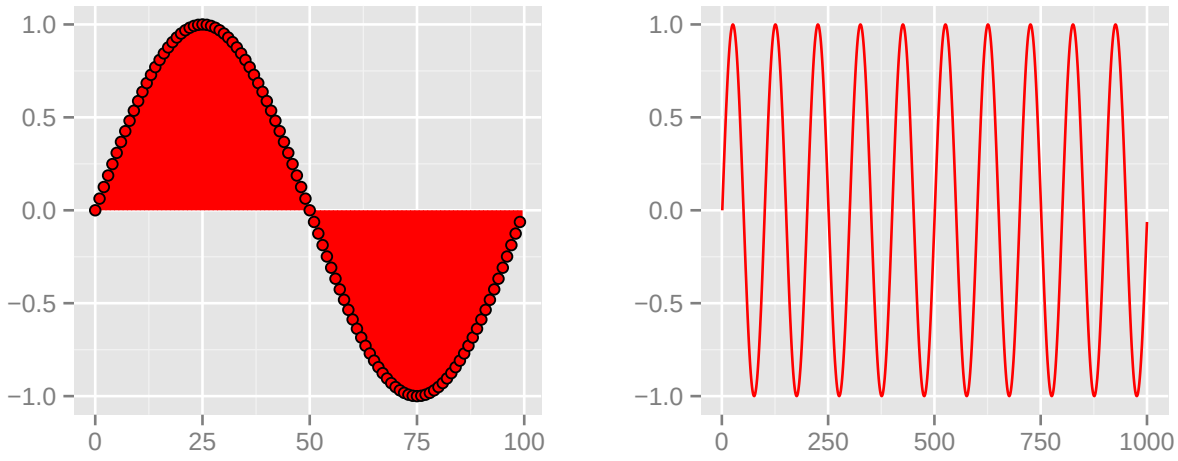
$$y[n] = \alpha y[n - M] + x[n]$$

```
ks <- function(x, M, alpha, N) {
  y <- c()
  for (n in 1:N) {
    yn <- alpha * ifelse(n > M, y[n-M], 0) +
          ifelse(n <= length(x), x[n], 0)
    y <- c(y, yn)
  }
  y
}
```

2.3.1 Sine wave

$M = 100, \alpha = 1, \overline{x}[n] = \sin(2\pi n/100)$ for $0 \leq n < 100$ and zero elsewhere

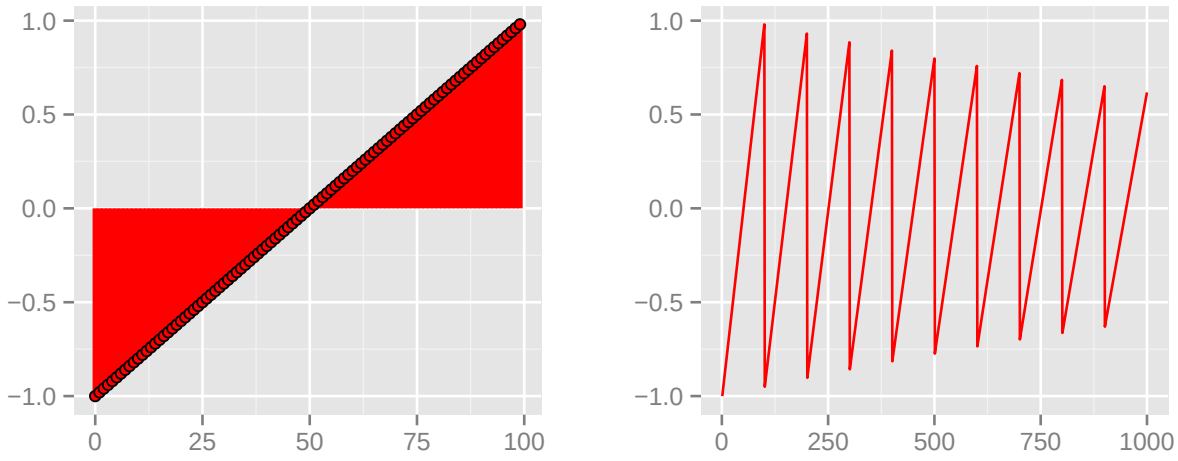
```
M <- 100; alpha <- 1; n <- 0:99
x <- sin(2*pi*n/100)
y <- ks(x, M, alpha, 1000)
grid.arrange(dsplot(n, x), csplot(1:length(y), y), ncol=2)
```



2.3.2 Proto-violin

$M = 100, \alpha = 0.95, \overline{x}[n]$: zero-mean sawtooth wave between 0 and 99, zero elsewhere

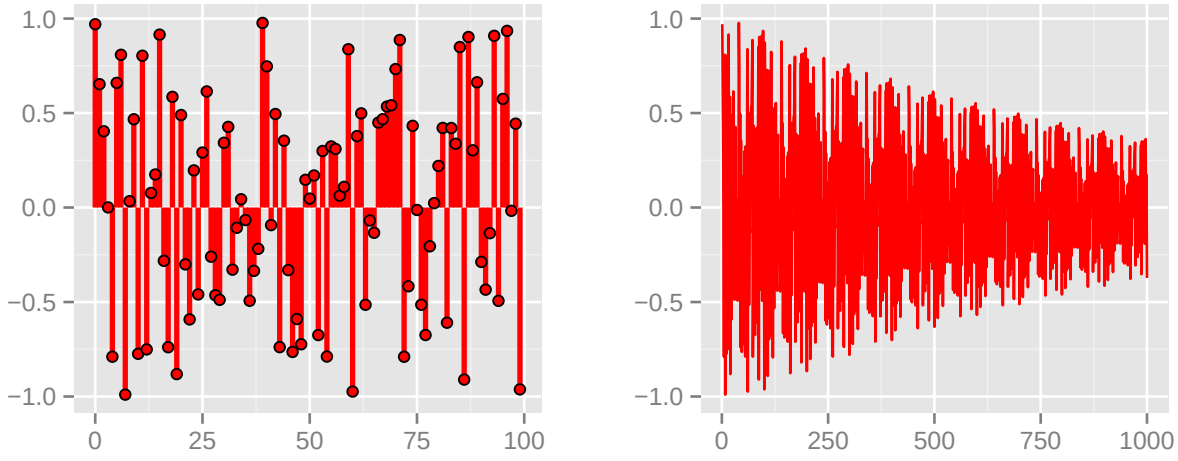
```
M <- 100; alpha <- 0.95; n <- 0:99
x <- -1 + 2*n/100
y <- ks(x, M, alpha, 1000)
grid.arrange(dsplot(n, x), csplot(1:length(y), y), ncol=2)
```



2.3.3 Random

$M = 100, \alpha = 0.9, \overline{x}[n]$: 100 random values between 0 and 99, zero elsewhere

```
M <- 100; alpha <- 0.9; n <- 0:99
x <- runif(100, -1, 1)
y <- ks(x, M, alpha, 1000)
grid.arrange(dsplot(n, x), csplot(1:length(y), y), ncol=2)
```



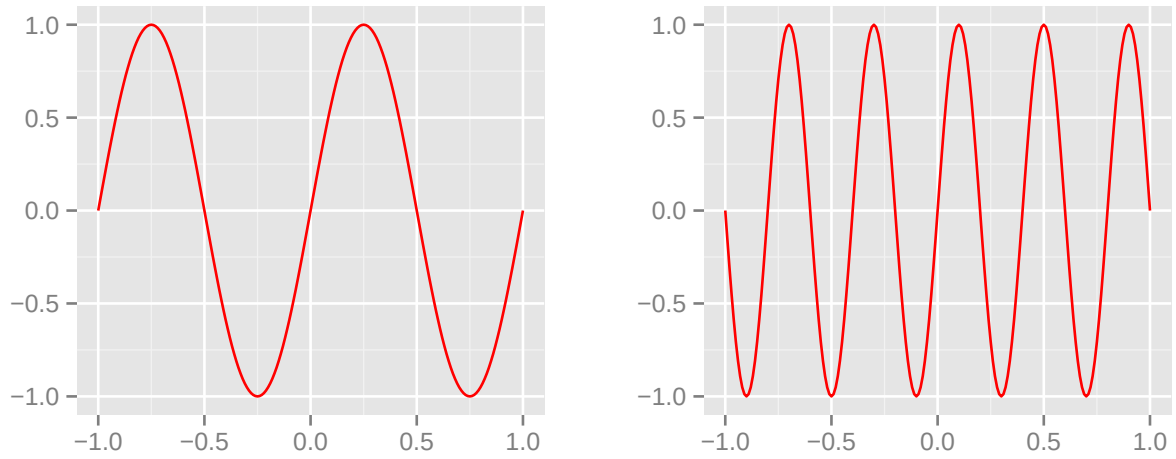
Module 3

Euclid and Hilbert

3.1 From Euclid to Hilbert

3.1.1 Space of square-integrable functions over $[-1,1]$: $l_2([-1,1])$

```
t <- seq(-1, 1, by=0.01)
f1 <- 2*pi; x1 <- sin(f1 * t)
f2 <- 5*pi; x2 <- sin(f2 * t)
grid.arrange(csplot(t, x1), csplot(t, x2), ncol=2)
```



$$x^{(1)} = \sin(f_1 t), \quad f_1 = 2\pi$$

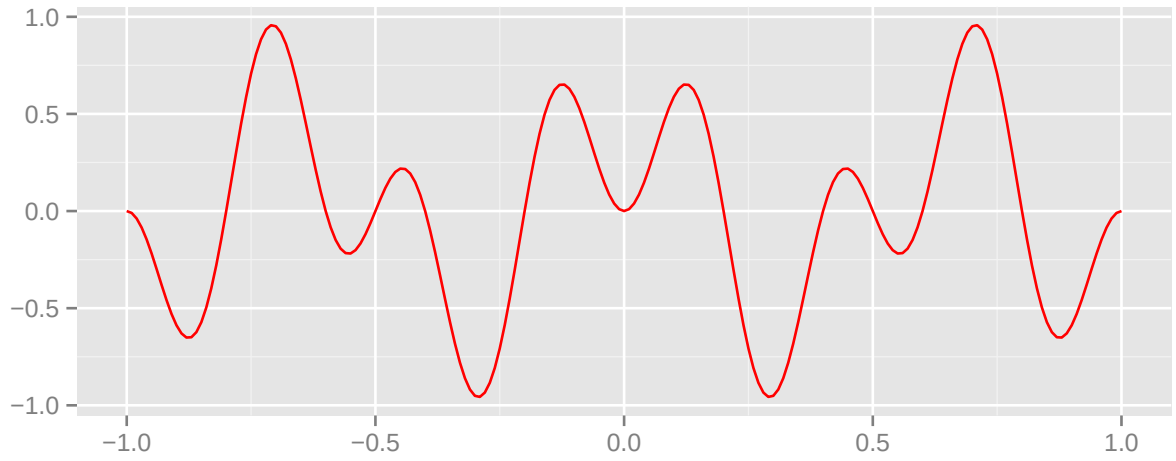
$$x^{(2)} = \sin(f_2 t), \quad f_2 = 5\pi$$

$$\langle x^{(1)}, x^{(2)} \rangle = \int_{-1}^{1} \sin(f_1 t) \sin(f_2 t) dt$$

$x^{(1)} \perp x^{)(2)}$ if $f_1 \neq f_2$ and f_1, f_2 integer multiples of a fundamental (harmonically related)

$$\sin(f_1 t) \sin(f_2 t), \quad f_1 = 2\pi, \; f_2 = 5\pi$$

```
csplot(t, x1*x2)
```



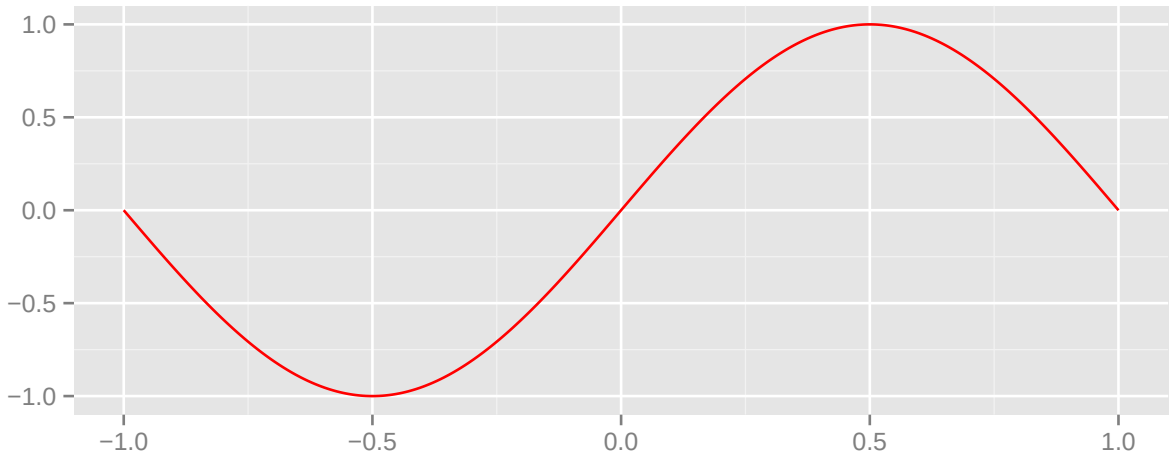

```
round(integrate(function(x) sin(f1*x)*sin(f2*x), lower=-1, upper=1)$value, 10)

## [1] 0
```

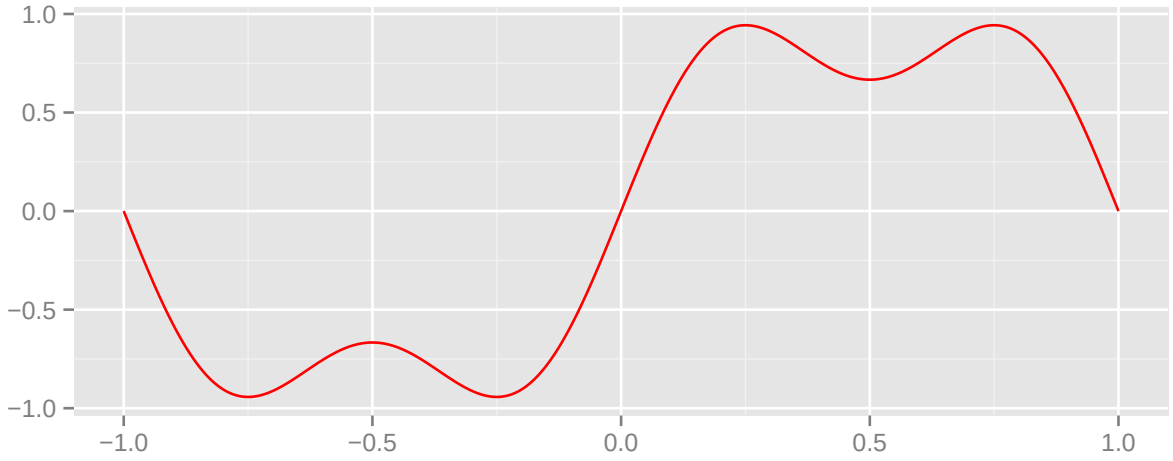
3.1.2 Interesting question when the space has ∞ dimensions

$$\sum_{k=0}^{N} x^{(2k+1)}, \quad x^{(n)} = \sin(\pi n t)/n, t \in [-1, 1]$$

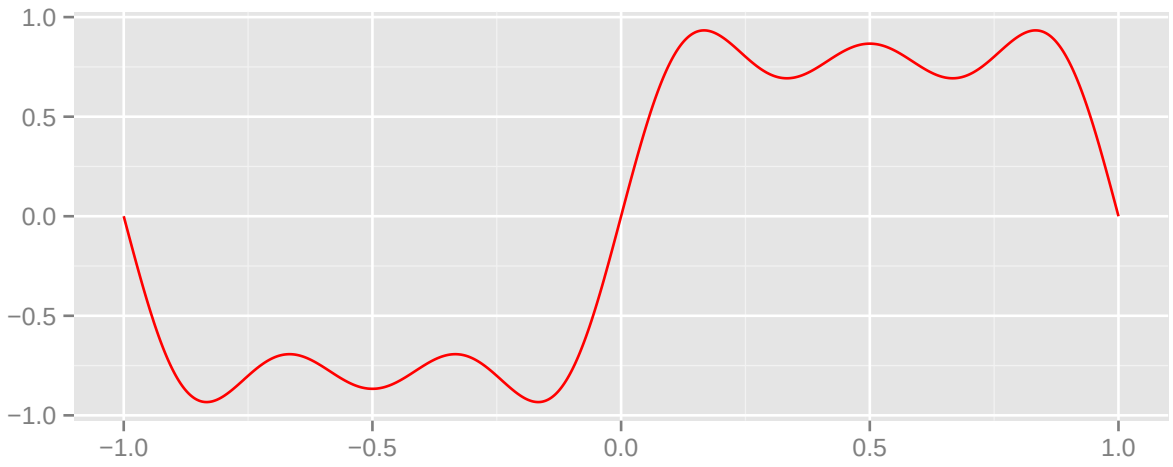
```
inftySpace <- function(N) {
  t <- seq(-1, 1, by=0.001)
  x <- rep(0, length(t))
  for (k in 0:N) {
    n <- 2 * k + 1
    x <- x + sin(pi*n*t)/n
  }
  csplot(t, x)
}
inftySpace(0)
```



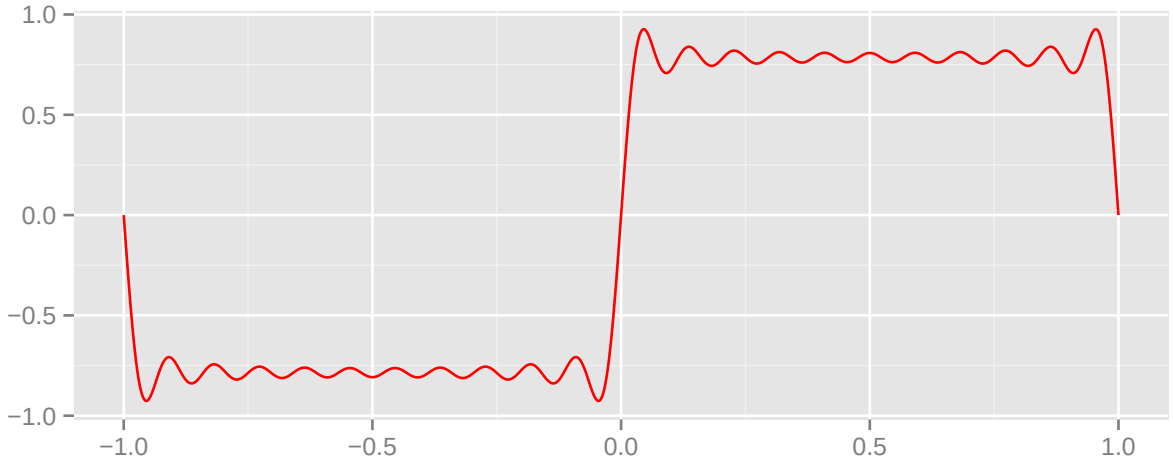

```
inftySpace(1)
```



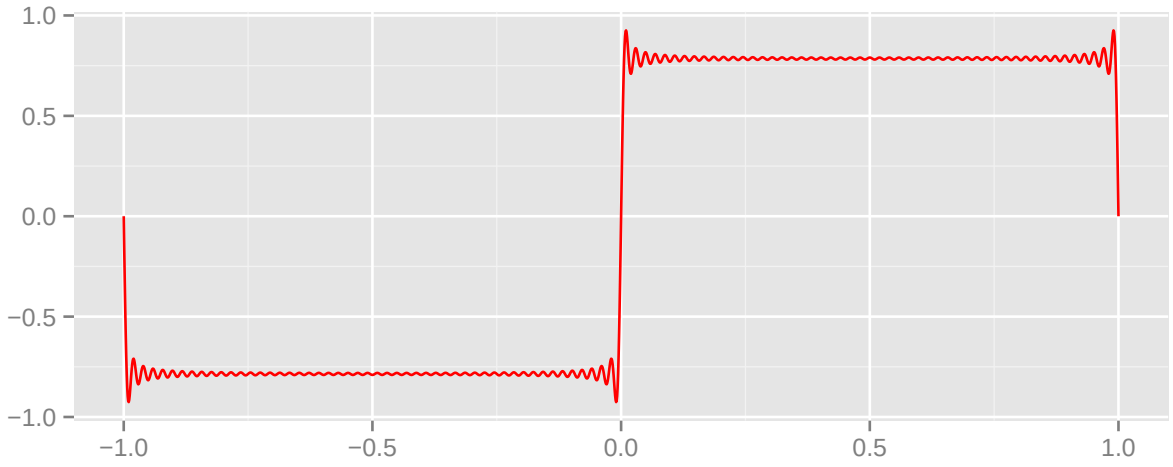

```
inftySpace(2)
```



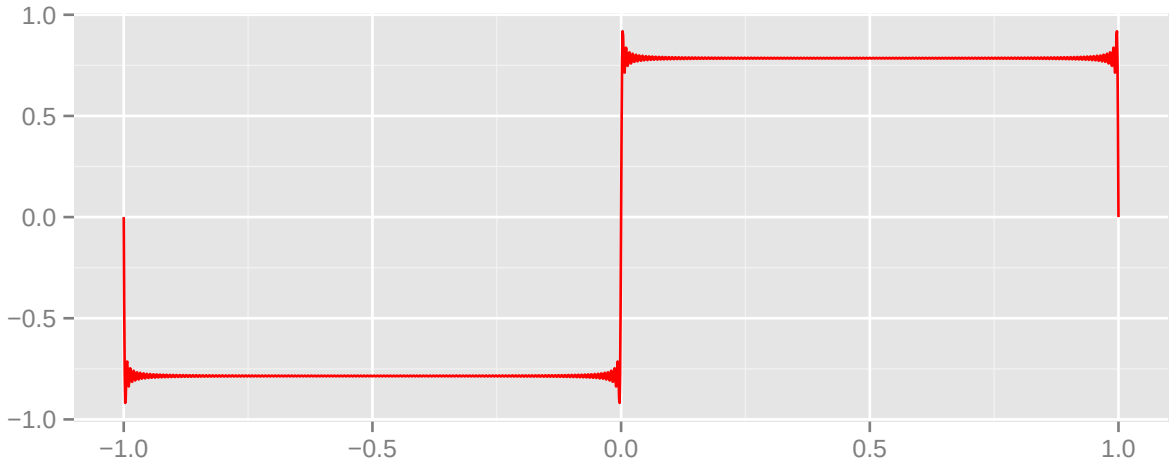

```
inftySpace(10)
```




```
inftySpace(50)
```




```
inftySpace(150)
```



3.2 Hilbert Space, properties and bases

3.2.1 Formal properties of a vector space

For $x, y, z \in V$ and $\alpha, \beta \in \mathbb{C}$:

- $x + y = y + x$
- $(x + y) + z = x + (y + z)$
- $\alpha(x + y) = +\alpha y$
- $(\alpha + \beta)x = \alpha x + \beta x$
- $\alpha(\beta x) = (\alpha\beta)x$
- $\exists 0 \in V \quad | \quad x + 0 = 0 + x = x$
- $\forall x \in V \; \exists(-x) \quad | \quad x + (-x) = 0$

3.2.2 Formal properties of the inner product

For $x, y, z \in V$ and $\alpha \in \mathbb{C}$:

- $\langle x + y, z\rangle = \langle x, z\rangle + \langle y, z\rangle$
- $\langle x, y\rangle = \langle y, x\rangle^*$
- $\langle \alpha x, y\rangle = \alpha^* \langle x, y\rangle$
- $\langle x, \alpha y\rangle = \alpha \langle x, y\rangle$
- $\langle x, x\rangle \geq 0$
- $\langle x, x\rangle = 0 \Leftrightarrow x = 0$
- if $\langle x, y\rangle = 0$ and $x, y \neq 0$ then x and y are called orthogonal

3.2.3 Inner product for signals

$$\langle x, y\rangle = \sum_{n=0}^{N-1} x^*[n] y[n]$$

$$\langle x, y\rangle = \sum_{n=-\infty}^{\infty} x^*[n] y[n]$$

We require sequences to be *square-summable*: $\sum |x[n]|^2 < \infty$.
Space of square-summable sequences: $l_2(\mathbb{Z})$.

3.2.4 Special bases

Orthogonal basis: $\langle w^(k), w^{(n)}\rangle = 0$ for $k \neq n$.
Orthonormal basis: $\langle w^(k), w^{(n)}\rangle = \sigma[n - k]$.

3.3 Hilbert Space and approximation

3.3.1 Parseval's Theorem

$$x = \sum_{k=0}^{K-1} \alpha_k w^{(k)}$$

For an orthonormal basis:

$$||x||^2 = \sum_{k=0}^{K-1} |\alpha_k|^2$$

3.3.2 Gram-Schmidt algorithm

The Gram-Schmidt algorithm leads to an orthonormal basis for $P_N([-1,1])$:

$$u^{(0)} = \sqrt{1/2}$$
$$u^{(1)} = \sqrt{3/2}t$$
$$u^{(2)} = \sqrt{5/8}(3t^2 - 1)$$
$$u^{(3)} = \dots$$

$$\alpha_k = \langle u^{(k)}, x \rangle$$

Module 4

Fourier Analysis

4.1 Exploration via a change of basis

4.1.1 The Fourier Basis for $\mathbb{C}^N$

In vector notation:

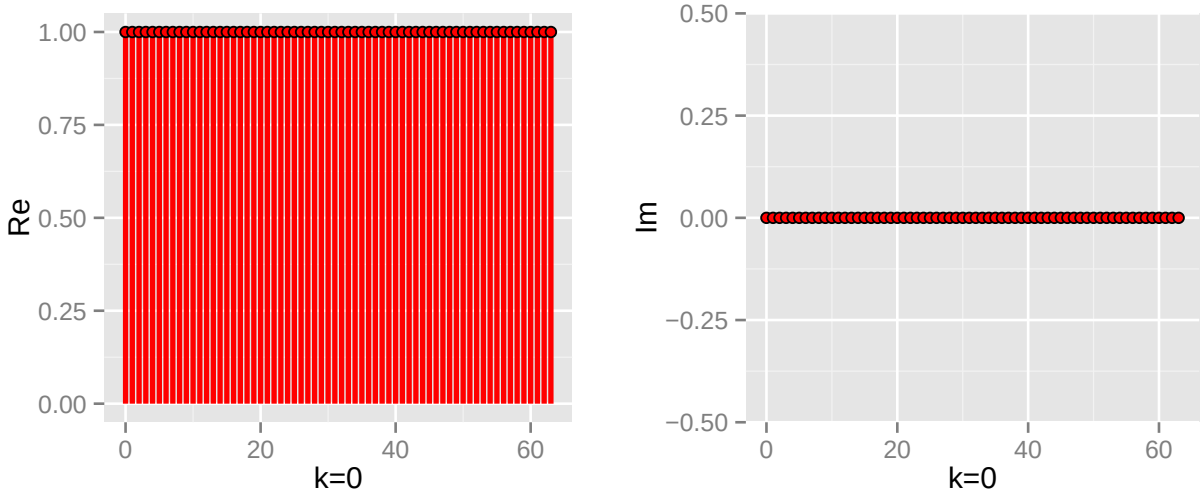
$$\{w^{(k)}_{k=0,1,\ldots,N-1}\}$$

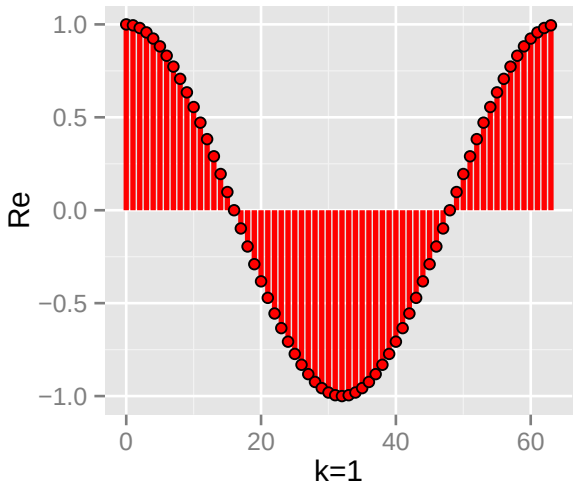
with

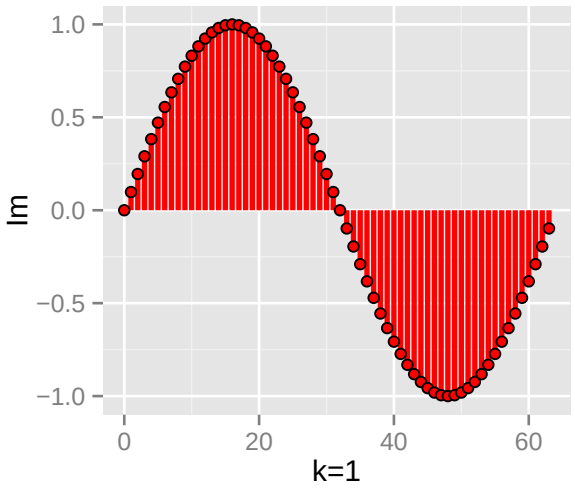
$$w^k_n = e^{j\frac{2\pi}{N}nk}$$

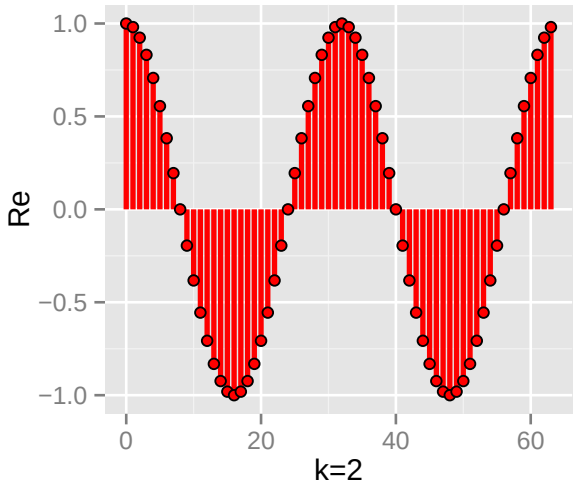
is an orthogonal basis in $\mathbb{C}^N$.

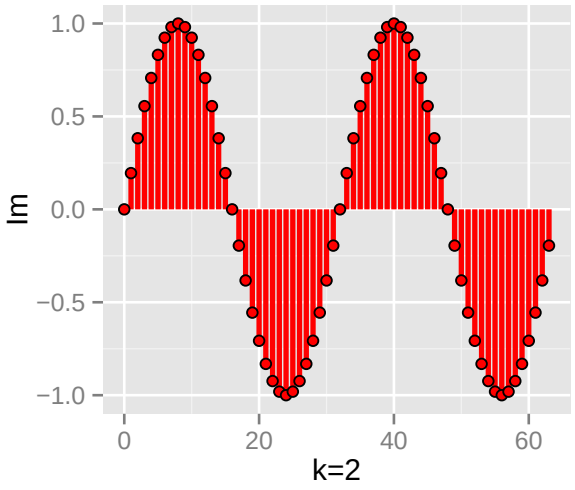
```
N <- 64
plotW <- function(k) {
  n <- 0:(N-1)
  w <- exp(1i * 2*pi/N * n * k)
  grid.arrange(dsplot(n, Re(w), xlab=paste0("k=", k), ylab="Re"),
               dsplot(n, round(Im(w),5), xlab=paste0("k=", k), ylab="Im"), ncol=2)
}
for (k in c(0,1,2,3,4,5,16,20,30,31,32,33,60,62,63))
  plotW(k)
```

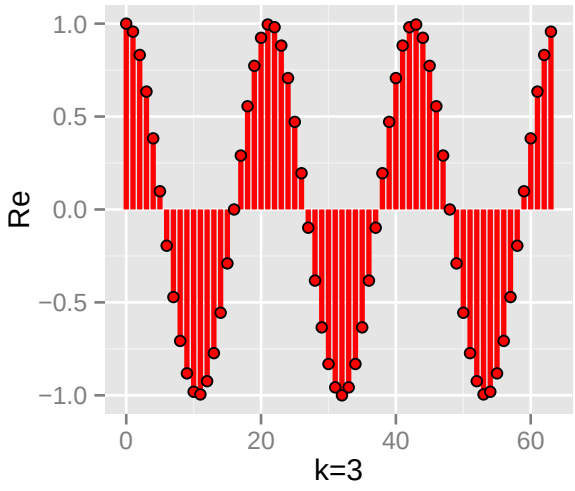


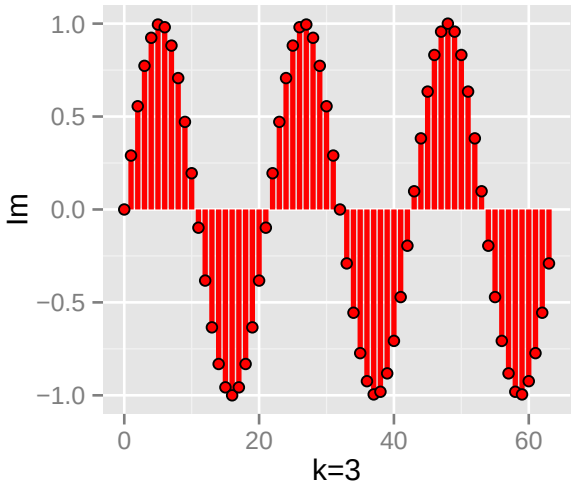

Re
1.0
0.5
0.0
−0.5
−1.0
0
20
40
60
k=1

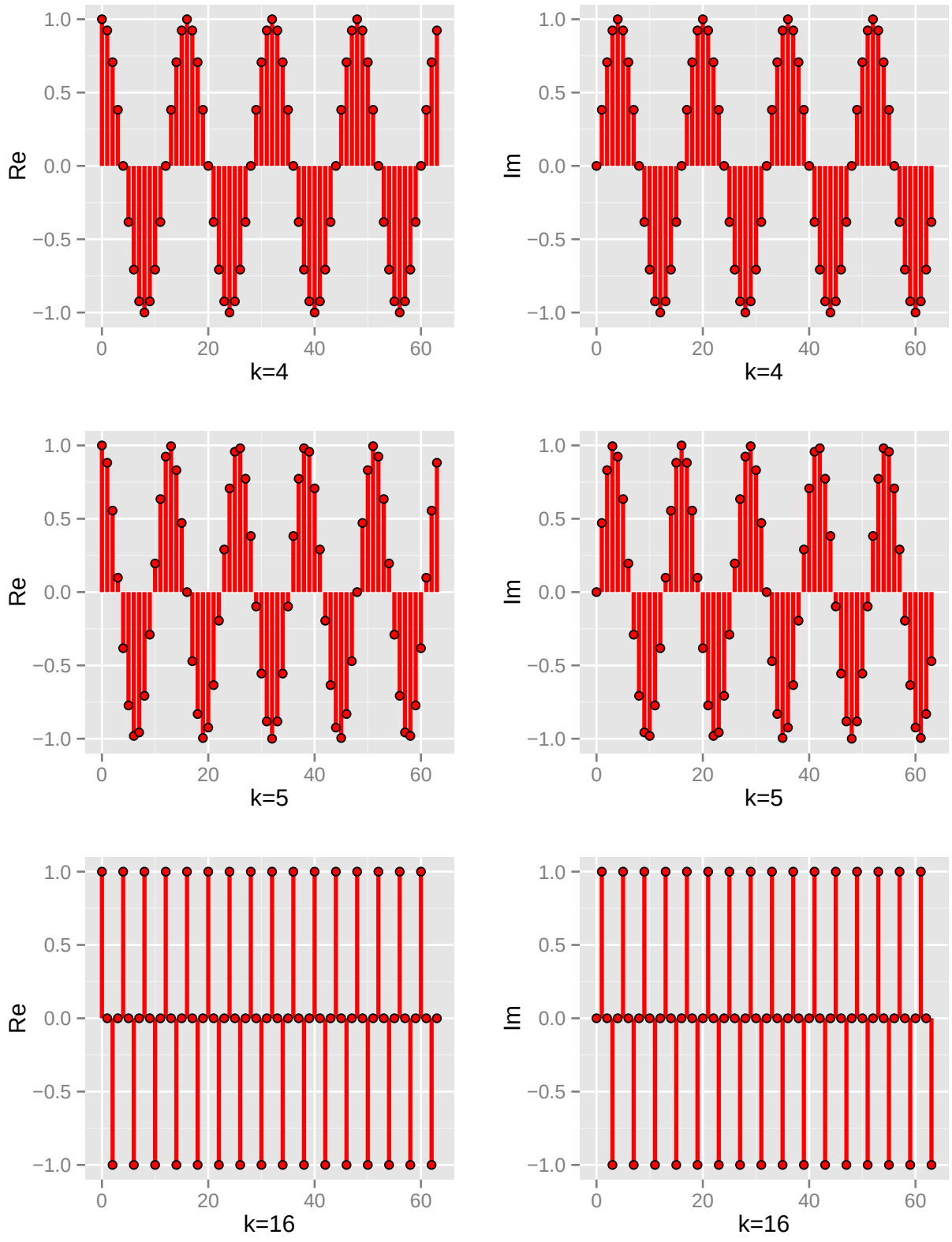

Im
1.0
0.5
0.0
−0.5
−1.0
0
20
40
60
k=1

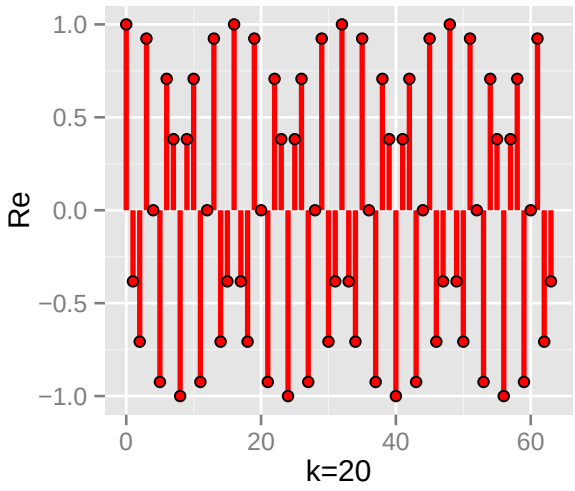

Re
1.0
0.5
0.0
−0.5
−1.0
0
20
40
60
k=2

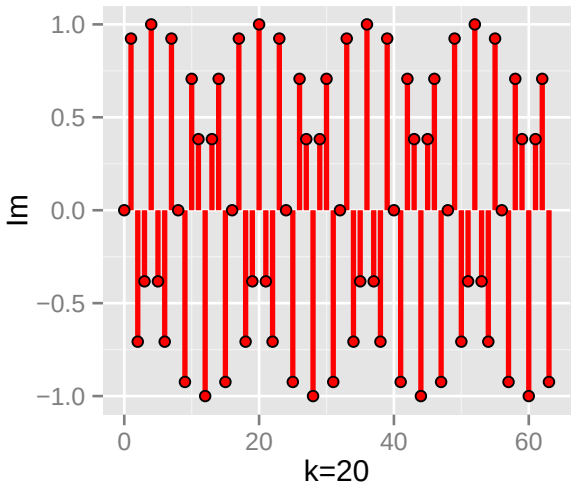

Im
1.0
0.5
0.0
−0.5
−1.0
0
20
40
60
k=2

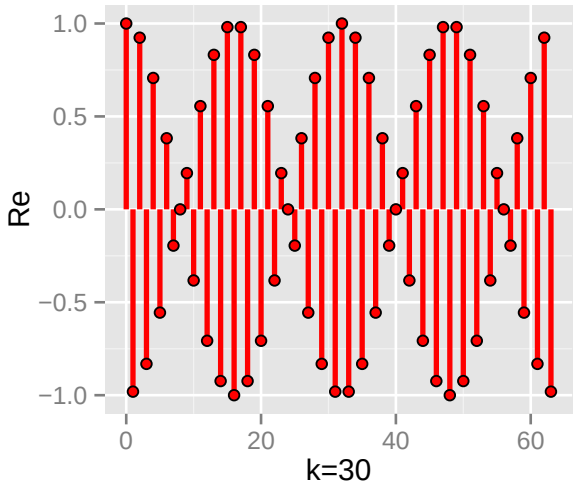

Re
1.0
0.5
0.0
−0.5
−1.0
0
20
40
60
k=3

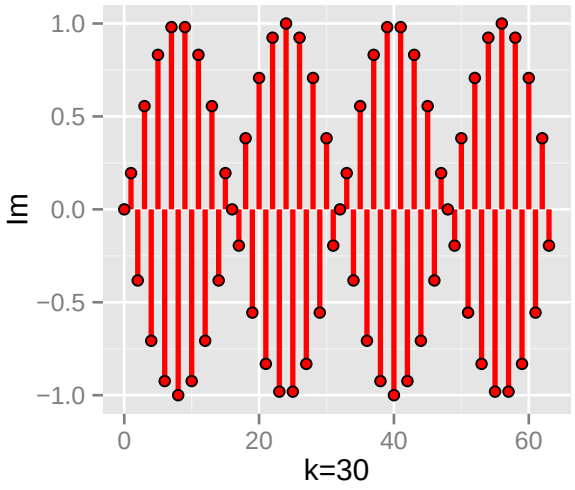

Im
1.0
0.5
0.0
−0.5
−1.0
0
20
40
60
k=3

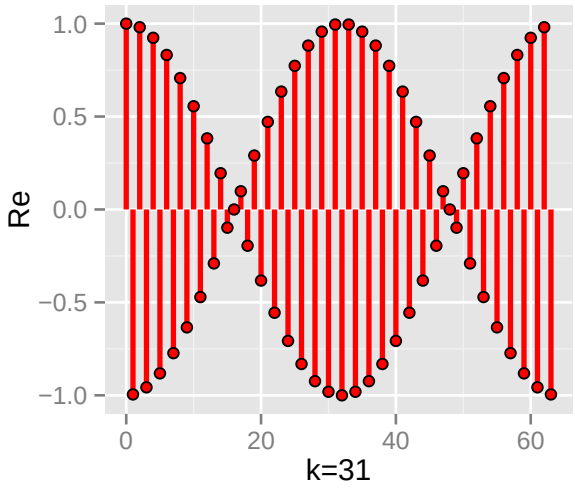

Re
Im
1.0
0.5
0.0
−0.5
−1.0
0
20
40
60
k=4
k=5
k=16

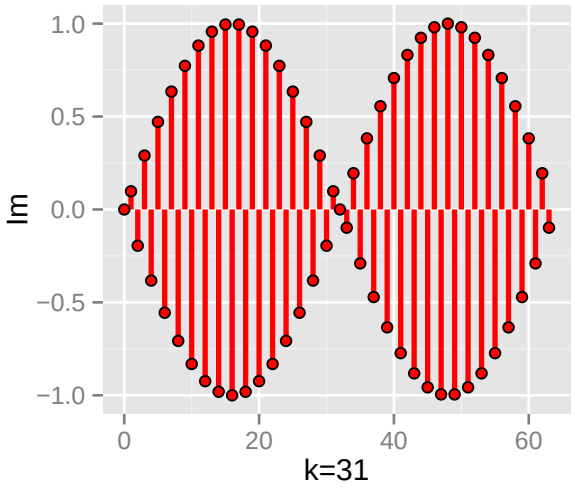

Re
1.0
0.5
0.0
−0.5
−1.0
0
20
40
60
k=20

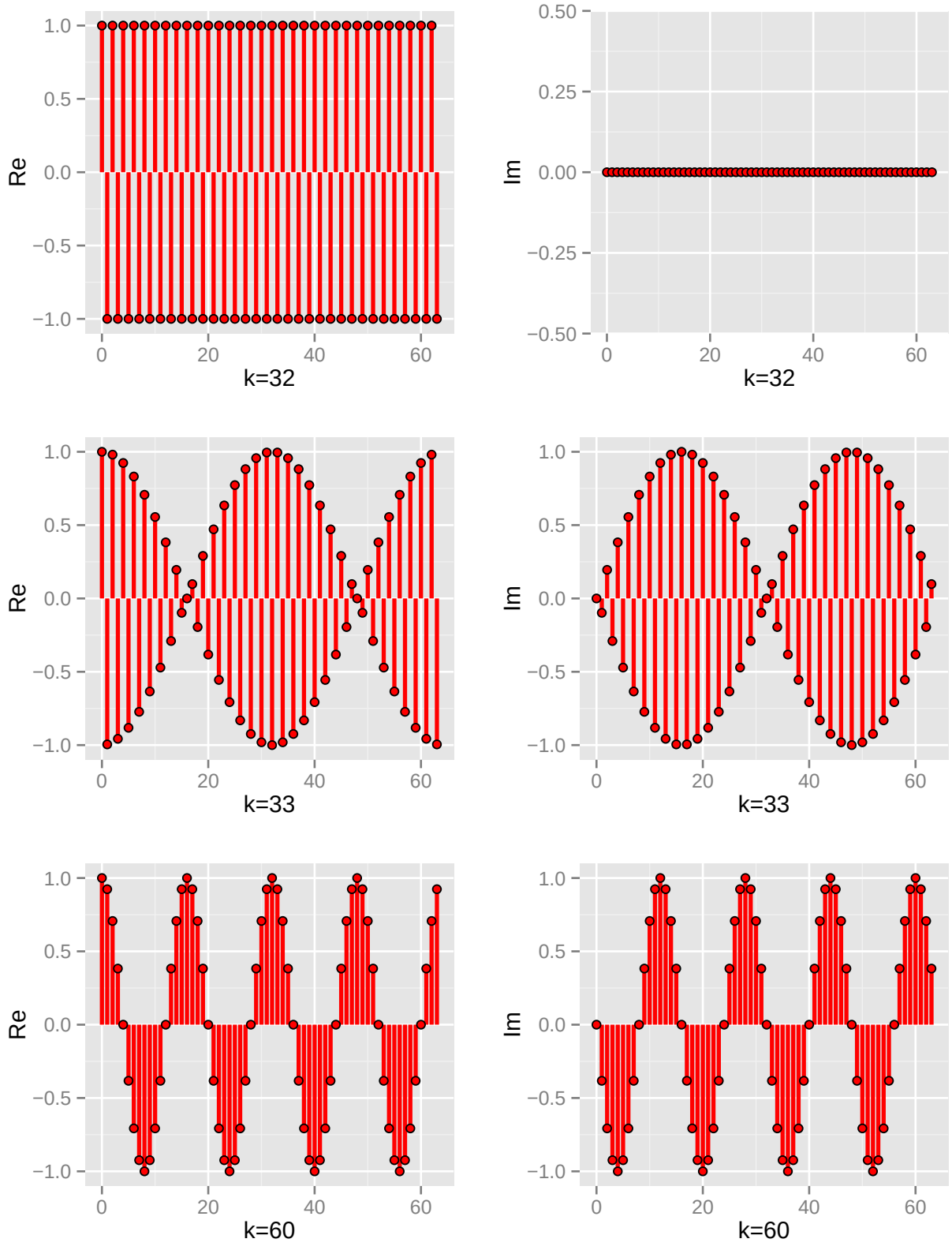

Im
1.0
0.5
0.0
−0.5
−1.0
0
20
40
60
k=20

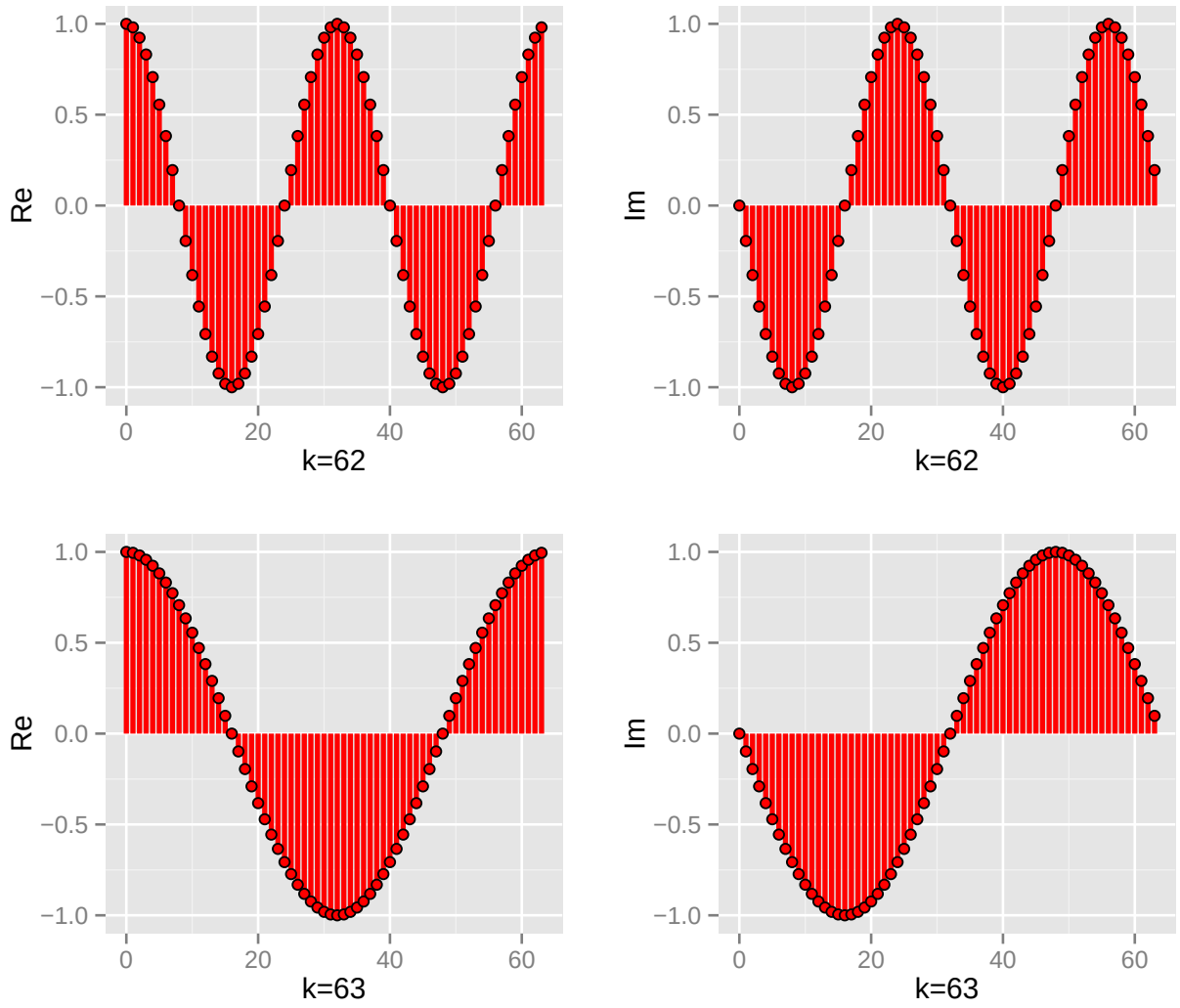

Re
1.0
0.5
0.0
−0.5
−1.0
0
20
40
60
k=30


Im
1.0
0.5
0.0
−0.5
−1.0
0
20
40
60
k=30


Re
1.0
0.5
0.0
−0.5
−1.0
0
20
40
60
k=31


Im
1.0
0.5
0.0
−0.5
−1.0
0
20
40
60
k=31


Re
Im
k=32
k=33
k=60
1.0
0.5
0.0
−0.5
−1.0
0.50
0.25
0.00
−0.25
−0.50
0
20
40
60



4.2 The Discrete Fourier Transform

Analysis formula:

$$X[k] = sum_{n=0}^{N=1} x[n] e^{-j\frac{2\pi}{N}nk}, \quad k = 0, 1, \ldots, N-1$$

N-point signal in the *frequency domain*.
Synthesis formula:

$$x[n] = \frac{1}{N} sum_{k=0}^{N-1} X[k] e^{j\frac{2\pi}{N}nk}, \quad n = 0, 1, \ldots, N-1$$

N-point signal in the *"time" domain*.
DFT is obviously linear:

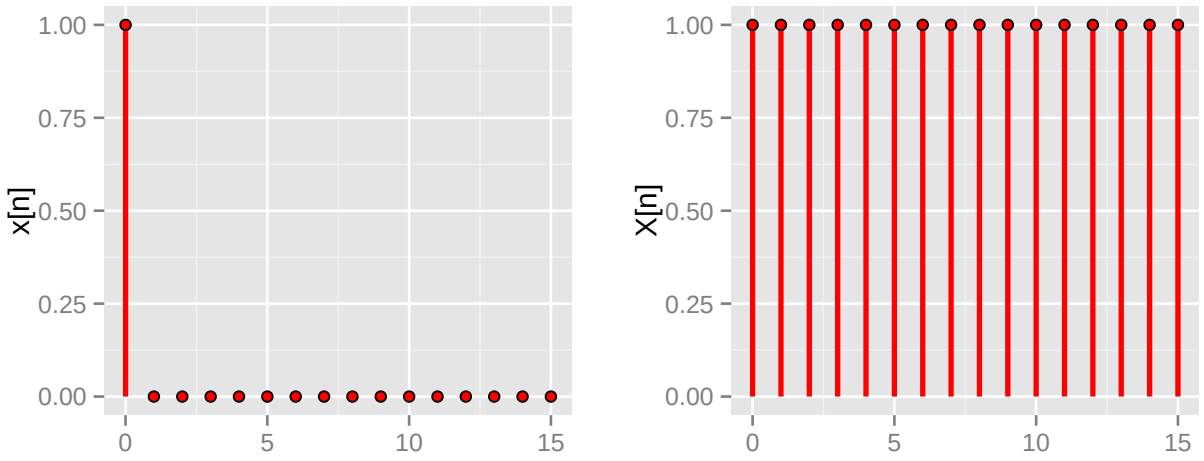
$$DFT\{\alpha x[n] + \beta y[n]\} = \alpha DFT\{x[n]\} + \beta DFT\{y[n]\}$$

4.2.1 DFT of the delta signal

$$DFTof x[n] = \sigma[n], \quad x[n] \in \mathbb{C}^N$$

$$X[k] = 1$$

```
n <- 0:15
x <- c(1, rep(0,15))
X <- fft(x)
grid.arrange(dsplot(n, x, ylab="x[n]"), dsplot(n, Re(X), ylab="X[n]"), ncol=2)
```

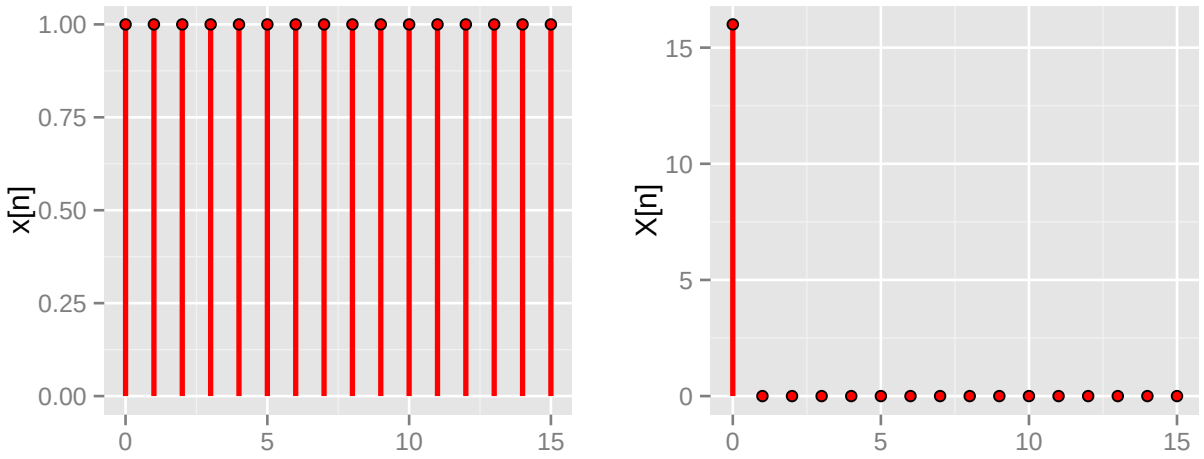


4.2.2 DFT of the uniform signal

$$DFTof x[n] = 1, \quad x[n] \in \mathbb{C}^N$$

$$X[k] = N\sigma[k]$$

```
n <- 0:15
x <- rep(1, 16)
X <- fft(x)
grid.arrange(dsplot(n, x, ylab="x[n]"), dsplot(n, Re(X), ylab="X[n]"), ncol=2)
```

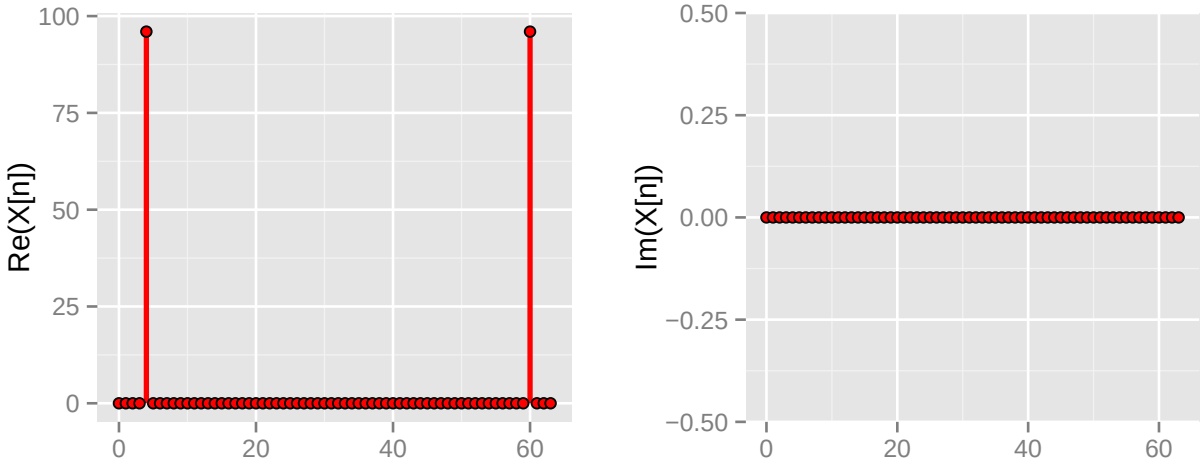


4.2.3 DFT of the cos signal

$$DFTof x[n] = 3\cos(2\pi/16n), \quad x[n] \in \mathbb{C}^6 4$$

$$X[k] = \begin{cases} 96 & \text{for} k = 4, 60 \\ 0 & otherwise \end{cases}$$

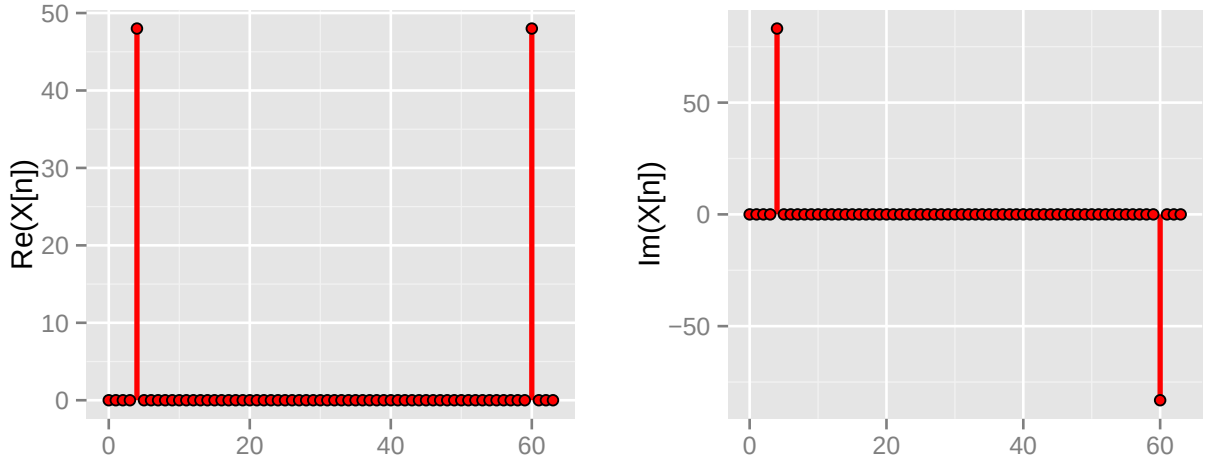
```
n <- 0:63
x <- 3 * cos(2*pi/16*n)
X <- fft(x)
grid.arrange(dsplot(n, Re(X), ylab="Re(X[n])"),
             dsplot(n, round(Im(X),4), ylab="Im(X[n])"), ncol=2)
```



$$DFTof x[n] = 3\cos(2\pi/16n + \pi/3), \quad x[n] \in \mathbb{C}^6 4$$

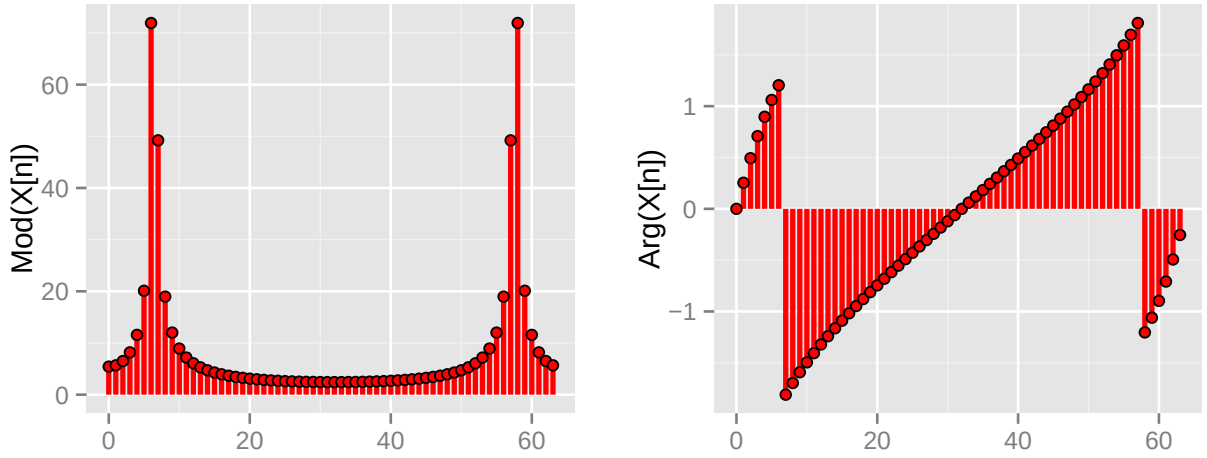
$$X[k] = \begin{cases} 96e^{j\frac{\pi}{3}} & \text{for} k = 4 \\ 96e^{-j\frac{\pi}{3}} & \text{for} k = 60 \\ 0 & otherwise \end{cases}$$

```
n <- 0:63
x <- 3 * cos(2*pi/16*n + pi/3)
X <- fft(x)
grid.arrange(dsplot(n, Re(X), ylab="Re(X[n])"),
             dsplot(n, round(Im(X),4), ylab="Im(X[n])"), ncol=2)
```



$$DFTofx[n] = 3\cos(2\pi/10n), \quad x[n] \in \mathbb{C}^6 4$$

```
n <- 0:63
x <- 3 * cos(2*pi/10*n)
X <- fft(x)
grid.arrange(dsplot(n, Mod(X), ylab="Mod(X[n])"),
             dsplot(n, Arg(X), ylab="Arg(X[n])"), ncol=2)
```

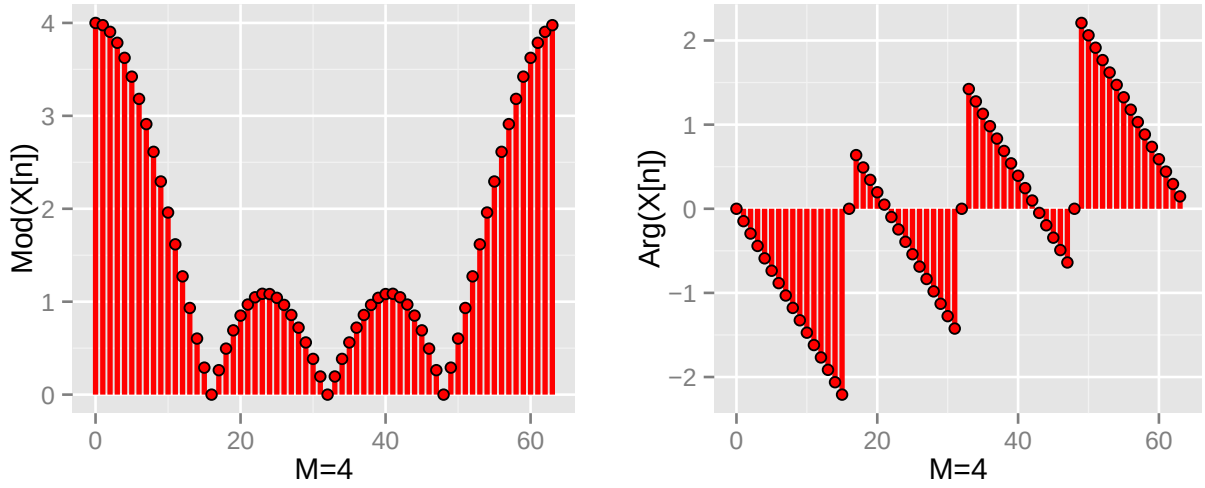


4.2.4 DFT of length-M step in $\mathbb{C}^N$

$$x[n] = \sum_{h=0}^{M-1} \sigma[n-h], \quad n = 0, 1, \ldots, N-1$$

$$X[k] = \frac{\sin(\frac{\pi}{N}Mk)}{\sin(\frac{\pi}{N}k)} e^{-j\frac{pi}{N}(M-1)k}$$

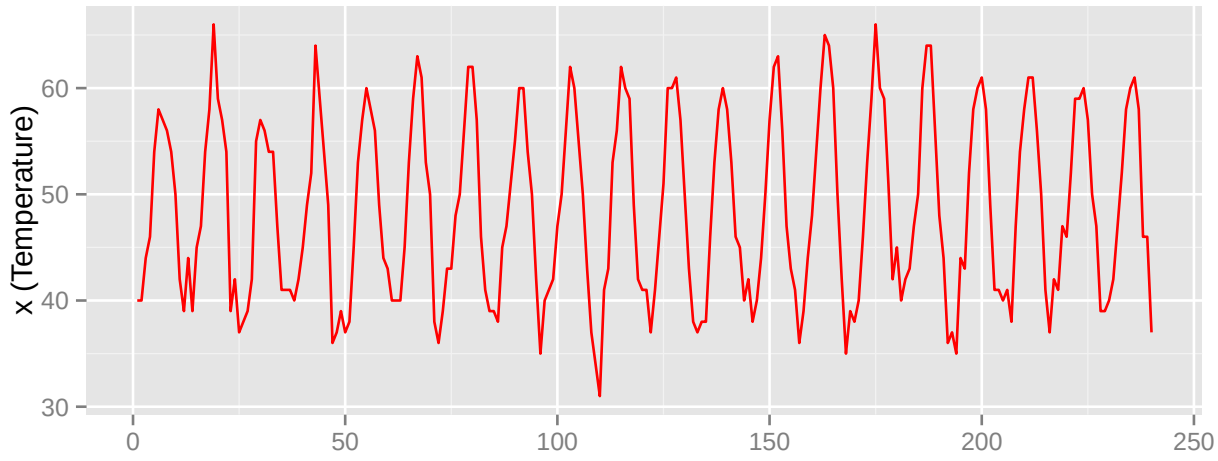
```
plotStep <- function(M) {
  n <- 0:63
  x <- c(rep(1, M), rep(0, 64-M))
  X <- fft(x)
  grid.arrange(dsplot(n, Mod(X), ylab="Mod(X[n])", xlab="M=4"),
               dsplot(n, Arg(X), ylab="Arg(X[n])", xlab="M=4"), ncol=2)
}
plotStep(4)
```



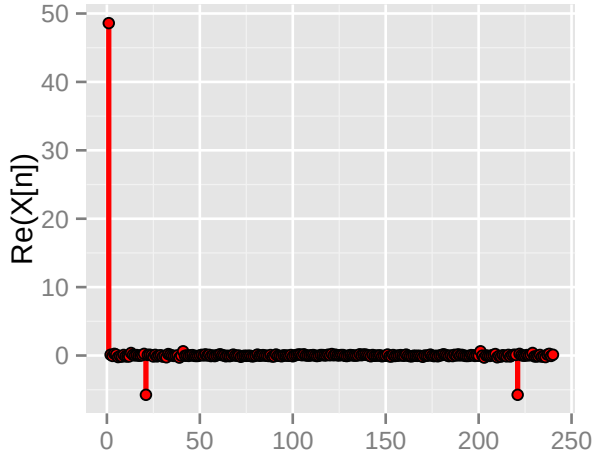
4.3 The DFT in practice

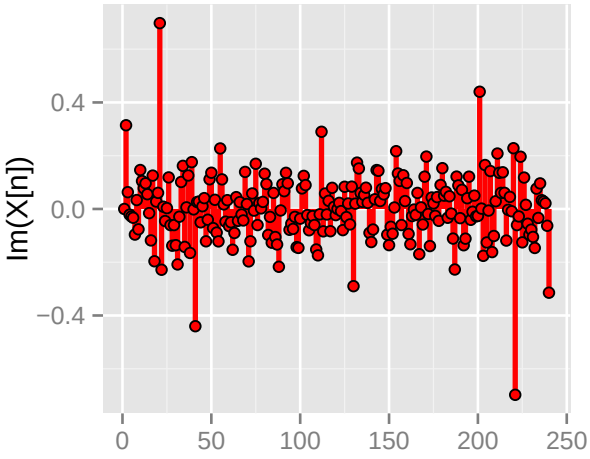
4.3.1 Average Monthly Temperatures at Nottingham, 1920-1939

```
x <- as.integer(nottem)
n <- 1:length(x)
X <- fft(x) / length(x)
csplot(n, x, ylab="x (Temperature)")
```




```
grid.arrange(dsplot(n, Re(X), ylab="Re(X[n])"),
             dsplot(n, Im(X), ylab="Im(X[n])"), ncol=2)
```






```
MainPeak = which.max(abs(Re(X[-1])))
cat("Average value:", Re(X[1]))

## Average value: 48.59583

cat("DFT main peak for k = ", MainPeak, ", value = ", abs(Re(X[MainPeak+1])))

## DFT main peak for k =  20 , value =  5.742034

cat("Month count:", length(n))

## Month count: 240

cat("Frequency: ", length(n) , "/", MainPeak, "=", length(n)/MainPeak, "months")

## Frequency:  240 / 20 = 12 months

cat("Temperature excursion: ", Re(X[1]), "+-", 2 * abs(Re(X[MainPeak+1])))

## Temperature excursion:  48.59583 +- 11.48407
```

4.4 DFT, DFS, DTFT

4.4.1 Discrete-Time Fourier Transform (DTFT)

Formal definition:

- $x[n] \in l_2(\mathbb{Z})$
- define the *function* of $\omega \in \mathbb{R}$

$$F(\omega) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n}$$

- inversion (when $F(\omega)$ exists):

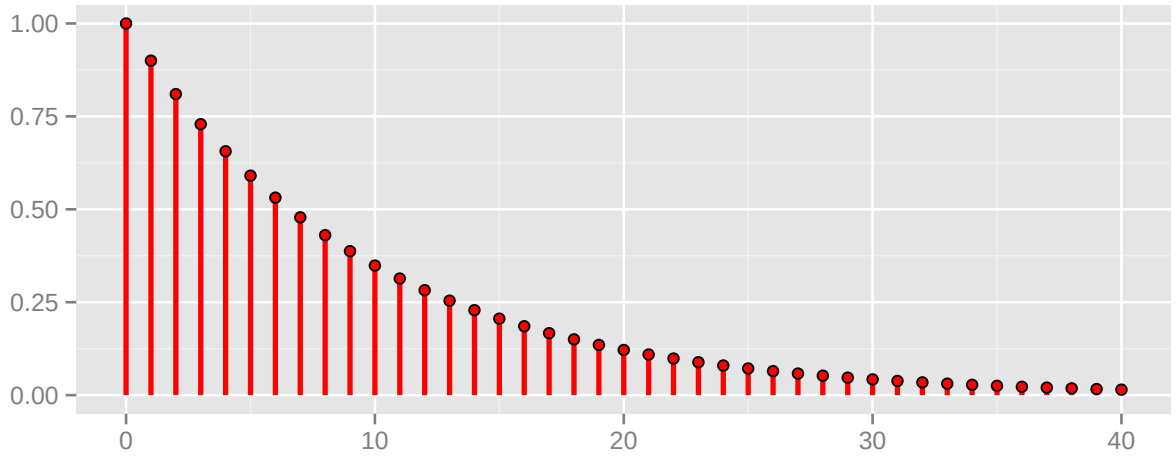
$$x[n] = \frac{1}{2\pi} \int_{-pi}^{pi} F(\omega)e^{j\omega n} d\omega$$

- F(ω)$is2\pi$-periodic
- to stress periodicity (and for other reasons) we will write

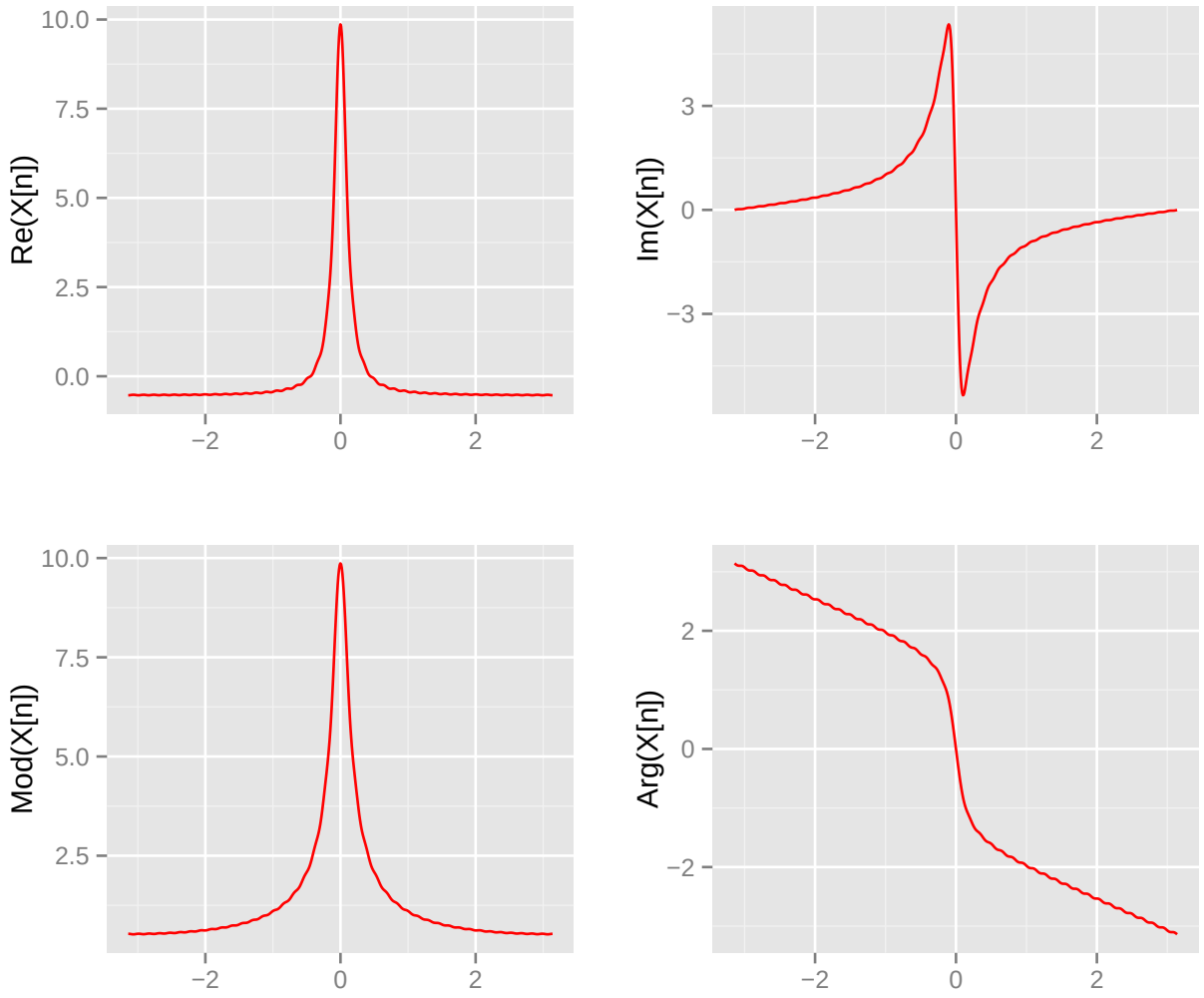
$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n}$$

- by convention, $X(e^{j\omega})$ is represented over $[-pi; pi]$

```
alpha <- 0.9; n <- 0:40
x <- alpha^n
dsplot(n, x)
```




```
dtftplot(x)
```


Re(X[n])
10.0
7.5
5.0
2.5
0.0
−2
0
2
Im(X[n])
3
0
−3
−2
0
2
Mod(X[n])
10.0
7.5
5.0
2.5
−2
0
2
Arg(X[n])
2
0
−2
−2
0
2

4.5 DTFT: intuition and properties

4.5.1 DTFT properties

- Linearity
$$DTFT\{\alpha x[n] + \beta y[n]\} = \alpha X(e^{j\omega}) + \beta Y(e^{j\omega})$$
- Time shift
$$DTFT\{x[n-M]\} = e^{-j\omega M} X(e^{j\omega})$$
- Modulation (dual)
$$DTFT\{e^{j\omega_0 n} x[n]\} = X(e^{j(\omega-\omega_0)})$$
- Time reversal
$$DTFT\{x[-n]\} = X(e^{-j\omega})$$
- Conjugation
$$DTFT\{x^*[n]\} = X^*(e^{-j\omega})$$

4.5.2 Some particular cases

- if $x[n]$ is symmetric, the DTFT is symmetric:
$$x[n] = x[-n] \Leftrightarrow X(e^{j\omega}) = X(e^{-j\omega})$$
- if $x[n]$ is real, the DTFT is Hermitian-symmetric:
$$x[n] = x^*[n] \Leftrightarrow X(e^{j\omega}) = X^*(e^{-j\omega})$$
- special case: if $x[n]$ is real, the magnitude of the DTFT is symmetric:
$$x[n] \in \mathbb{R} \Rightarrow |X(e^{j\omega})| = |X(e^{-j\omega})|$$
- more special case: if $x[n]$ is real and symmetric, $X(e^{j\omega})$ is also real and symmetric

4.5.3 DTFT as basic expansion

- $DFT\{\delta[n]\} = 1$
- $DTFT\{\delta[n]\} = 1$
- $DFT\{1\} = N\delta[k]$
- $DTFT\{1\} = \tilde{\delta}(\omega)$
- $DTFT\{e^{j\omega_0 n}\} = \tilde{\delta}(\omega - \omega_0)$
- $DTFT\{\cos(\omega_0 n)\} = \Big(\tilde{\delta}(\omega-\omega_0) + \tilde{\delta}(\omega+\omega_0)\Big)/2$
- $DTFT\{\sin(\omega_0 n)\} = -j\Big(\tilde{\delta}(\omega-\omega_0) - \tilde{\delta}(\omega+\omega_0)\Big)/2$

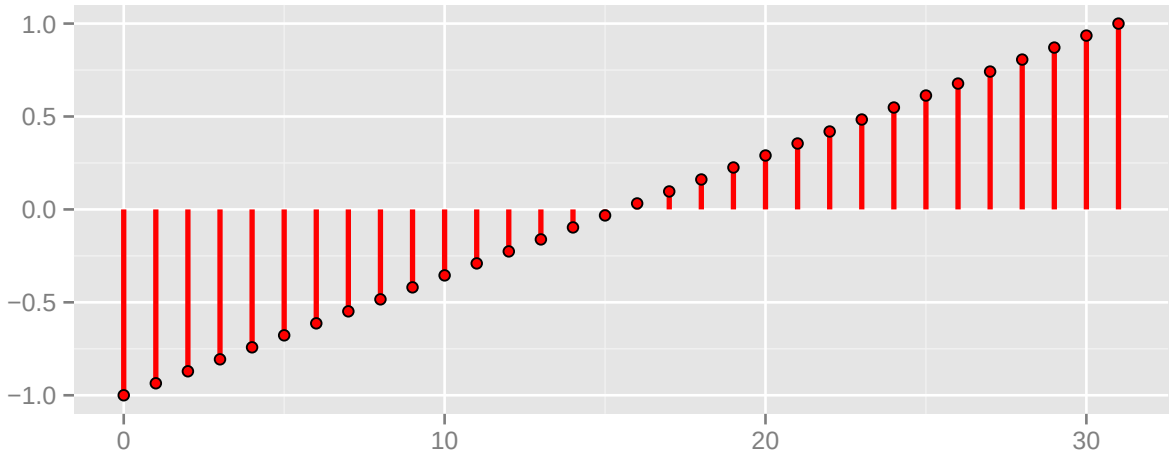
4.6 Relationship between transforms

4.6.1 DTFT of periodic signals

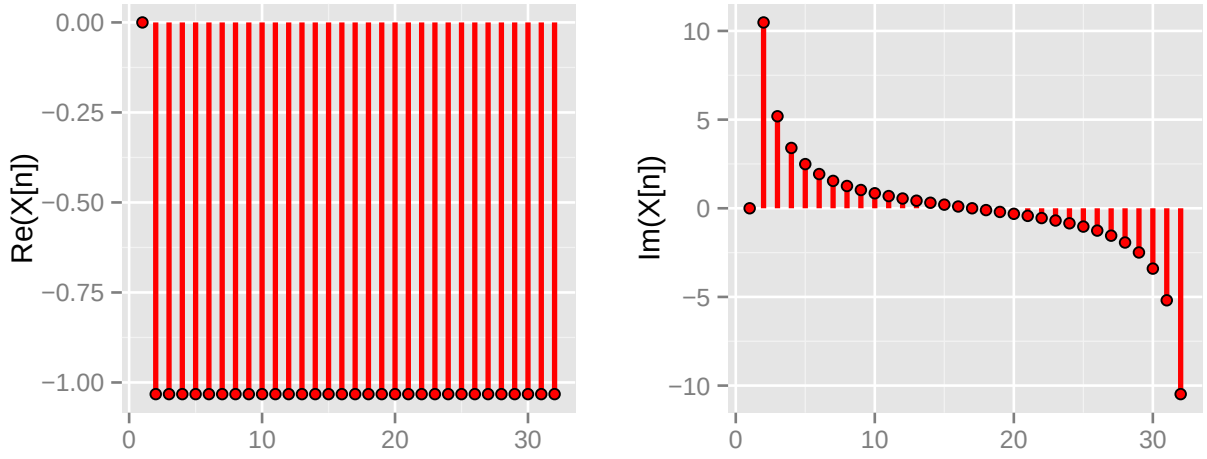
$$\tilde{X}(e^{j\omega}) = \frac{1}{N}\sum_{k=0}^{N-1} X[k]\tilde{\delta}(\omega - \frac{2\pi}{N}k)$$

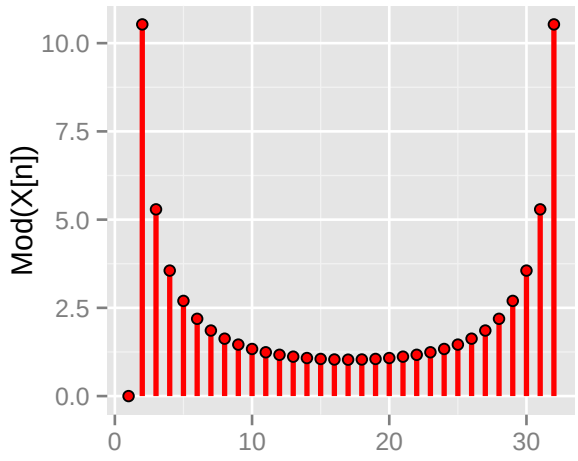
4.6.2 32-tap sawtooth

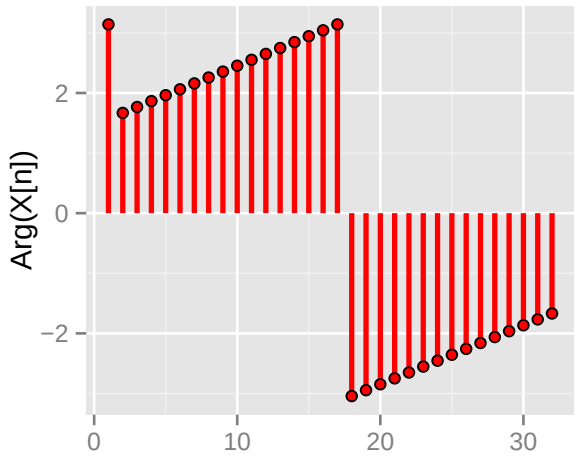
```
n <- 0:31
x <- -1 + 2*n/31
dsplot(n, x)
```



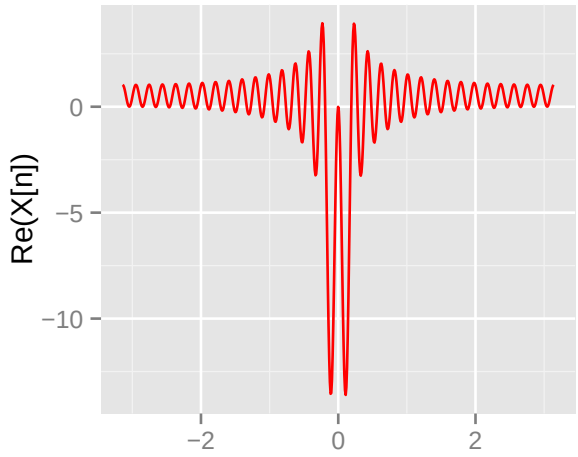

```
dftplot(x)
```

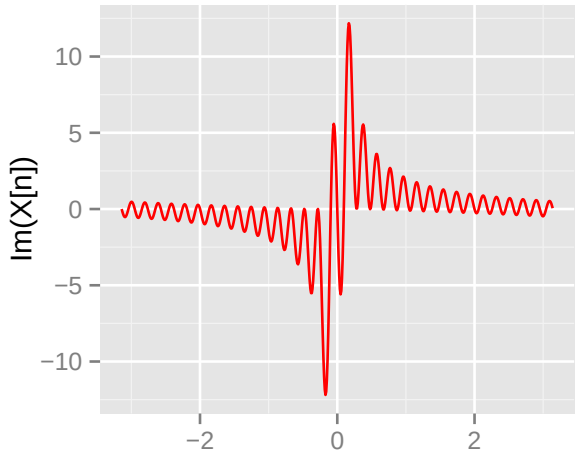


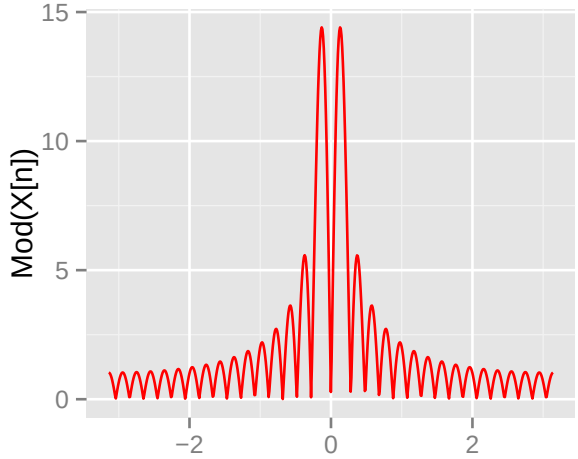

Mod(X[n])
10.0
7.5
5.0
2.5
0.0
0
10
20
30

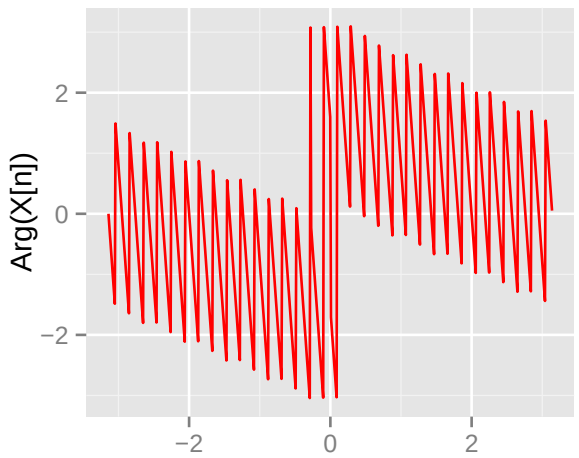

Arg(X[n])
2
0
−2
0
10
20
30

```
dtftplot(x)
```


Re(X[n])
0
−5
−10
−2
0
2

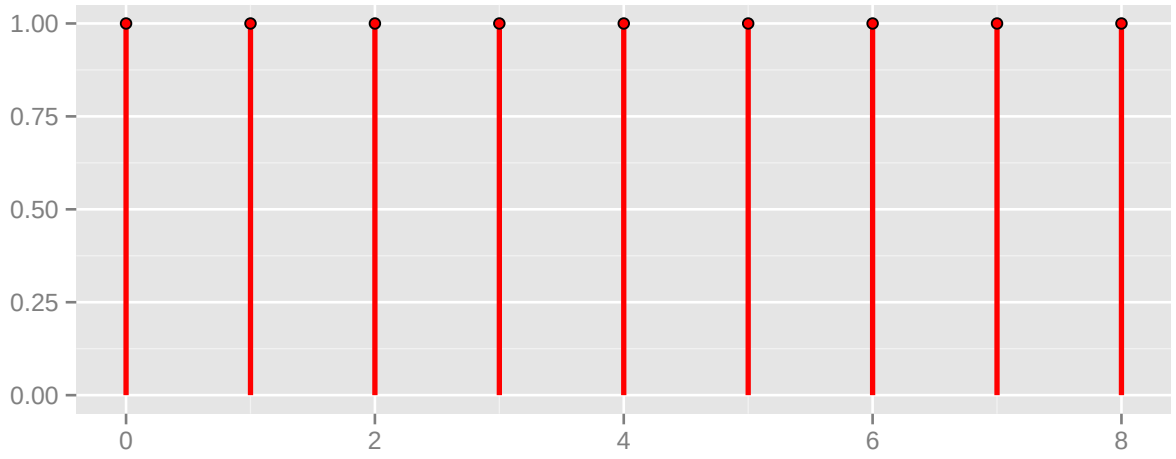

Im(X[n])
10
5
0
−5
−10
−2
0
2


Mod(X[n])
15
10
5
0
−2
0
2

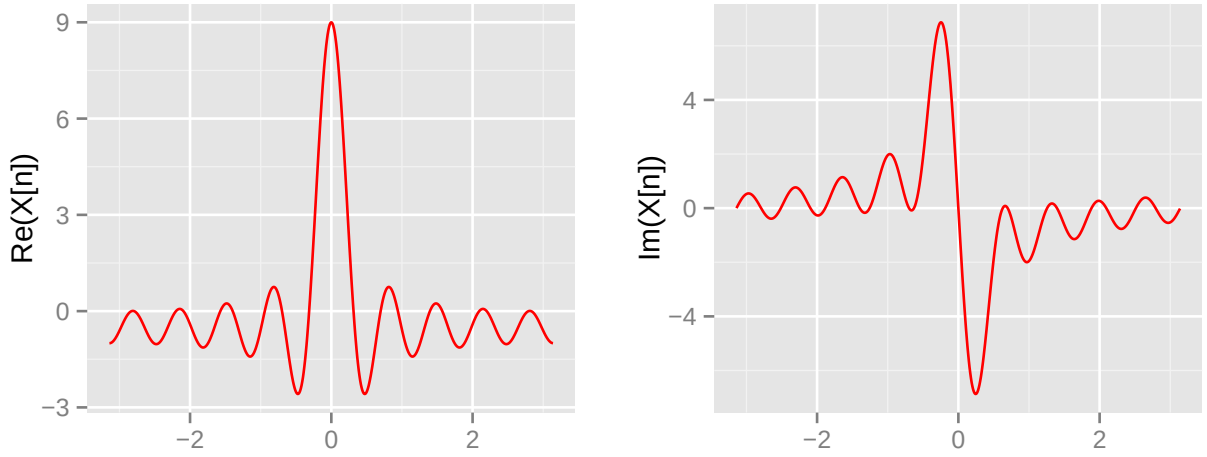

Arg(X[n])
2
0
−2
−2
0
2

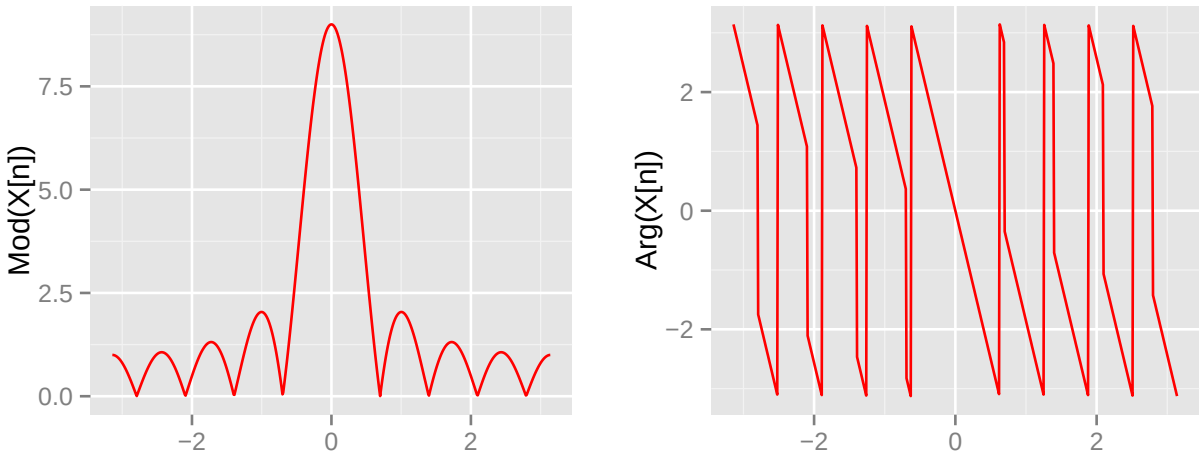
4.6.3 Interval indicator signal

```
N <- 9
n <- 0:(N-1)
x <- rep(1, N)
dsplot(n, x)
```




```
dtftplot(x)
```





4.7 Sinusoidal modulation

4.7.1 Classifying signals in frequency

Three broad categories according to where most of the spectral energy resides:

- lowpass signals (also known as "baseband" signals)
- highpass signals
- bandpass signals

4.7.2 Sinusoidal modulation

$$DTFT\{x[n]\cos(\omega_c n)\} = DTFT\left\{\frac{1}{2}e^{j\omega_c n}x[n] + \frac{1}{2}e^{-j\omega_c n}x[n]\right\} = \frac{1}{2}\left(X(e^{j(\omega-\omega_c)}) + X(e^{j(\omega+\omega_c)})\right)$$

- usually $x[n]$ baseband
- ω_c is the *carrier* frequency

Applications:

- voice and music are lowpass signals
- radio channels are bandpass, in much higher frequencies
- modulation brings the baseband signal in the transmission band
- demodulation at the receiver brings it back

Demodulation in the frequency domain:

- we recovered the baseband signal exactly...
- but we have some spurious high-frequency components
- in the next Module we will learn how to get rid of them!

4.7.3 Tuning a guitar

Problem (abstraction):

- reference sinusoid at frequency ω_0
- tunable sinusoid of frequency ω
- make $\omega = \omega_0$ "by ear"

The procedure:

1. bring ω close to ω_0 (easy)
2. when $\omega \approx \omega_0$ play both sinusoid together
3. trigonometry comes to the rescue:

$$x[n] = cos(\omega_0 n) + cos(\omega n) = 2\cos(\frac{\omega_0 + \omega}{2} n)\cos(\frac{\omega_0 - \omega}{2} n) \approx 2\cos(\Delta_\omega n)\cos(\omega_0 n)$$

Let's see what's happening:

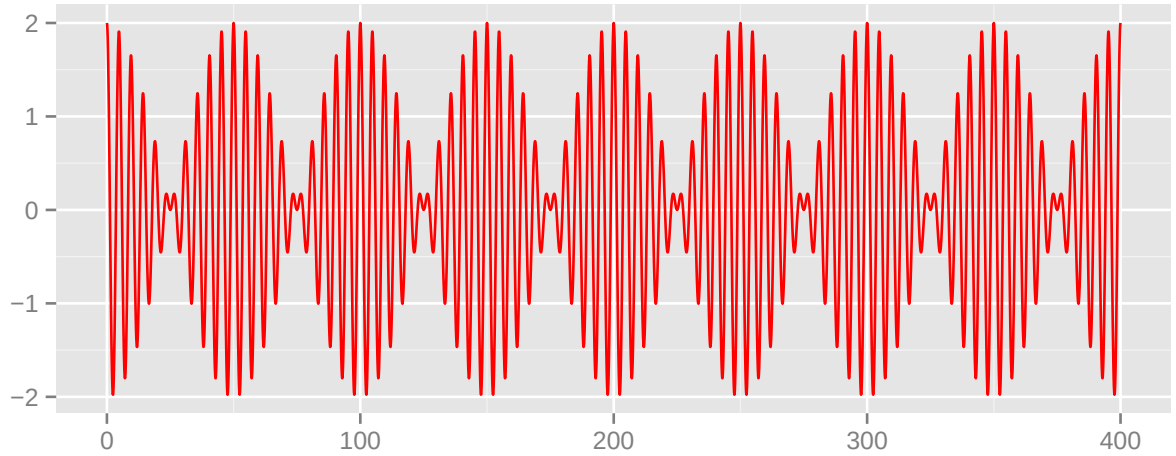
$$x[n] \approx 2\cos(\Delta_\omega n)\cos(\omega_0 n)$$

- "error" signal: $2\cos(\Delta_\omega n)$
- modulation at ω_0: $\cos(\omega_0 n)$
- when $\omega \approx \omega_0$, the error signal is too low to be heard; modulation brings it up to hearing range and we perceive it as amplitude oscillations of the carrier frequency

Examples in the time domain :

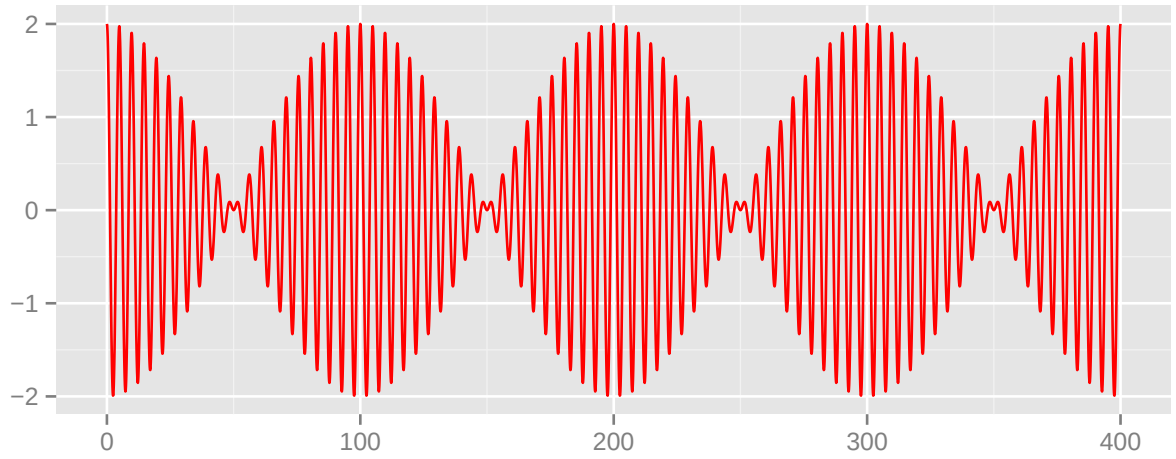
$$\omega_0 = 2\pi \cdot 0.2, \omega = 2\pi \cdot 0.22 \Delta_\omega = 2\pi \cdot 0.0100$$

```
omega0 <- 2*pi*0.2
omega <- 2*pi*0.22
t <- seq(0, 400, by=0.1)
x <- cos(omega0*t) + cos(omega*t)
csplot(t, x)
```



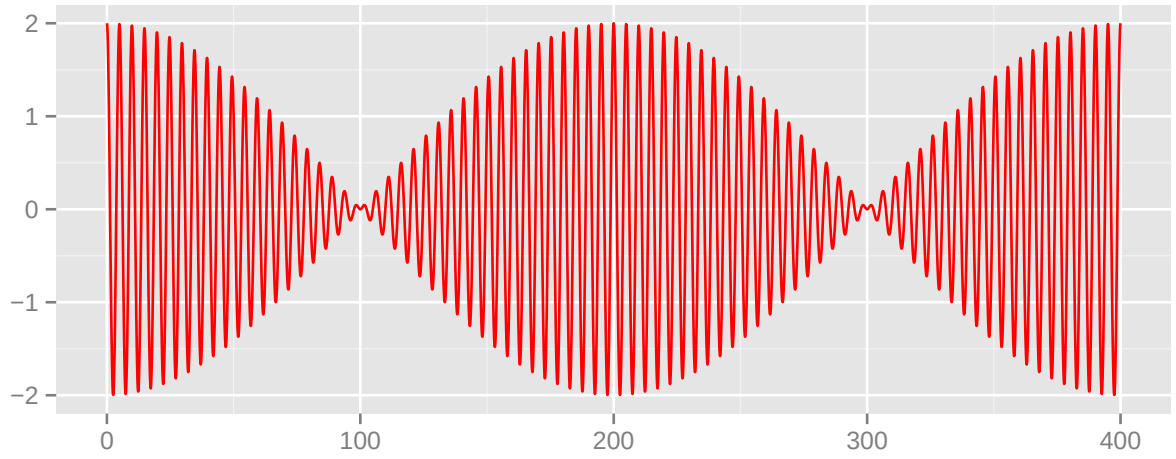
$$\omega_0 = 2\pi \cdot 0.2, \omega = 2\pi \cdot 0.21 \Delta_\omega = 2\pi \cdot 0.0050$$

```
omega0 <- 2*pi*0.2
omega <- 2*pi*0.21
t <- seq(0, 400, by=0.1)
x <- cos(omega0*t) + cos(omega*t)
csplot(t, x)
```



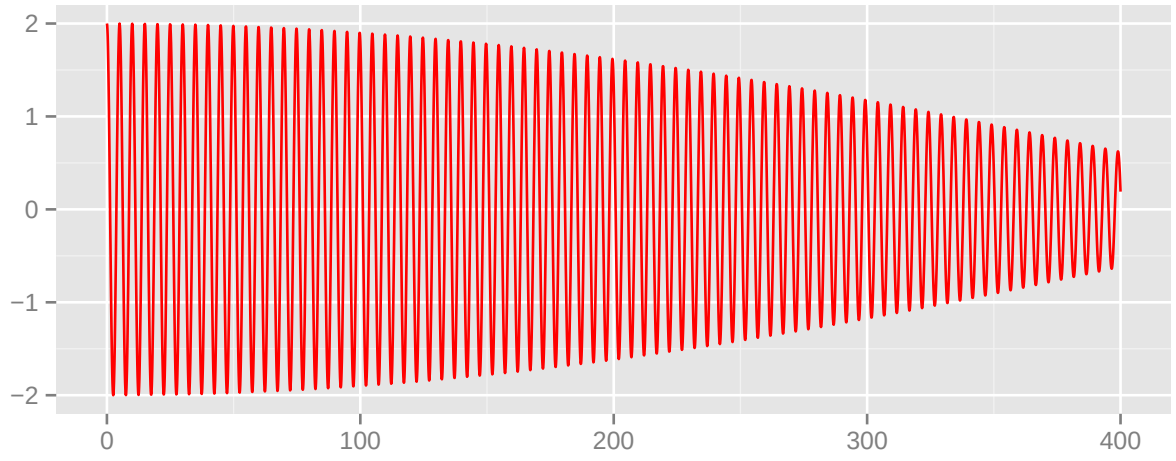
$$\omega_0 = 2\pi \cdot 0.2, \omega = 2\pi \cdot 0.205 \Delta_\omega = 2\pi \cdot 0.0025$$

```
omega0 <- 2*pi*0.2
omega <- 2*pi*0.205
t <- seq(0, 400, by=0.1)
x <- cos(omega0*t) + cos(omega*t)
csplot(t, x)
```



$$\omega_0 = 2\pi \cdot 0.2, \omega = 2\pi \cdot 0.201 \Delta_\omega = 2\pi \cdot 0.0005$$

```
omega0 <- 2*pi*0.2
omega <- 2*pi*0.201
t <- seq(0, 400, by=0.1)
x <- cos(omega0*t) + cos(omega*t)
csplot(t, x)
```



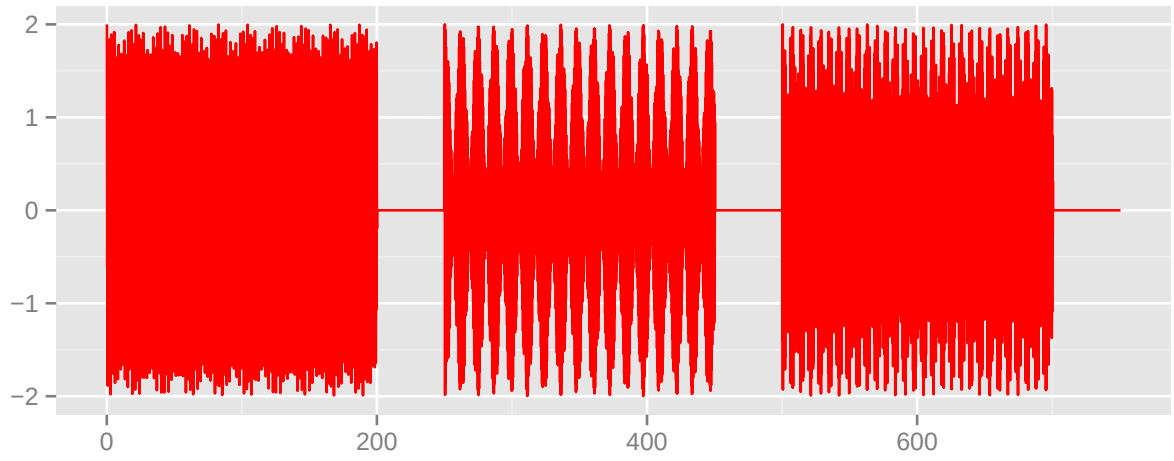
4.8 The Short-Time Fourier Transform

Dual-Tone Multi Frequency (DTMF) dialing:

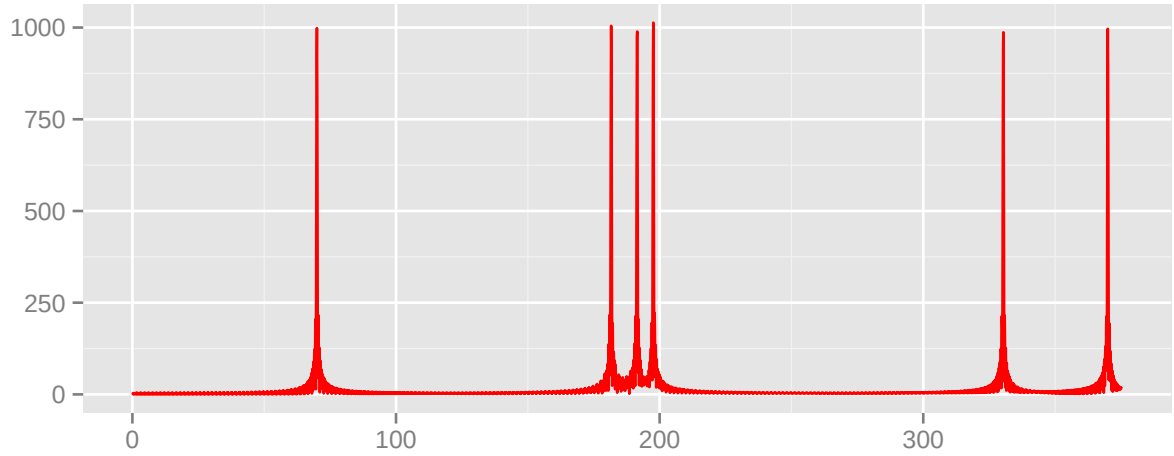
```
f123 <- 697
f456 <- 770
f789 <- 852
fs0s <- 941
f147s <- 1209
f2580 <- 1336
f369s <- 1477
```

4.8.1 Example: 1-5-9

```
step <- 0.1
t1 <- seq(0, 200, by=step)
t2 <- seq(0, 50, by=step)
x <- c(
  cos(t1 * f123) + cos(t1 * f147s), # 1
  t2 * 0,
  cos(t1 * f456) + cos(t1 * f2580), # 5
  t2 * 0,
  cos(t1 * f789) + cos(t1 * f369s), # 9
  t2 * 0)
N <- length(x)
t <- (1:N)*step
csplot(t, x)
```




```
X <- fft(x)
csplot(t[1:(N/2)], Mod(X)[1:(N/2)])
```



4.8.2 Short-Time Fourier Transform

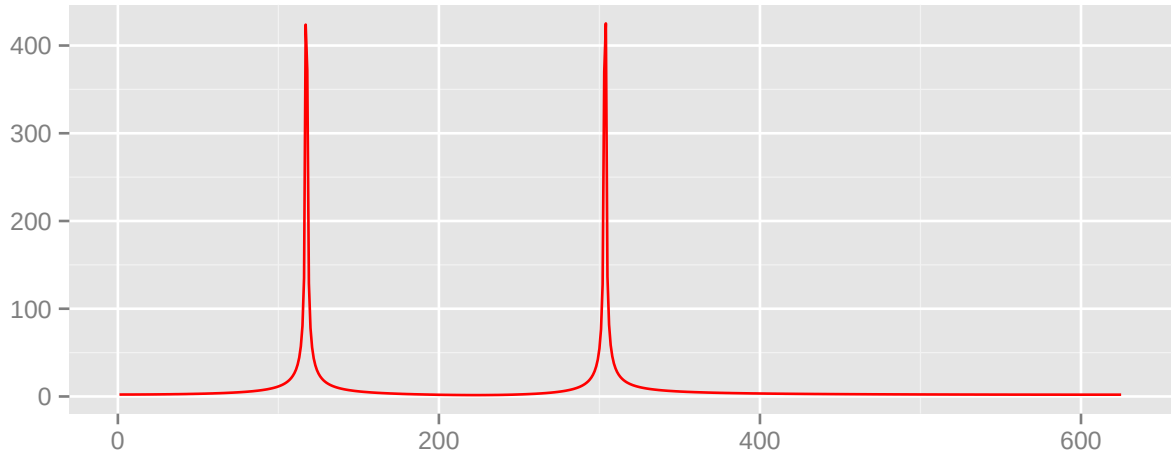
Idea:

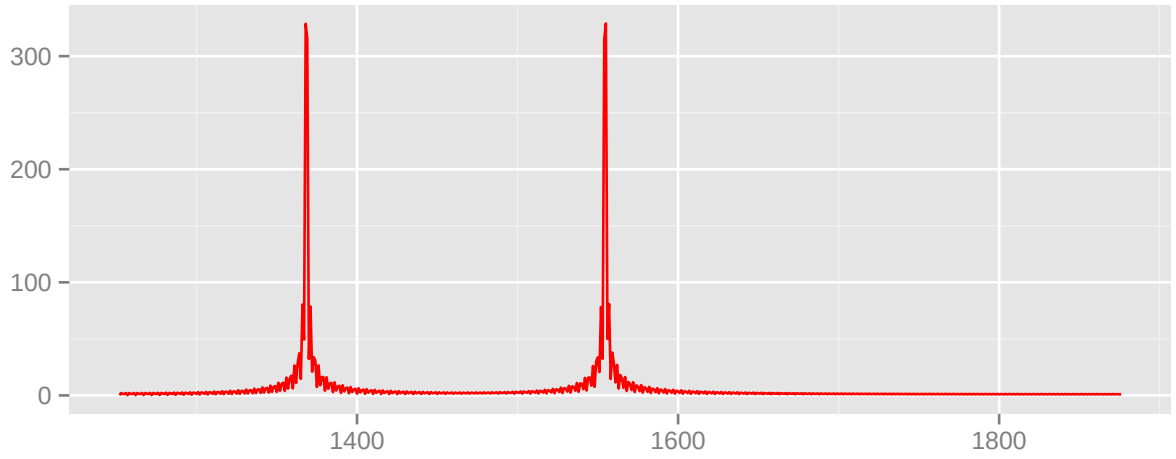
- take small signal pieces of length L
- look at the DFT of each piece:

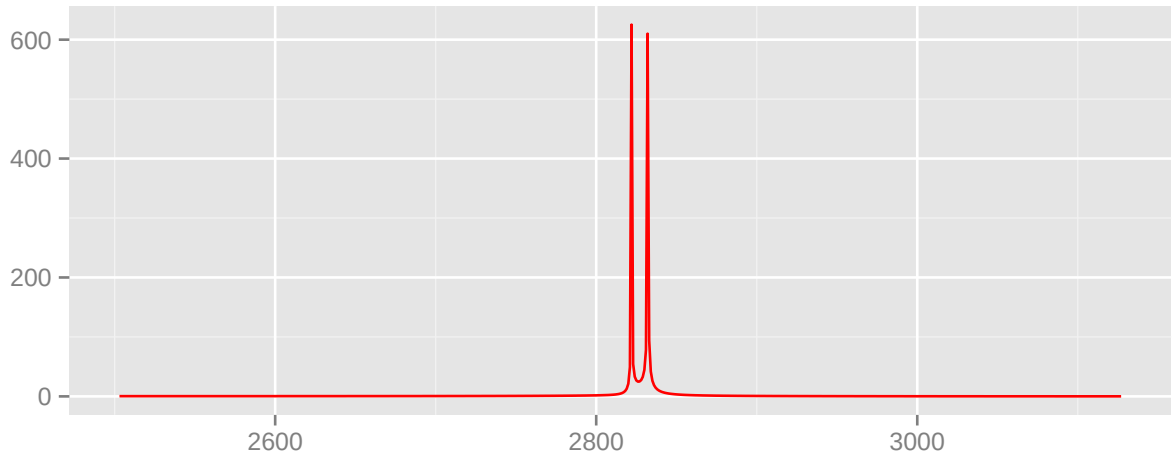
$$X[m;k] = \sum_{n=0}^{L-1} x[m+n]e^{-j\frac{2\pi}{L}nk}$$

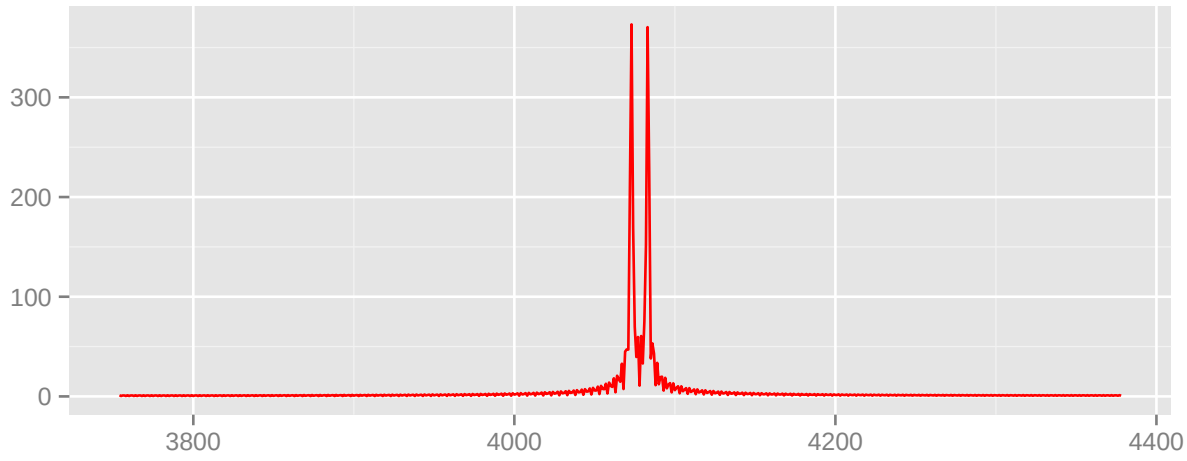
```
count <- 6
L <- N / count
for (i in 1:count) {
  subt <- (L*(i - 1)+1):(L*i)
  X <- fft(x[subt])
```

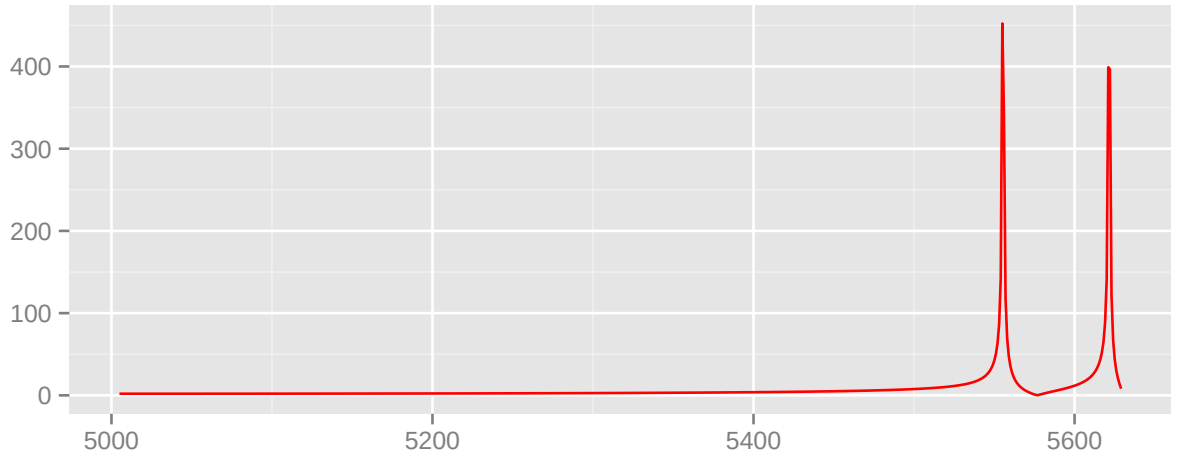
```
  print(csplot(subt[1:(L/2)], Mod(X)[1:(L/2)]))
}
```

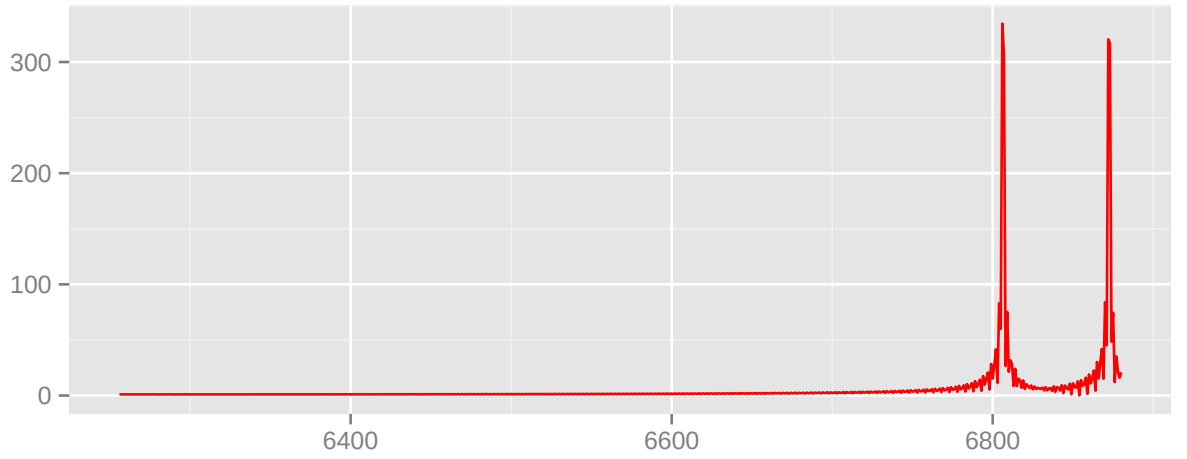







300
200
100
0
3800
4000
4200
4400


400
300
200
100
0
5000
5200
5400
5600

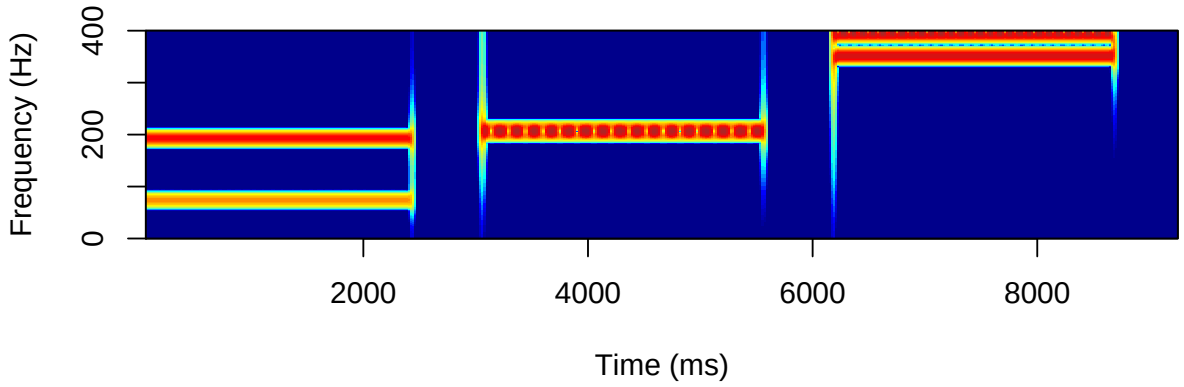

300
200
100
0
6400
6600
6800

4.8.3 Spectrogram

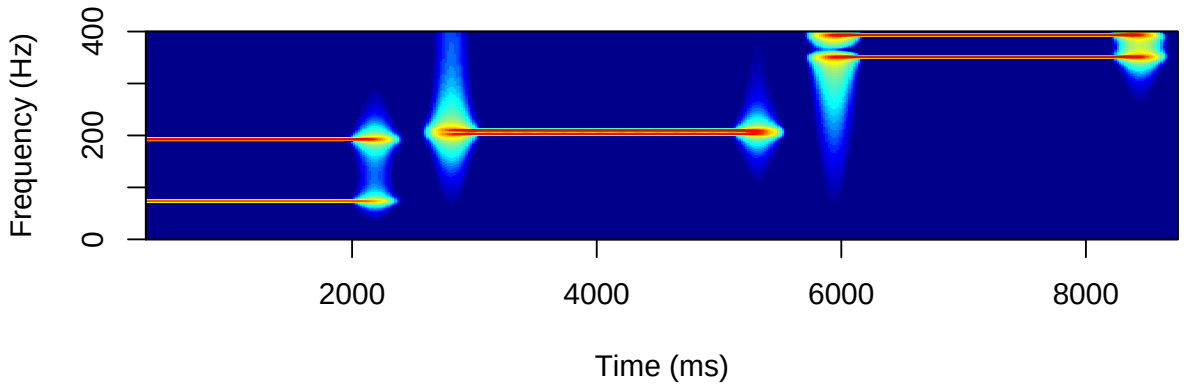
Idea:

- color-code the magnitude: dark is small, white is large
- use $10\log_{10}(|X[m;k]|)$ to see better (pwer id dBs)
- plot spectral slices one after another

```
spectrogram(x, fs=800, windowlength=N/60)
```




```
spectrogram(x, fs=800, windowlength=N/12)
```



4.9 FFT: history and algorithm

4.9.1 History

- Gauss computes trigonometric series efficiently in 1805
- Fourier invents Fourier series in 1807
- People start computing Fourier series, and develop tricks
- Good comes up with an algorithm in 1958
- Cooley and Tukey (re)-discover the fast Fourier transform algorithm in 1965 for N a power of a prime
- Winograd comines all methods to give the most efficient FFTs

4.9.2 The DFT matrix

- $W_N = e^{-j\frac{2\pi}{N}}$ (or simply W when N is clear from the context)
- power of N can be taken modulo N, since $W^N = 1$.
- DFT Matrix of size N by N:

$$W = \begin{bmatrix} 1 & 1 & 1 & 1 & \dots & 1 \\ 1 & W^1 & W^2 & W^3 & \dots & W^{N-1} \\ 1 & W^2 & W^4 & W^6 & \dots & W^{2(N-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & W^{N-1} & W^{2(N-1)} & W^{3(N-1)} & \dots & W^{(N-1)^2} \end{bmatrix}$$

4.9.3 Divide and Conquer for DFT — One step

Recall: computing $X = W_n x has complexity O(N^2)$
Idea:

- Take a problem of size N where N is a power of 2.
- Cut into two problems of size N/2 that use complexity $N^4/4$ each
- There might be some complexity to recover the full solution, say N.
- The divide-and-conquer solution has complexity $N^2/2 + N$ for one step
- For $N \geq 4$ this is much better then N^2!

4.9.4 Divide and Conquer for DFT — Multiple steps

Divide and conquer can be reapplied

- If it worked once, it will work again (recall, $N = 2^K$)
- the two problems of size $N/2$ into 4 problems of size $N/4$
- There might be some complexity to recover the full solution, say N at each step
- You can do this $\log_2(N-1) = K-1$ times, until problem of size 2 is obtained
- This requires order N complexity each time
- The divide-and-conquer solution has therefor complexity of order $N \log_2 N$
- For $N \geq 4$ this is much better then N^2!

4.9.5 Conclusion

Don't worry, be happy!

- The Cooley-Tukey is the most popular algorithm, mostly for $N = 2^N$
- Note that there is always a good FFT algorithm around the corner
- Do not zero-pad to lengthen a vector to have a size equal to a power of 2
- There are good packages out there (e.g. Fastest Fourier Transform in the West, SPIRAL)
- It does make a BIG difference!

4.10 Why the DFT is useful: A few examples

4.10.1 Overview

- Analysis of musical instruments
- Approximation of periodical phenomena: the case of tides in Venice
- The secret of MP3 compression
- Magnetic Resonance Imaging

Module 5

Linear Filters

5.1 Linear filters

5.1.1 A generic signal processing device

The input signal: $x[n]$, the output signal: $y[n]$.
The device scheme:

$$x[n] \quad \longmapsto \quad \boxed{\mathcal{H}} \quad \longmapsto \quad y[n]$$

The equation:

$$y[n] = \mathcal{H}\{x[n]\}$$

5.1.2 Linearity

$$\mathcal{H}\{\alpha x_1[n] + \beta x_2[n]\} = \alpha\mathcal{H}\{x_1[n]\} + \beta\mathcal{H}\{x_2[n]\}$$

5.1.3 Time invariance

$$y[n] = \mathcal{H}\{x[n]\} \quad \Longleftrightarrow \quad \mathcal{H}\{x[n-n_0]\} = y[n-n_0]$$

5.1.4 Linear, time-invariant (LTI) systems

$$y[n] = H(x[n], x[n-1], x[n-2], \ldots, y[n-1], y[n-2], \ldots)$$

with $H(\cdot)$ a linear function of its arguments

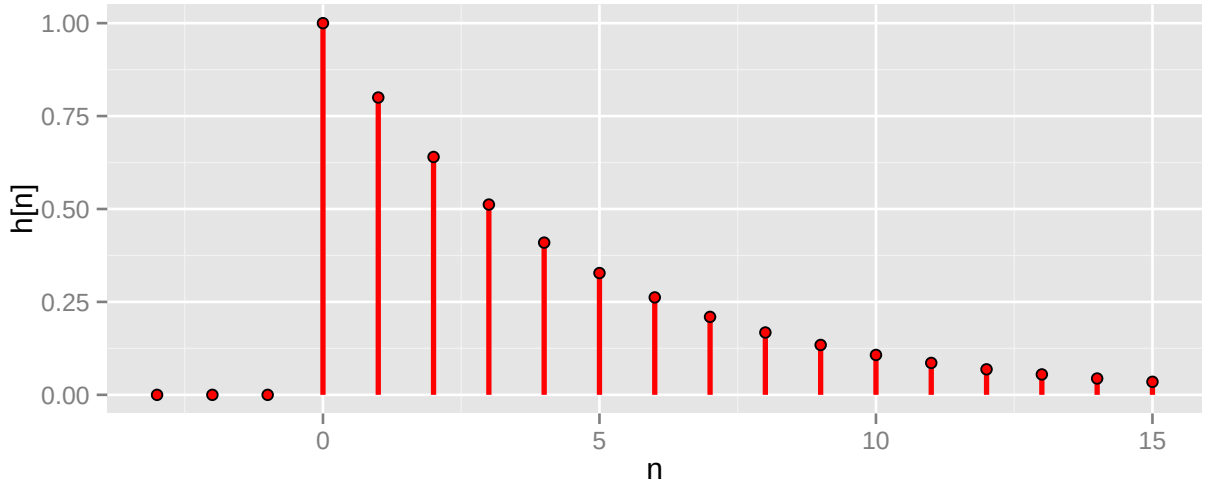
5.1.5 Impulse response

$$h[n] = \mathcal{H}\{\delta[n]\}$$

Fundamental results: impulse response fully characterizes the LTI system!
An example

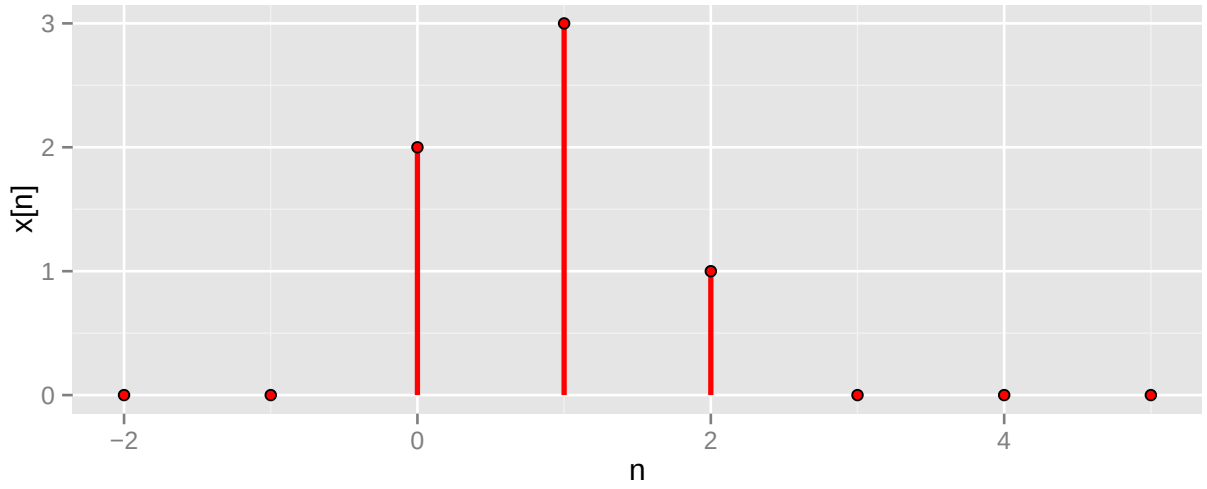
$$h[n] = \alpha^n u[n]$$

```
alpha <- 0.8
n <- -3:15
h <- c(0, 0, 0, alpha^(0:15))
dsplot(n, h, xlab="n", ylab="h[n]")
```



$$x[n] = \begin{cases} 2 & n = 0 \\ 3 & n = 1 \\ 1 & n = 2 \\ 0 & otherwise \end{cases}$$

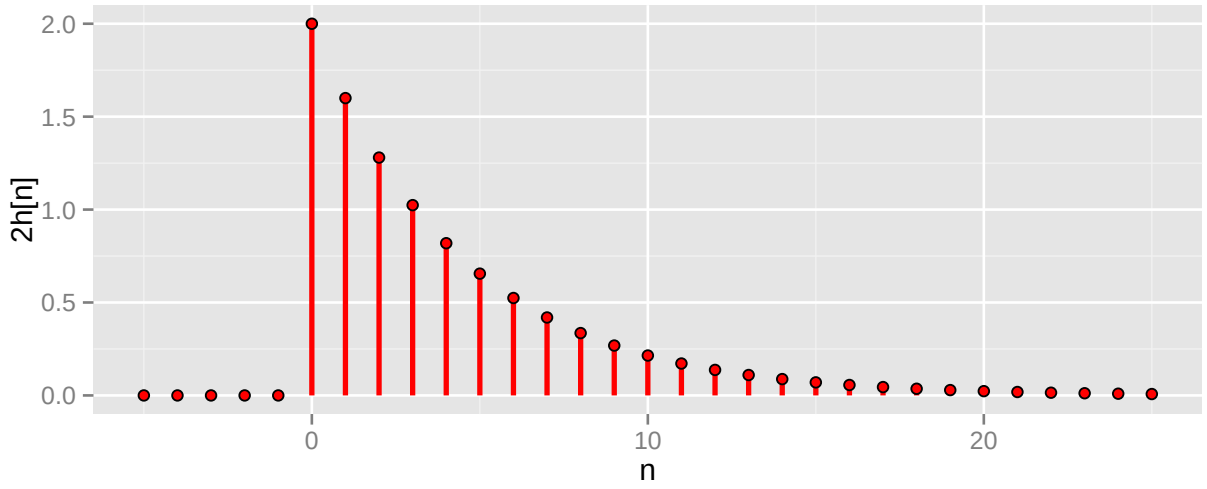
```
n <- -2:5
x <- c(0, 0, 2, 3, 1, 0, 0, 0)
dsplot(n, x, xlab="n", ylab="x[n]")
```



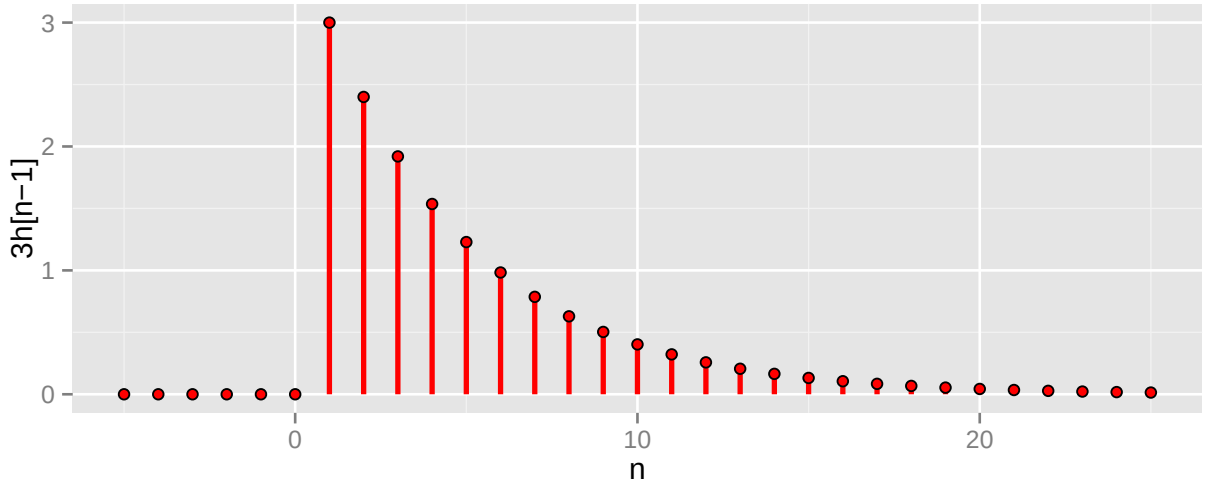
- $x[n] = 2\delta[n] + 3\delta[n-1] + \delta[n-2]$
- we know the impulse response $h[n] = \mathcal{H}\{\delta[n]\}$
- compute $y[n] = \mathcal{H}\{x[n]\}$ exploiting linearity and time-invariance

$$\begin{aligned} y[n] = & \quad \mathcal{H}\{2\delta[n] + 3\delta[n-1] + \delta[n-2]\} \\ = & \quad 2\mathcal{H}\{\delta[n]\} + 3\mathcal{H}\{\delta[n-1]\} + \mathcal{H}\{\delta[n-2]\} = \\ = & \quad 2h[n] + 3h[n-1] + h[n-2] \end{aligned}$$

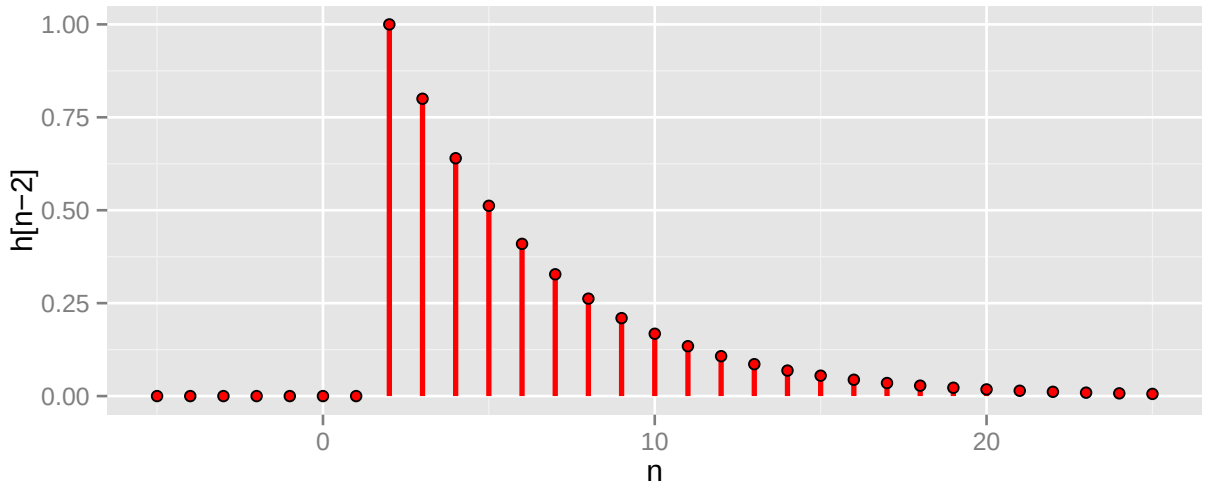
```
n <- -5:25
h <- c(0, 0, 0, 0, 0, alpha^(0:25))
shift <- function(s, z) c(rep(0, z), s[1:(length(s)-z)])
dsplot(n, 2*h, xlab="n", ylab="2h[n]")
```



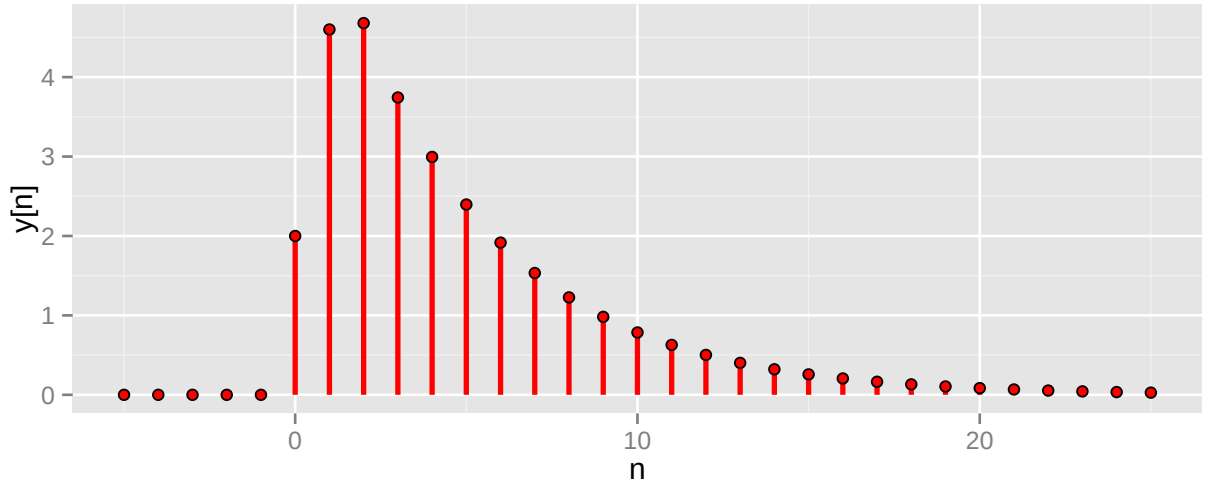

```
dsplot(n, 3*shift(h,1), xlab="n", ylab="3h[n-1]")
```




```
dsplot(n, shift(h,2), xlab="n", ylab="h[n-2]")
```




```
y <- 2 * h + 3 * shift(h,1) + shift(h,2)
dsplot(n, y, xlab="n", ylab="y[n]")
```



5.1.6 Convolution

We can always write (remember Module 3.2):

$$x[n] = \sum_{k=-\infty}^{\infty} x[k]\delta[n-k]$$

by linearity and time invariance:

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k] = x[n] \cdot h[n]$$

Performing the convolution algorithmically

$$x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]$$

Ingredients:

- a sequence $x[m]$
- a second sequence $h[m]$

The recipe:

- time-reverse $h[m]$
- at each step nn (from $-\infty$ to ∞):
  - center the time-reversed $h[m]$ in n (i.e. shift by $-n$)
  - compute the inner product

5.1.7 Convolution properties

- linearity and time invariance (by definition)
- commutativity: $(x * h)[n] = (h * x)[n]$
- associativity for absolutely- and square-summable sequences: $((x * h) * w)[n] = (x * (h * w))[n]$

5.2 Filtering by example

5.2.1 Donoising my Moving Average

- idea: replace each sample by the local average
- for instance: $y[n] = (x[n] + x[n-1])/2$
- more generally:

$$y[n] = \frac{1}{M} \sum_{k=0}^{M-1} x[n-k]$$

5.2.2 MA: impulse response

$$h[n] = \frac{1}{M} \sum_{k=0}^{M-1} \delta[n-k] = \begin{cases} 1/M & \text{for } 0 \leqslant n < M \\ 0 & \text{otherwise} \end{cases}$$

5.2.3 MA: analysis

- smoothing effect proportional to M
- number of operations and storage also proportional to M

5.2.4 From the MA to a first-order recursion

$$y_M[n] = \frac{1}{M} x[n] + \frac{1}{M}(x[n-1] + x[n-2] + \ldots + x[n-M+1])$$

where $(x[n-1] + x[n-2] + \ldots + x[n-M+1])$ is "almost" $y_{M-1}[n-1]$ i.e., moving average over $M-1$ points, delayed by one.

Formally:

$$y_M[n] = \frac{1}{M}\sum_{k=0}^{M-1} x[n-k]$$

$$y_M[n-1] = \frac{1}{M}\sum_{k=1}^{M} x[n-k]$$

$$y_{M-1}[n] = \frac{1}{M-1}\sum_{k=1}^{M-2} x[n-k]$$

$$y_{M-1}[n-1] = \frac{1}{M-1}\sum_{k=1}^{M-1} x[n-k]$$

$$\sum_{k=0}^{M-1} x[n-k] = x[n] + \sum_{k=1}^{M-1} x[n-k]$$

$$My_M[n] = x[n] + (M-1)y_{M-1}[n-1]$$

$$y_M[n] = \frac{M-1}{M}y_{M-1}[n-1] + \frac{1}{M}x[n]$$

$$y_M[n] = \lambda y_{M-1}[n-1] + (1-\lambda)x[n], \quad \lambda = \frac{M-1}{M}$$

5.2.5 The Leaky Integrator

- when M is large, $y_{M-1}[n] \approx y_M[n]$ (and $\lambda \approx 1$)
- try the filter
$$y[n] = \lambda y[n-1] + (1-\lambda)x[n]$$
- filter is now recursive, since it uses its previous output value

5.2.6 What about the impulse response?

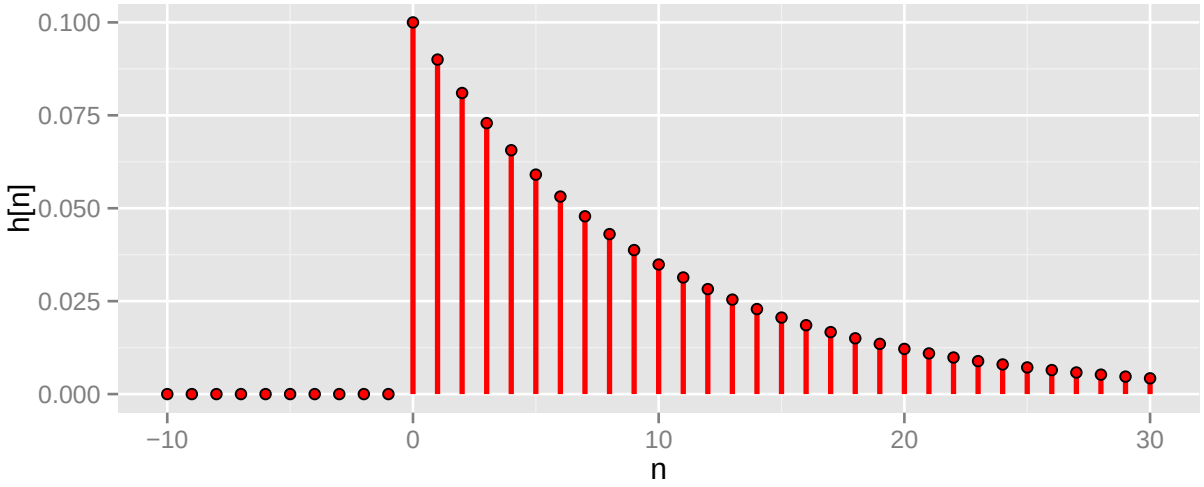
$$y[n] = \lambda y[n-1] + (1-\lambda)\delta[n]$$

- $y[n] = 0$ for all $n < 0$
- $y[0] = \lambda y[-1] + (1-\lambda)\delta[0] = (1-\lambda)$
- $y[1] = \lambda y[0] + (1-\lambda)\delta[1] = \lambda(1-\lambda)$
- $y[2] = \lambda y[1] + (1-\lambda)\delta[2] = \lambda^2(1-\lambda)$
- $y[3] = \lambda y[2] + (1-\lambda)\delta[3] = \lambda^3(1-\lambda)$
- ...

Impulse response:

$$h[n] = (1-\lambda)\lambda^n u[n]$$

```
lambda <- 0.9
n <- -10:30
h <- c(rep(0, 10), (1-lambda)*lambda^(0:30))
dsplot(n, h, xlab="n", ylab="h[n]")
```



5.2.7 Leaky Integrator: why the name

Discrete-time integrator is a boundless accumulator:

$$y[n] = \sum_{k=-\infty}^{n} x[k]$$

We can rewrite the integrator as

$$y[n] = y[n-1] + x[n]$$

To prevent "explosion" pick $\lambda < 1$

$$y[n] = \lambda y[n-1] + (1+\lambda)x[n]$$

- $\lambda y[n-1]$: keep only a fraction λ of the accumulated value so far and forget ("leak") a fraction $1-\lambda$
- $(1+\lambda)x[n]$: add only a fraction $1-\lambda$ of the current value to the accumulator

5.3 Filter stability

5.3.1 Filter types according to impulse response

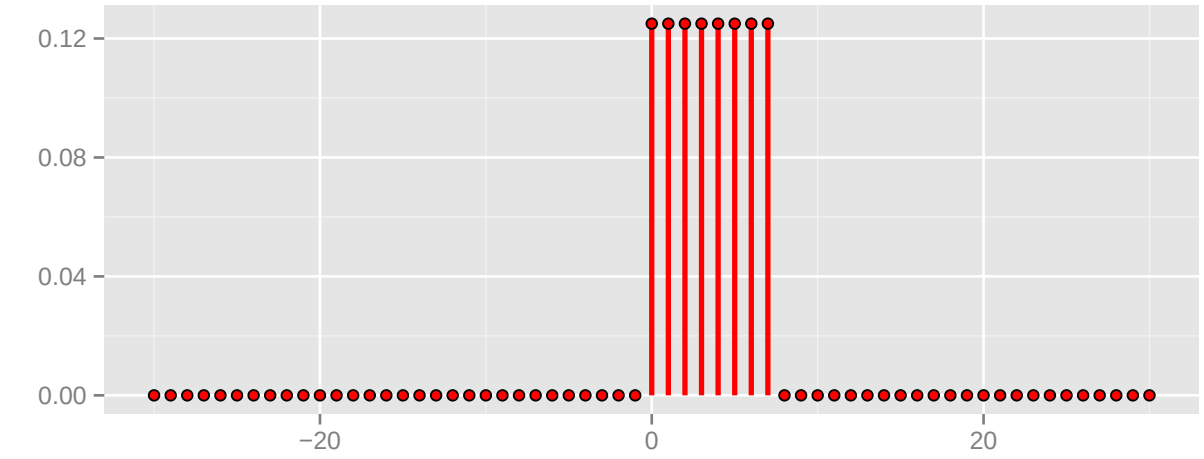
- Finite Impulse Response (FIR)
- Infinite Impulse Response (IIR)
- causal
- noncausal

5.3.2 FIR

- impulse response has finite support
- only a finite number of samples are involved in the computation of each output sample

An example: Moving Average filter

```
M <- 8
n <- -30:30
x <- ifelse(n >= 0 & n < M, 1/M, 0)
dsplot(n, x)
```

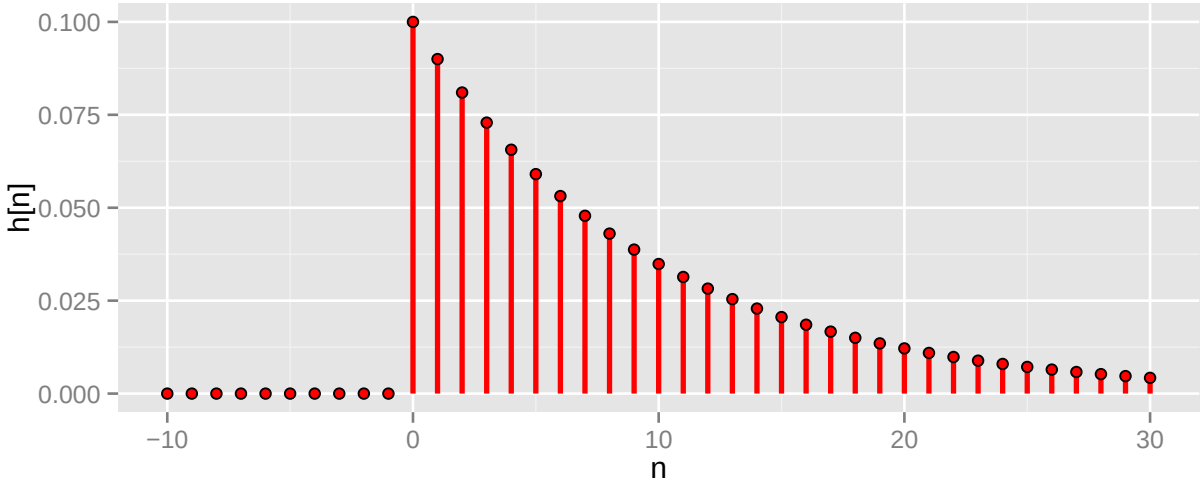


5.3.3 IIR

- impulse response has infinite support
- a potentially infinite number of samples are involved in the computation of each output sample
- surprisingly, in many cases the computation can still be performed in a finite amount of steps

An example: Leaky Integrator

```
lambda <- 0.9
n <- -10:30
h <- c(rep(0, 10), (1-lambda)*lambda^(0:30))
dsplot(n, h, xlab="n", ylab="h[n]")
```



5.3.4 Causal vs Noncausal

- causal:
  - impulse response is zero for $n < 0$
  - only past samples (with respect to the present) are involved in the computation of each output sample
  - causal filters can work "on line" since they only need the past
- noncausal:
  - impulse response is nonzero for some (or all) $n < 0$
  - can still be implemented in a offline fashion (when all input data is available on storage, e.g. in Image Processing)

5.3.5 Stability

- key concept: avoid "explosions" if the input is nice
- a nice signal is a bounded signal: $|x[n]| < M$ for all n
- Bounded-Input Bounded-Output (BIBO) stability: if the input is nice the output should be nice

5.3.6 Fundamental Stability Theorem

A filter is BIBO stable if and only if its impulse response is absolutely summable
Proof $\Rightarrow$
Hypotheses: $|x[n]| < M$ and $\sum_n |h[n]| = L < \infty$
Thesis: $|y[n]|$ bounded
Proof:

$$|y[n]| = \Big| \sum_{k=-\infty}^{\infty} h[k]x[n-k] \Big| \leqslant \sum_{k=-\infty}^{\infty} |h[k]x[n-k]| \leqslant M \sum_{k=-\infty}^{\infty} |h[k]| \leqslant ML$$

Proof $\Leftarrow$
Hypotheses: $|x[n]| < M$ and $|y[n]| < P$
Thesis: $h[n]$ absolutely summable
Proof (by contradiction):

- assume $\sum_n |h[n]| = \infty$
- build $x[n] = \begin{cases} +1 & \text{if} h[-n] \geqslant 0 \\ -1 & \text{if} h[-n] < 0 \end{cases}$
- clearly, $x[n]$ is bounded
- however

$$y[0] = (x * h)[0] = \sum_{k=-\infty}^{\infty} h[k]x[-k] = \sum_{k=-\infty}^{\infty} |h[k]| = \infty$$

The good news: FIR filters are always stable

5.3.7 Checking the stability of IIRs

Let's check the Leaky Integrator:

$$\sum_{n=-\infty} \infty |h[n]| = |1-\lambda| \sum_{n=0}^{\infty} |\lambda|^n = \lim_{n\to\infty} |1-\lambda| \frac{1-|\lambda|^{n+1}}{1-|\lambda|} < \infty \text{ for } |\lambda| < 1$$

stability is guaranteed for $|\lambda| < 1$.
We will study indirect methods for filter stability later in this Module.