TÉCNICO LISBOA

TÉCNICO LISBOA

# Principles of Biosignals and Biomedical Imaging

3rd year, $P_3$ (ECTS: 3.0), LEBiom

2022/2023

João Sanches
jmrs@tecnico.ulisboa.pt
Department of Bioengineering
Instituto Superior Técnico
University of Lisbon

TÉCNICO LISBOA

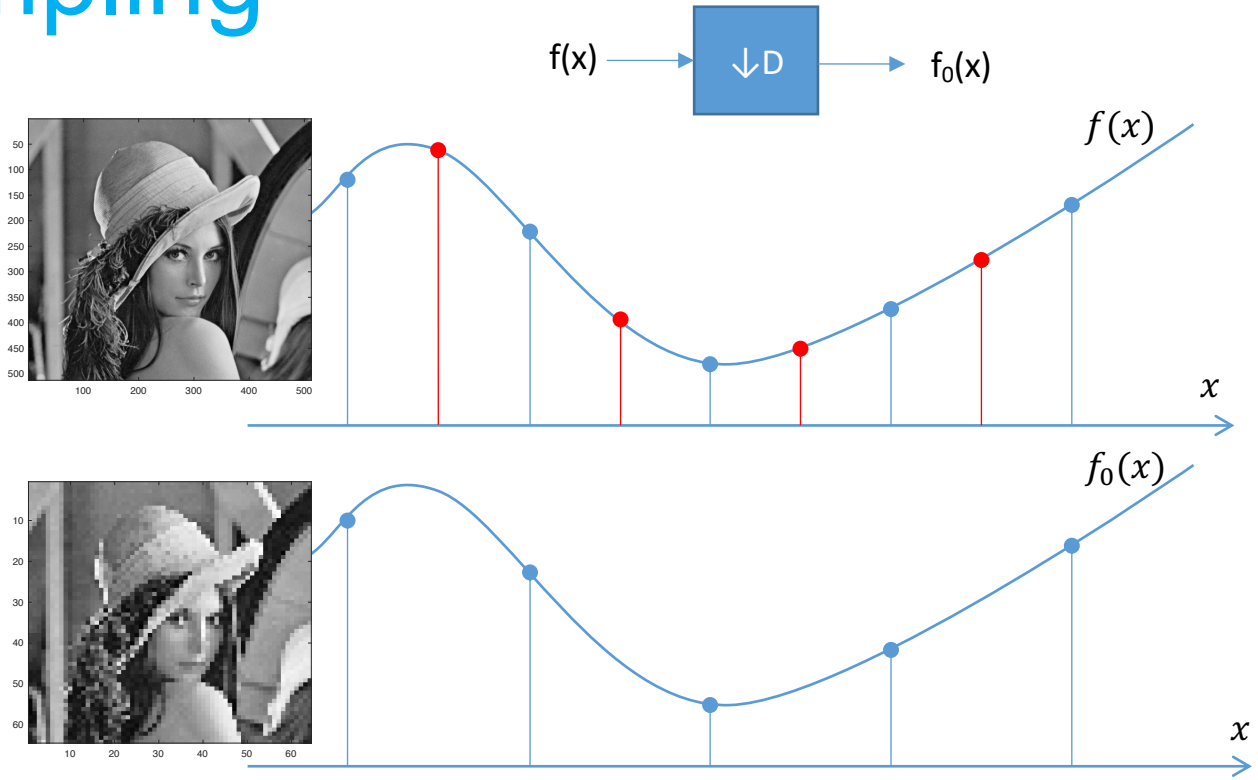
# Resizing

## DownSampling and Decimation
## UpSampling and Interpolation

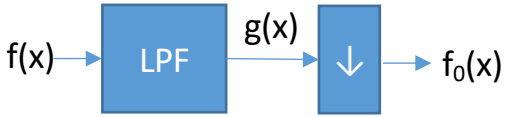
# Downsampling

- Downsampling ratio D: D-1 out D are removed
- Aliasing problem

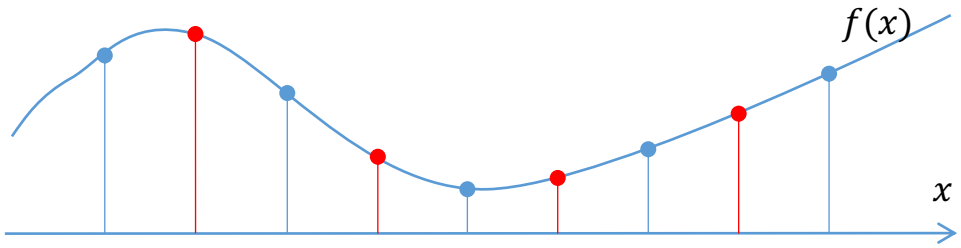
```
% Original
x=double(imread('len_gray.jpg'));
D=8;
xDow=x(1:D:end, 1:D:end);
```



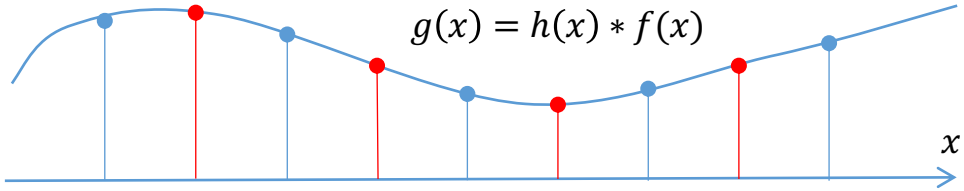
# Decimation

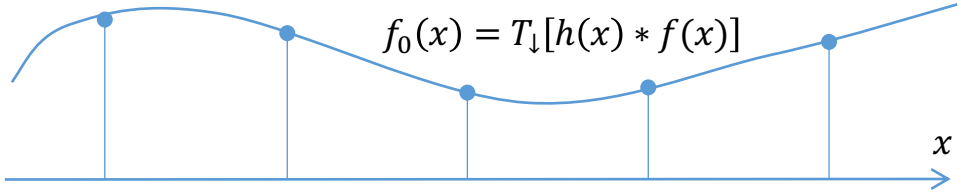



- Downsampling ratio D: D-1 out D are removed
- Aliasing problem



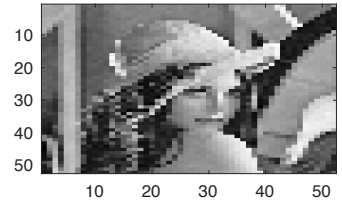

```
%% Decimation
xDec=zeros(size(x));

%filter design and definition
B=fir1(100,0.1);

%filtering
xLP=conv2(B,B',x,'same');

%Downsampling
xDec=xLP(1:D:end, 1:D:end);
```



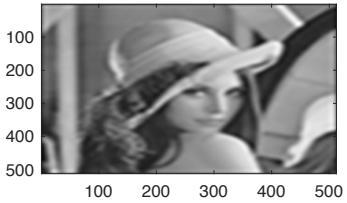




**Original**

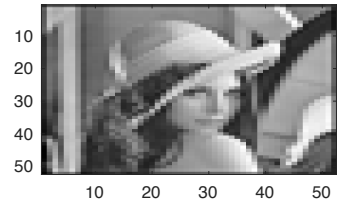

**Downsampling**




**Antialiasing FIR filtered**

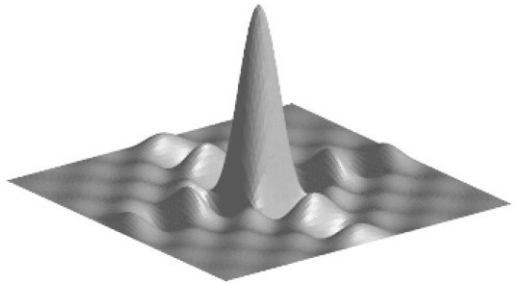



**Decimation**



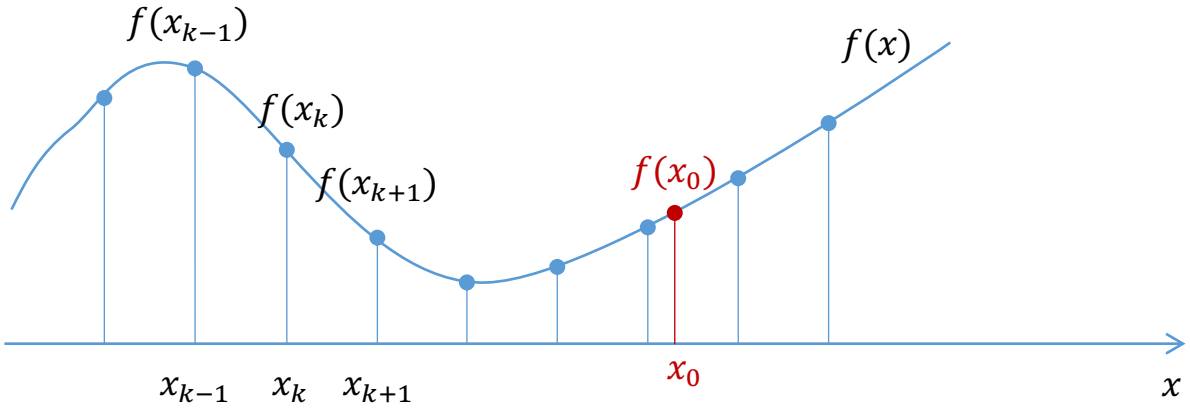
TÉCNICO LISBOA

# Interpolation



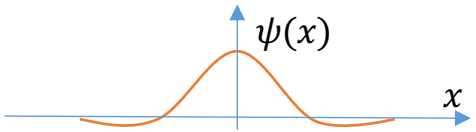
# Interpolation

Problem:
Given a set of samples of a continuous function f(x), $y_k=f(x_k)$, compute $f(x_0)$ at an arbitrary location $x_0$



# Interpolation functions

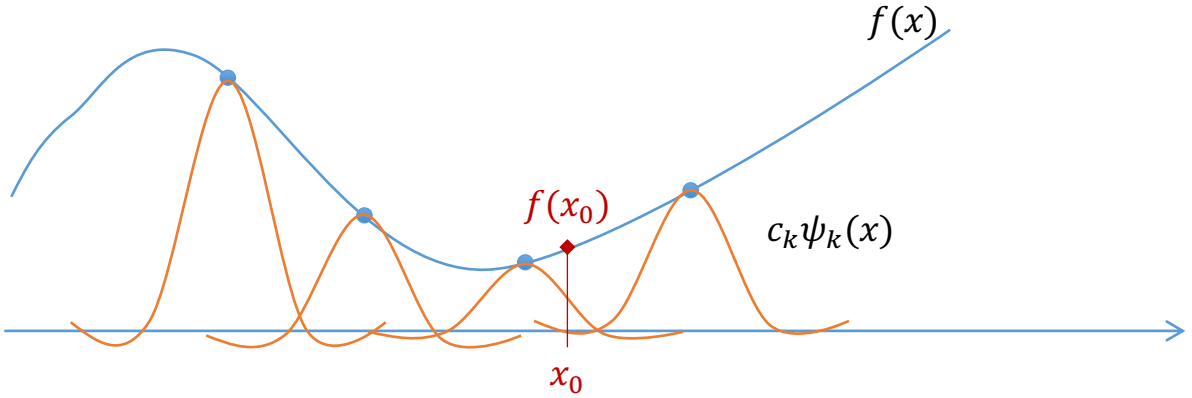
How to represent continuous functions from a set of discrete values – discrete signals



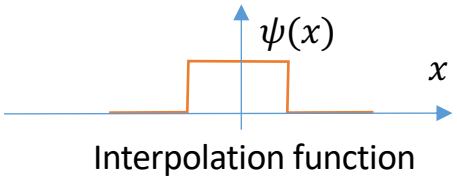

Interpolation function

$$f(x) = \sum_k c_k \psi_k(x)$$

$$\psi_k(x) = \psi(\alpha_k x - \beta_k)$$



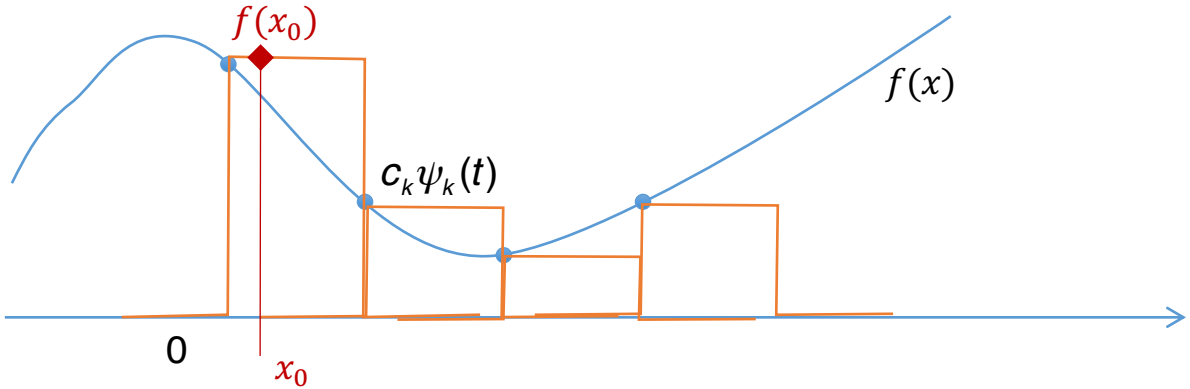
# Zero order



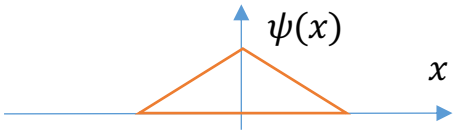

Interpolation function

$$f(x) = \sum_k c_k \psi_k(x)$$

$$\psi_k(x) = \psi(\alpha_k x - \beta_k)$$



# Piecewise linear interpolation

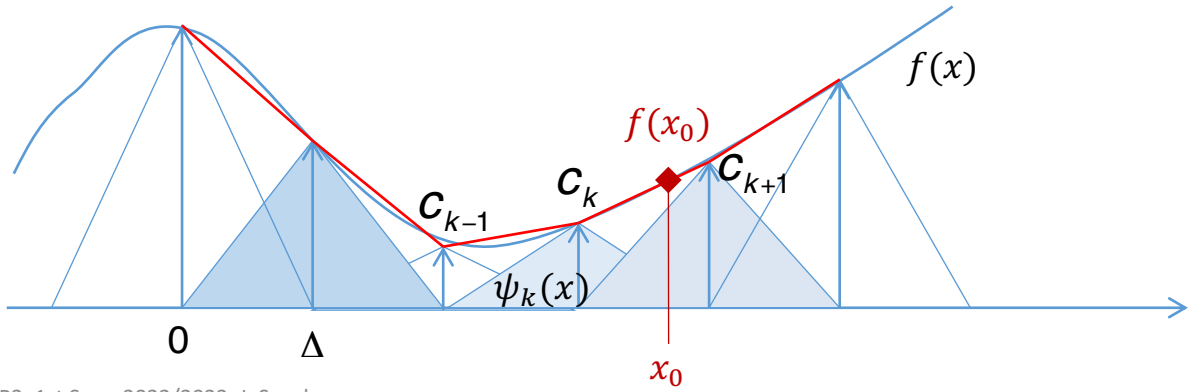



Interpolation function

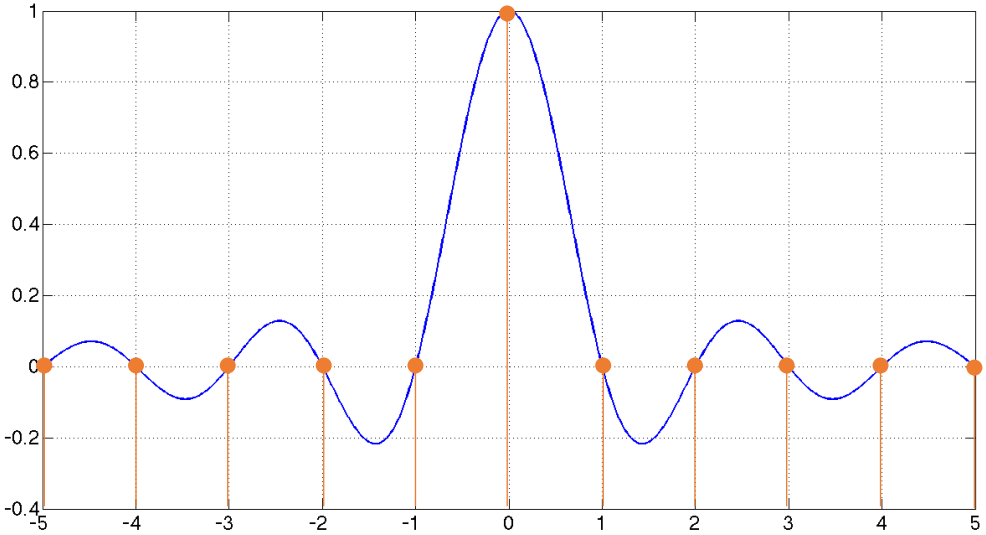
$$\psi(x) = \begin{cases} 1+x & -1 \le x < 0 \\ 1-x & 0 \le x \le 1 \\ 0 & otherwise \end{cases}$$

$$f(x) = \sum_k c_k \psi_k(x)$$

$$\psi_k(x) = \psi(\alpha_k x - \beta_k)$$



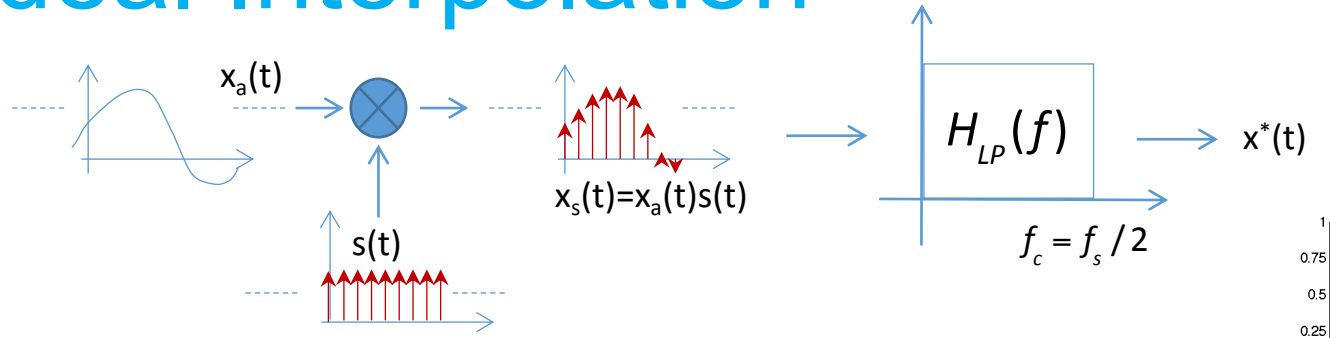
# (Ideal?) Sinc basis functions

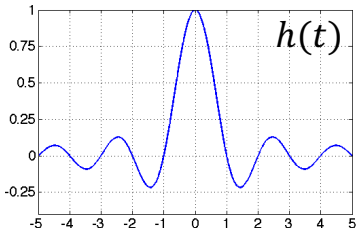


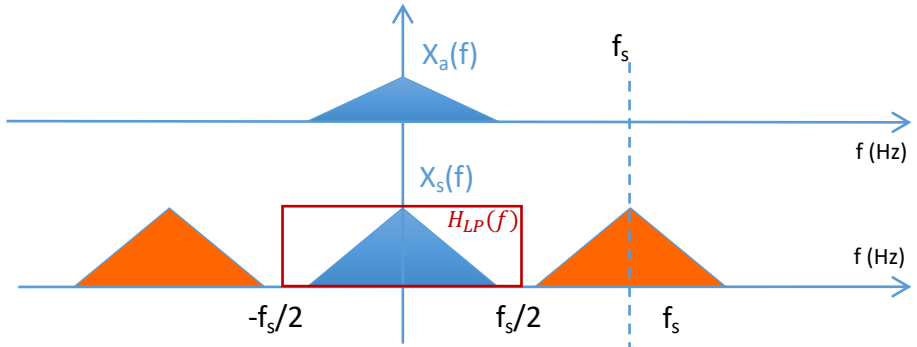

$$\psi(x) = \frac{\sin(\pi x)}{\pi x}$$

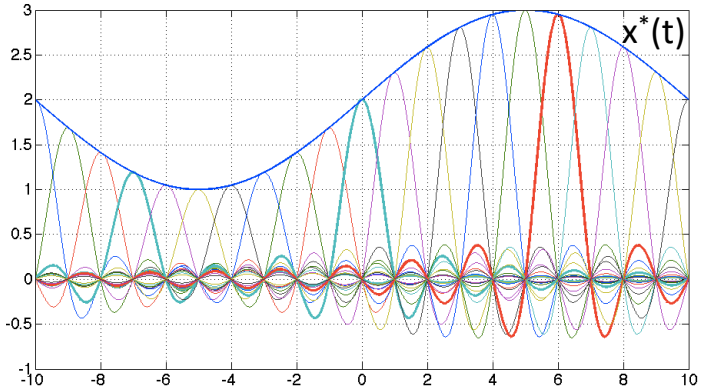
$$\psi_k(x) = sinc\left(\frac{x}{\Delta} - k\right)$$

# Ideal Interpolation



TÉCNICO LISBOA

# Interval

TÉCNICO LISBOA

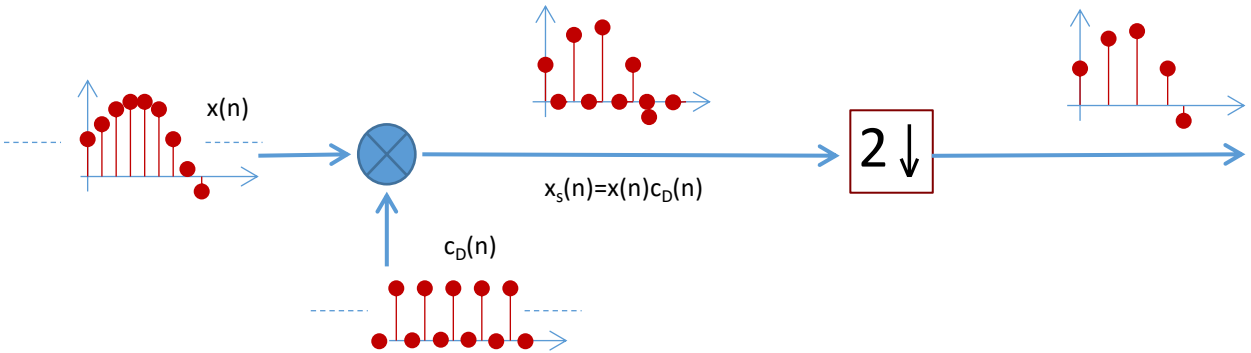
# Digital sampling

# Down-sampling

Sample removal

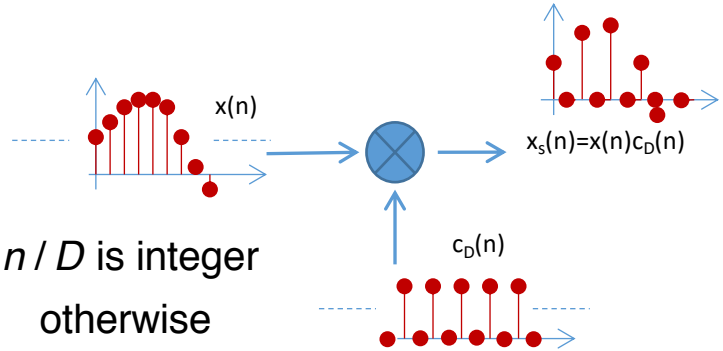
=

Ideal Sampling + zero sample removal



# Ideal discrete sampling

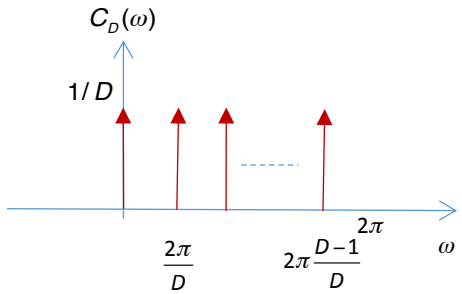
- Dirac combo/train




$$c_D(n)=\frac{1}{D}\sum_{k=0}^{D-1}e^{j\frac{2\pi}{D}kn}=\begin{cases}1 & \text{if } n/D \text{ is integer}\\ 0 & \text{otherwise}\end{cases}$$

Which means that the spectral components are complex exponentials with frequencies $\omega_k = 2\pi k/D$ with $k=0,\ldots,D-1$.

$$C_D(\omega)=\frac{1}{D}\sum_{k=0}^{D-1}\delta\left(\omega-\frac{2\pi}{D}k\right)$$

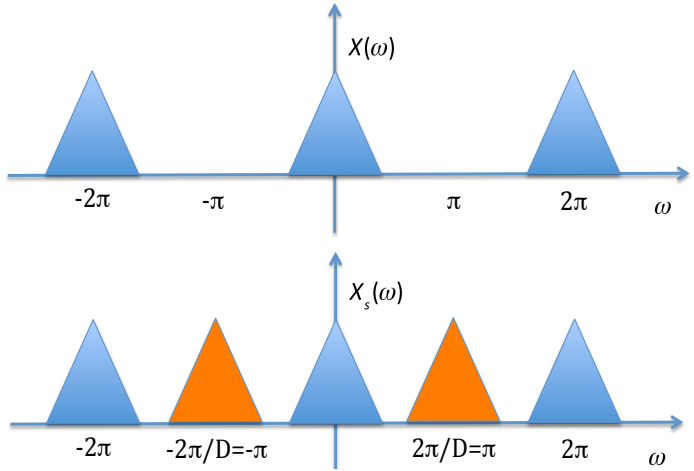


# Ideal discrete sampling

By the convolution theorem:

$$FT\left[x_s(n) = x(n)c_D(n)\right] = X(\omega) * C_D(\omega)$$

$$c(n) = \frac{1}{D}\sum_{k=0}^{D-1} e^{j\frac{2\pi}{D}kn} \qquad X_s(\omega) = \frac{1}{D}\sum_{k=0}^{D-1} X\left(\omega - \frac{2\pi}{D}k\right)$$



# Down-sampling

$2\downarrow$

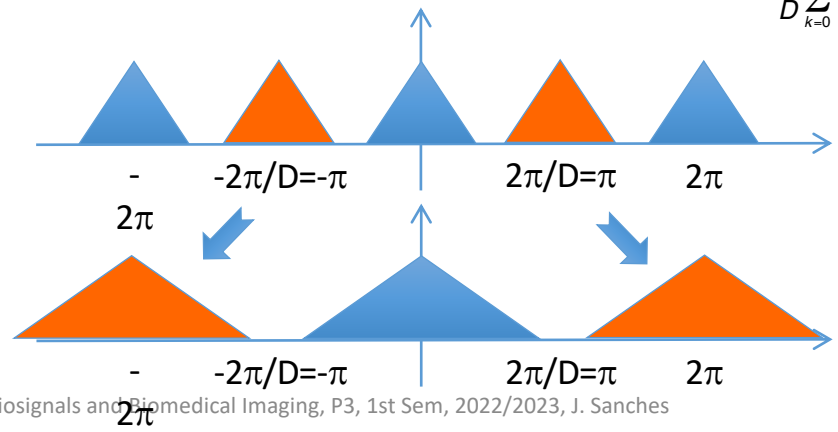
$$y(n) = x(nD) \qquad \text{(zeros discarded)}$$

$$Y(\omega) = \ldots + \underbrace{y(0)}_{x(0)} + \underbrace{y(1)e^{-j\omega}}_{x(D)} + \ldots + \underbrace{y(n)e^{-j\omega n}}_{x(nD)} + \ldots = \sum_n x(nD)e^{-j\omega n} = \sum_m x_s(m)e^{-j\omega m/D} = X_s(\omega / D)$$

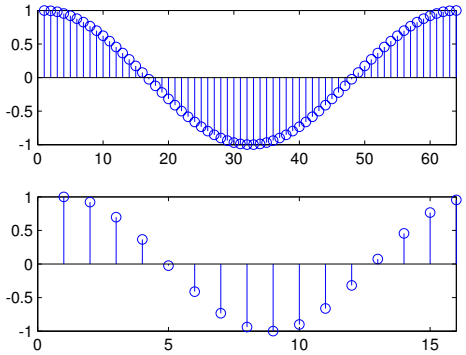
$$Y(\omega) = X_s(\omega / D) = \frac{1}{D}\sum_{k=0}^{D-1} X\left(\frac{\omega}{D} - \frac{2\pi k}{D}\right) = \frac{1}{D}\sum_{k=0}^{D-1} X\left(\frac{\omega - 2\pi k}{D}\right)$$

$$D=2: \frac{1}{D}\sum_{k=0}^{D-1} X\left(\frac{\omega - 2\pi k}{D}\right) \underset{\downarrow 2}{\rightarrow} \frac{1}{2}\left[X\left(\frac{\omega}{2}\right) + X\left(\frac{\omega}{2} - \pi\right)\right]$$

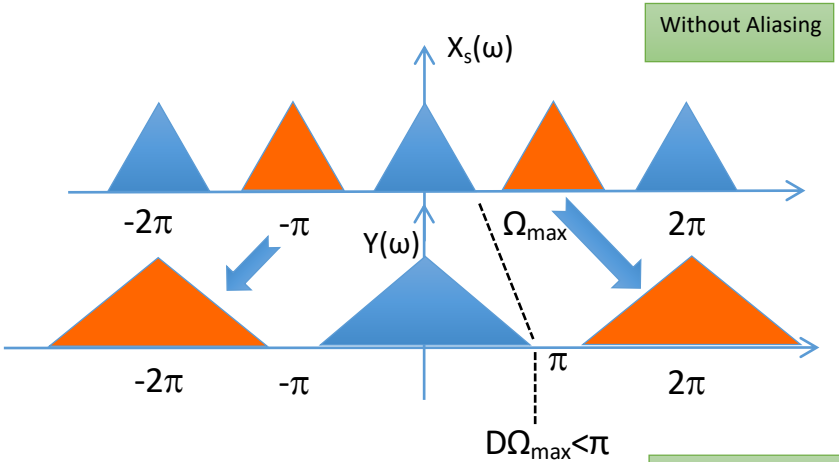
$$D=3: \frac{1}{D}\sum_{k=0}^{D-1} X\left(\frac{\omega - 2\pi k}{D}\right) \underset{\downarrow 2}{\rightarrow} \frac{1}{3}\left[X\left(\frac{\omega}{3}\right) + X\left(\frac{\omega}{3} - \frac{2}{3}\pi\right) + X\left(\frac{\omega}{3} - \frac{4}{3}\pi\right)\right]$$

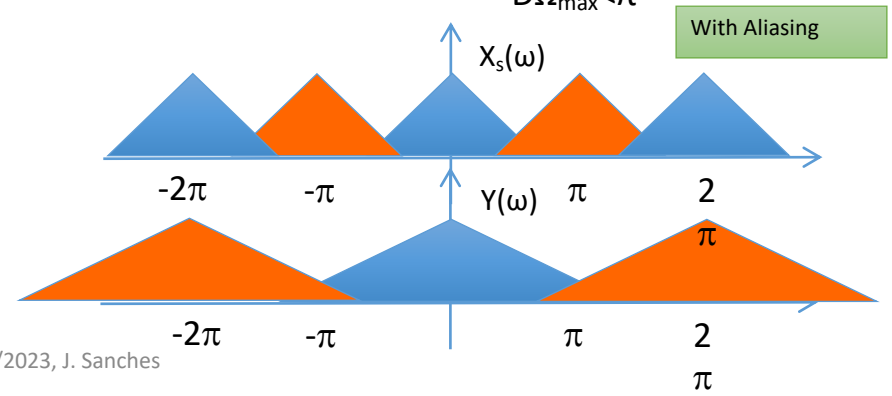


# Digital Aliasing

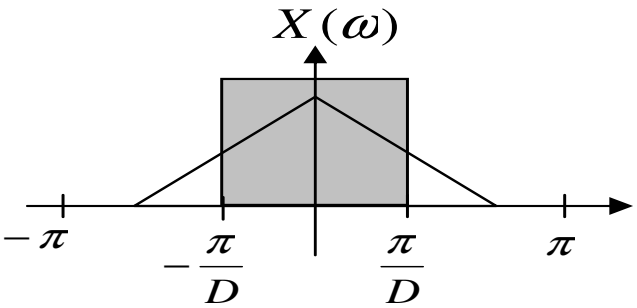


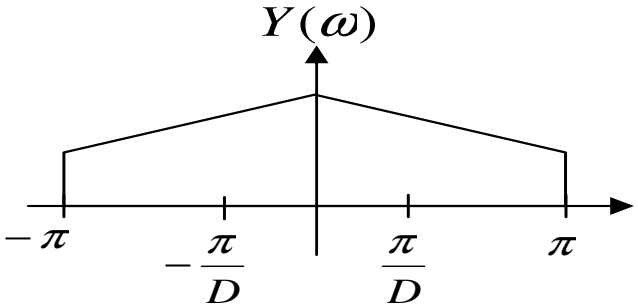

$$Y(\omega) = \frac{1}{D}\sum_{k=0}^{D-1} X\left(\frac{\omega - 2\pi k}{D}\right)$$

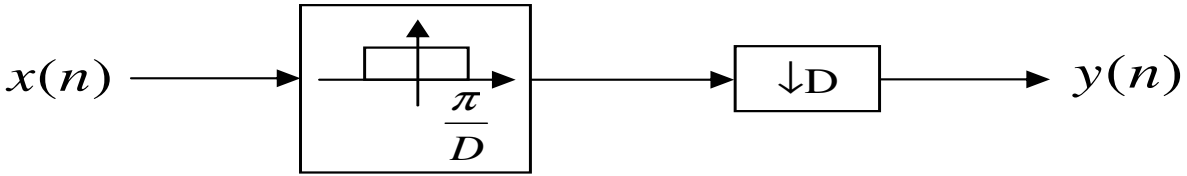


# Decimation



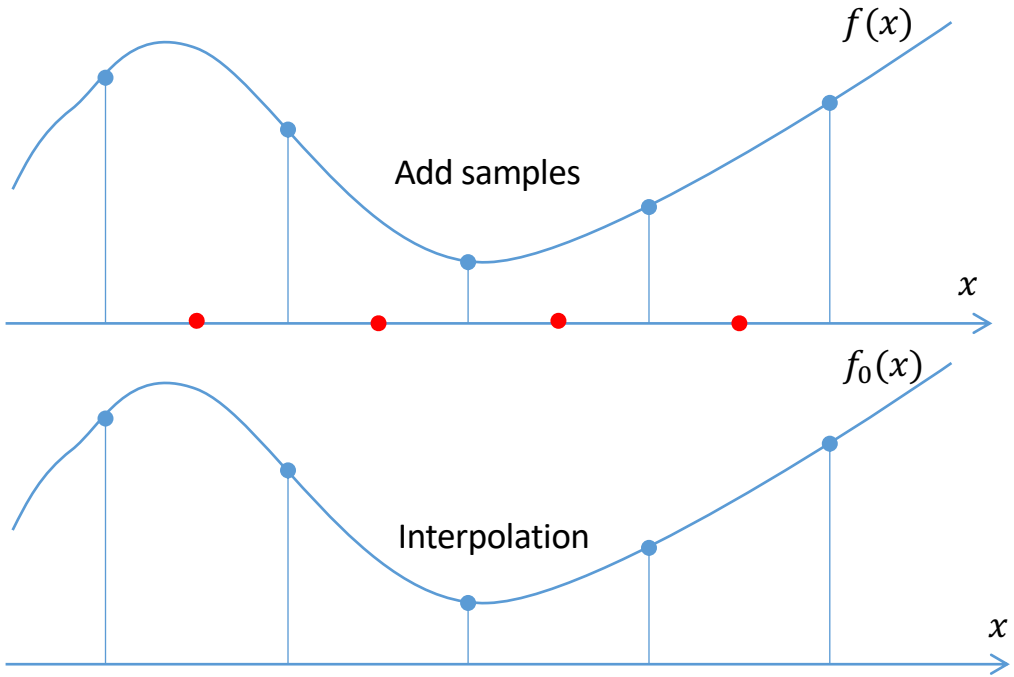




Filtro Passa Baixo

# Up-sampling

2↑

- Upsampling ratio L:
  L-1 new zero samples per each sample are added
- The new samples are interpolated
- No aliasing problem

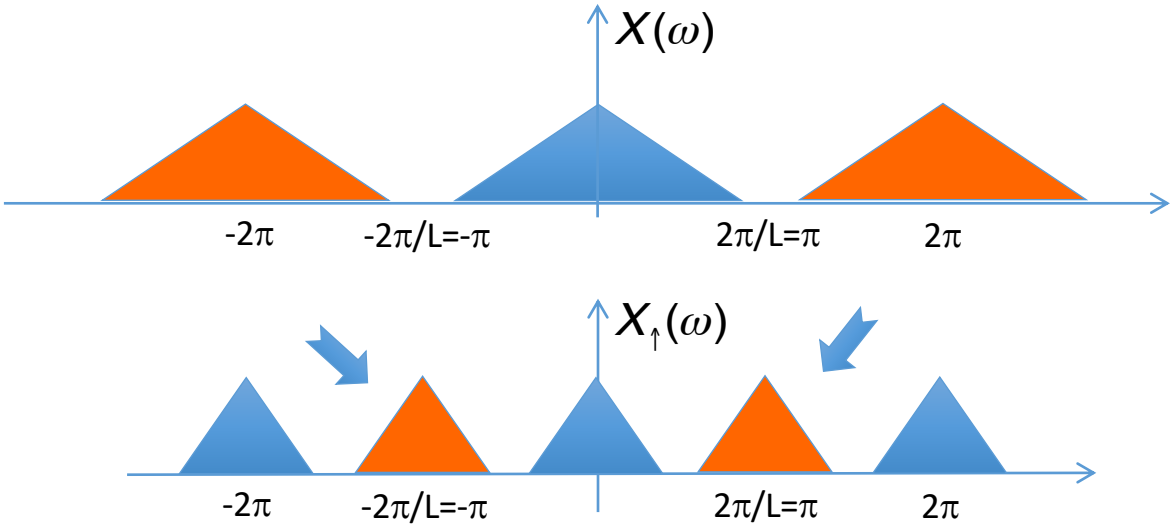



```
%% Upsampling
L=10;
xUp=zeros(L*size(xDec));
xUp(1:L:end,1:L:end)=xDec;
```

# Up-sampling

$2\uparrow$

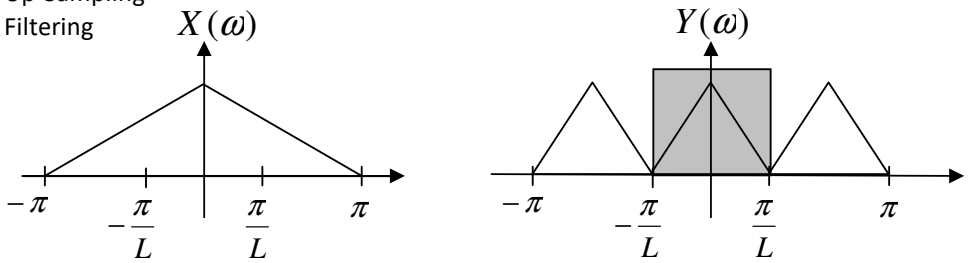
$$X_{\uparrow}(\omega)=\sum_{n}x_{\uparrow}(n)e^{-j\omega n}=\sum_{m}\underbrace{x_{\uparrow}(mL)}_{x(m)}e^{-j\omega mL}=\sum_{n}x(n)e^{-j\omega nL}=X(\omega L)$$

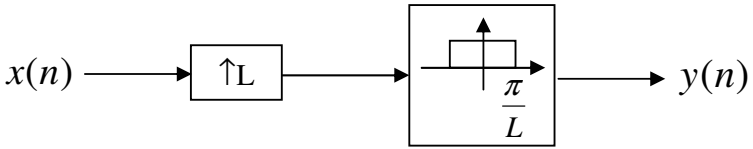


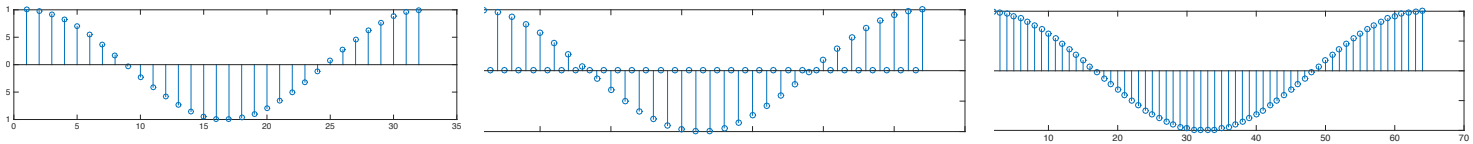
# Interpolation

2↑

Up-Sampling
Filtering



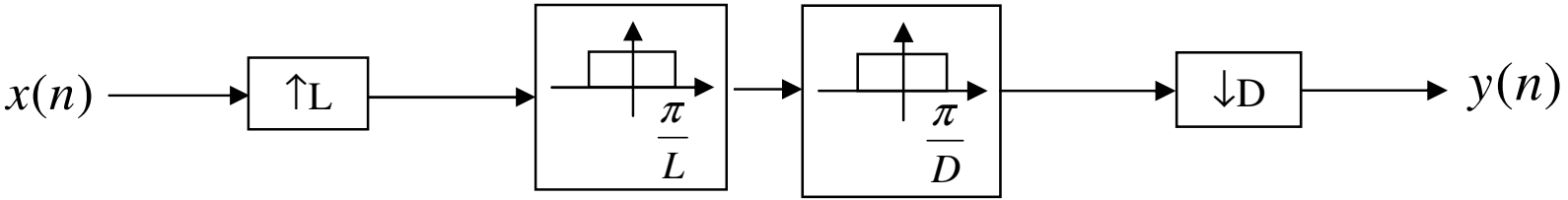



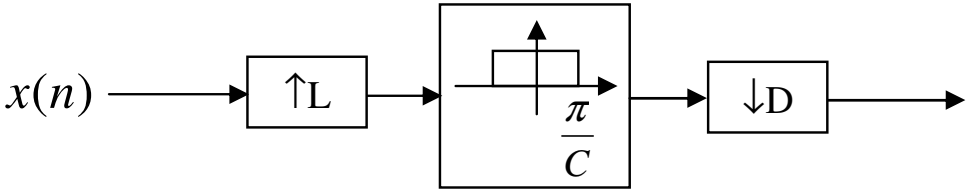



# Sampling rate changing by a rational non integer factor

$R\uparrow$

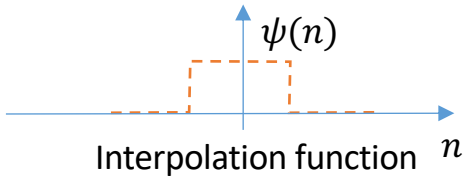
Sampling rate increasing by a factor L followed by a decreasing by a factor D, that is, R = L/D.

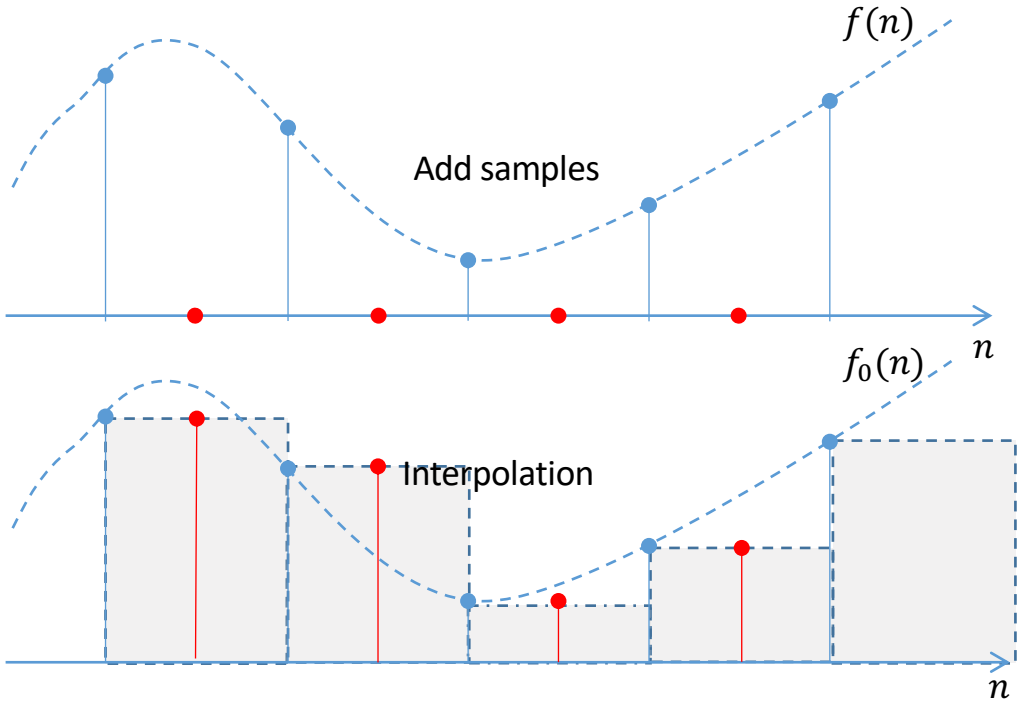


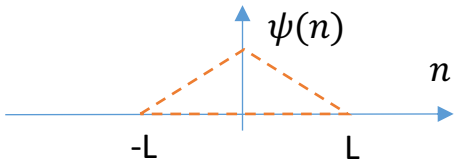

Where $C = \max\{ D, \ L \}$

# Zero order interpolation



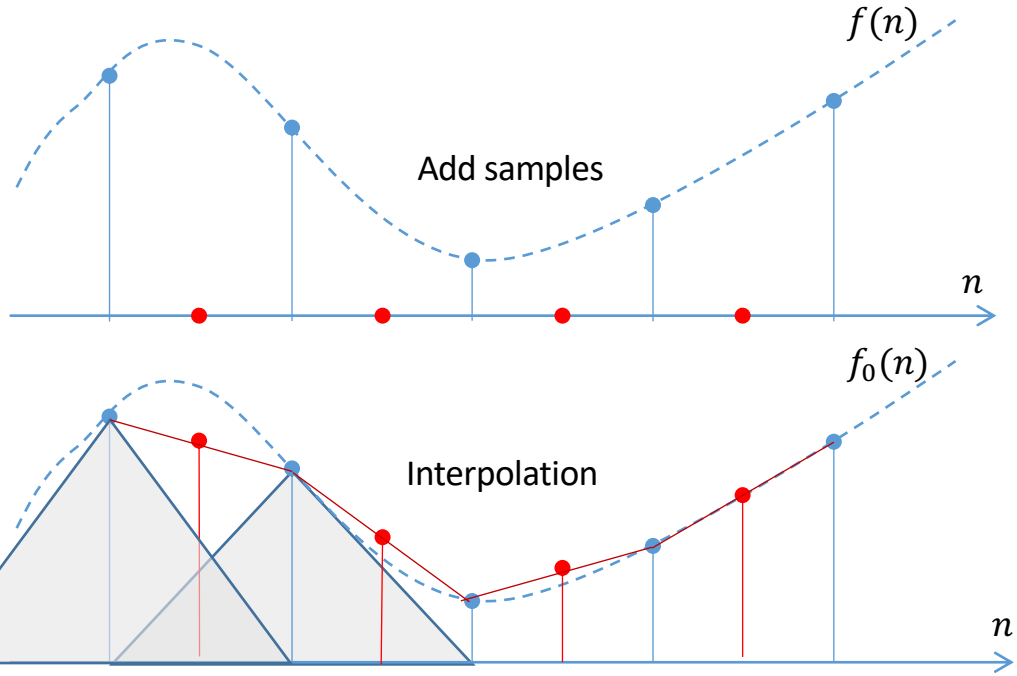



# First order interpolation




Interpolation function

$$\psi(n) = \begin{cases} 1 + n/L & -L \le n < 0 \\ 1 - n/L & 0 \le n \le L \\ 0 & otherwise \end{cases}$$



# Image resize

## imresize

Resize image

### Syntax

```
B = imresize(A,scale)
B = imresize(A,[numrows numcols])
[Y,newmap] = imresize(X,map,__)
__ = imresize(__,method)
__ = imresize(__,Name,Value)
```

**`method` — Interpolation method**

`'bicubic'` (default) | character vector | two-element cell array

Interpolation method, specified as a character vector or two-element cell array.

When `method` is a character vector, it identifies a particular method or named interpolation kernel, listed in the following table.

| Method | Description |
|---|---|
| `'nearest'` | Nearest-neighbor interpolation; the output pixel is assigned the value of the pixel that the point falls within. No other pixels are considered. |
| `'bilinear'` | Bilinear interpolation; the output pixel value is a weighted average of pixels in the nearest 2-by-2 neighborhood |
| `'bicubic'` | Bicubic interpolation; the output pixel value is a weighted average of pixels in the nearest 4-by-4 neighborhood<br>**Note**<br>Bicubic interpolation can produce pixel values outside the original range. |
| **Interpolation Kernel** | **Description** |
| `'box'` | Box-shaped kernel |
| `'triangle'` | Triangular kernel (equivalent to `'bilinear'`) |
| `'cubic'` | Cubic kernel (equivalent to `'bicubic'`) |
| `'lanczos2'` | Lanczos-2 kernel |
| `'lanczos3'` | Lanczos-3 kernel |

TÉCNICO LISBOA