

À la fréquence de Fourier ⇌

1.1 La Transformée de Fourier ⇌

À quoi ça sert ? ⇌

Un monde parallèle où les choses deviennent plus simples (ou plus compliquées)

- **1.2.1 Traitement du signal** Séparation des contributions de chaque fréquence
- **1.2.2 Avantage computationnel** L'opération de convolution devient une multiplication classique

1.1.1 Illustration : séparation des fréquences ⇌

Audio ⇌

play la

play ré

play la + ré

Signal temporel ⇌

Un son est un signal continu mais on va l'échantillonner. C'est à dire qu'on va prendre un nombre fini de valeur par seconde à des distance égale dans le temps. Dans cet exemple, on prend une valeur toute les 1024^e de seconde.

Fréquence d'échantillonnage =  1024

Ça correspond à une distance en seconde entre deux échantillons de:

```
Δt = 0.0009765625
```

```
1 Δt = 1 / échantillonnage
```

nombre_échantillons =  1024

```
temps_total = 1.0
```

En prenant 1024, le signal dure 1.0 secondes. Comme ces nombres sont équidistants (distance de Δt secondes entre eux), on peut représenter cette suite de façon compact comme suit:

```
temps = 0.0009765625:0.0009765625:1.0
```

```
1 temps = range(Δt, stop=temps_total, length=nombre_échantillons)
```

Le *la* est une note de musique de fréquence 440 Hz

```
la =
```

```
► [-0.903989+0.427555im, 0.634393-0.77301im, -0.24298+0.970031im, -0.19509-0.980785im, 0.5]
```

```
1 la = cispi.(2*440*temps)
```

La note de musique *ré* a une fréquence de 293.7 Hz

```
ré =
```

```
► [-0.229267+0.973364im, -0.894874-0.44632im, 0.639596-0.768711im, 0.601597+0.7988im, -0.9]
```

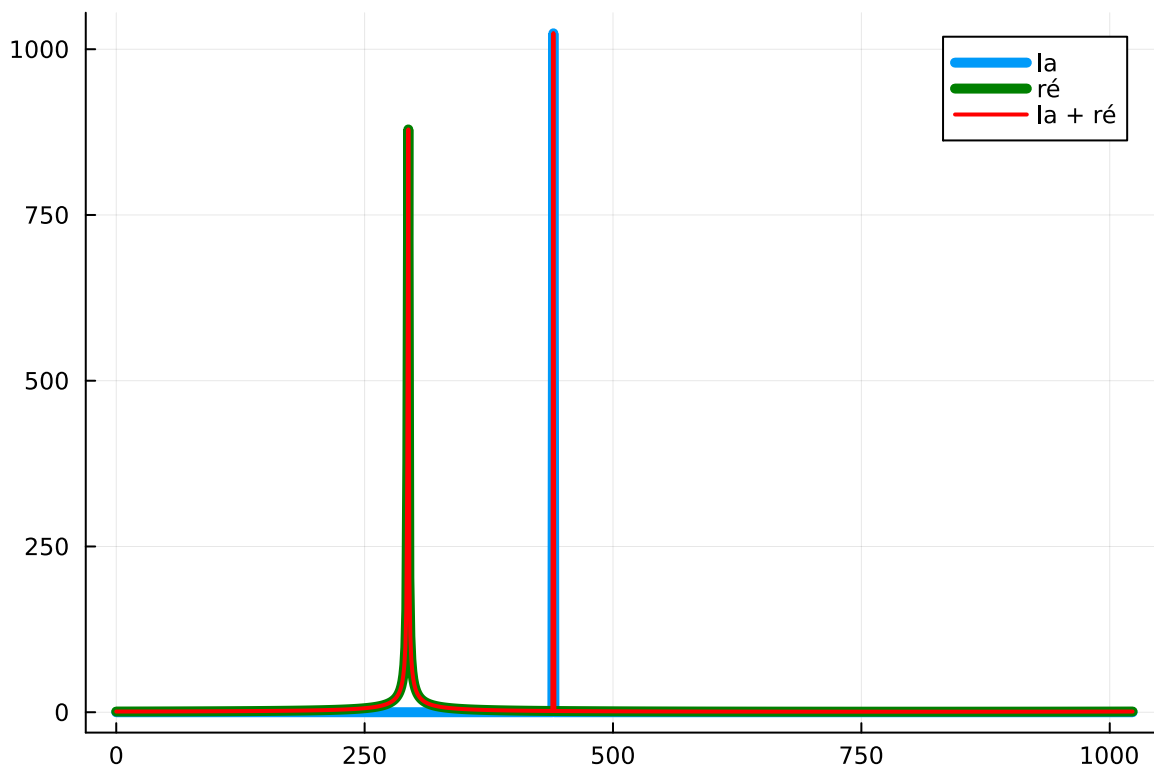
```
1 ré = cispi.(2*293.7*temps)
```

temps_zoom =  0.032447076612903226



Signal fréquentiel ⇔

abs ▼



1.1.2 Calcul rapide de convolution ⇔

La convolution continue:

$$(f * g)(x) = \int_{-\infty}^{\infty} f(s)g(x-s)ds$$

La convolution discrète:

$$(f * g)_n = \sum_{k=-\infty}^{\infty} f_k g_{n-k}$$

Propriétés algébriques:

commutatif	$f * g = g * f$
associatif	$(f * g) * h = f * (g * h)$
distributif	$f * (g + h) = f * g + f * h$
conjugué	$\overline{f * g} = \overline{f} * \overline{g}$

La cross-corrélation:

$$(f \star g)(x) = \int_{-\infty}^{\infty} f(s)g(x+s)ds \quad (f \star g)_n = \sum_{-\infty}^{\infty} f_k g_{n+k}ds$$

Convolution theorem: Notons la transformée de Fourier de f par $\mathcal{F}(f)$. Par la transformée de Fourier, la convolution devient un produit classique:

$$f * g = \mathcal{F}^{-1}(\mathcal{F}(f) \cdot \mathcal{F}(g))$$

► Quel est la complexité de la convolution vs produit classique s'ils sont discrets de longueur n ?

1.1.2.1 Illustration: convolution d'images ⇌

mandrill =



```
1 mandrill = testimage("mandrill")
```

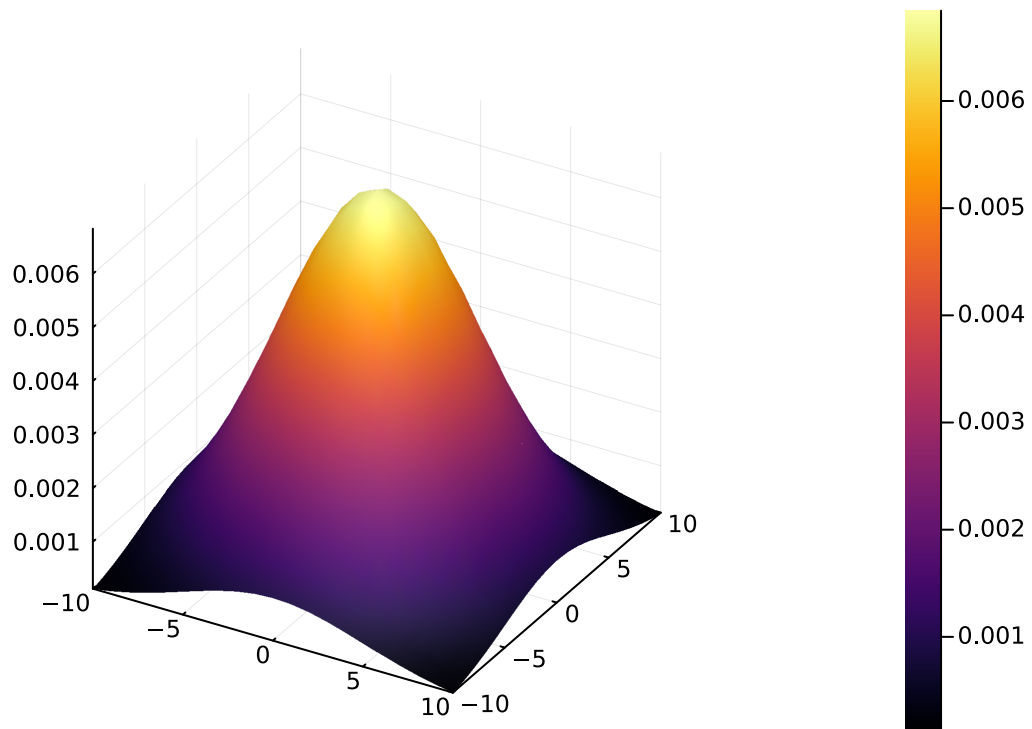


Downloading artifact: images



Gaussian Kernel ⇌

La fonction `gaussian(d)` donne une matrix de taille $(2d + 1) \times (2d + 1)$ contenant la valeur d'une Gaussienne.



```
1 surface(-2d:2d, -2d:2d, kernel.gaussian(d))
```

d = 5

La fonction `imfilter` calcule la cross-corrélation.



```
1 imfilter(mandrill, Kernel.gaussian(d))
```

La convolution est obtenue avec `reflect`.



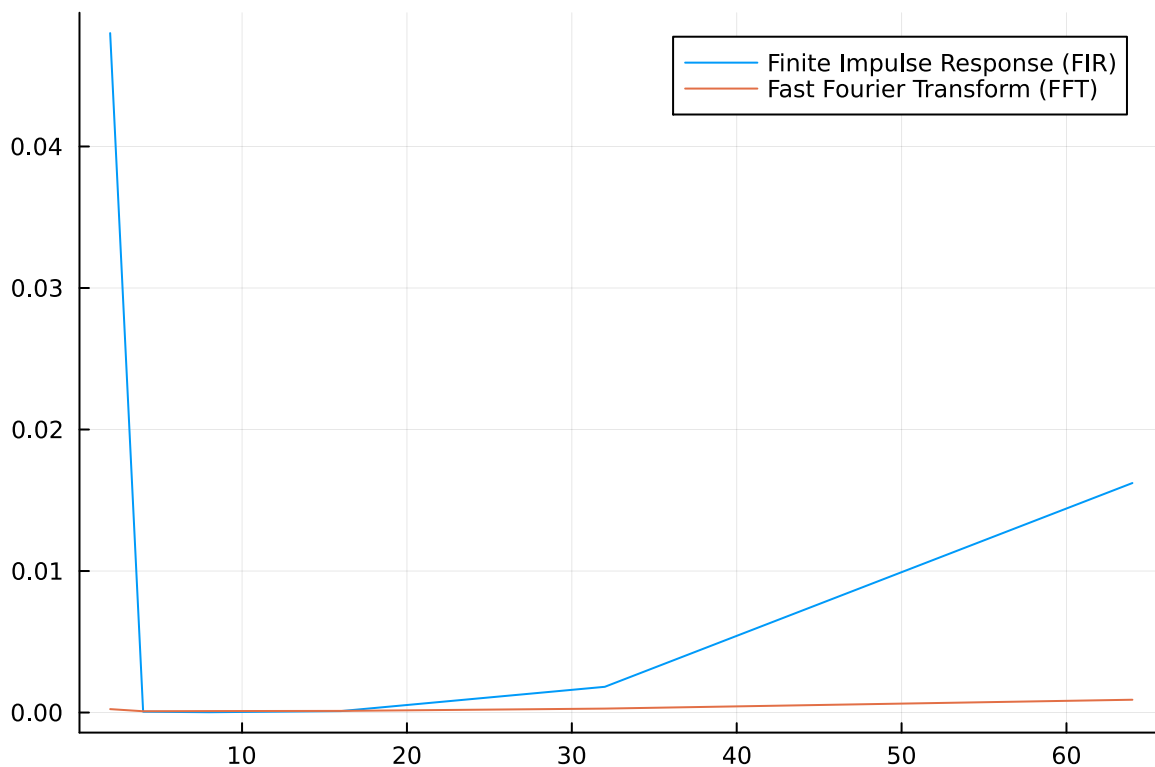
```
1 imfilter(mandrill, reflect(Kernel.gaussian(d)))
```

► Pourquoi obtient-on le même résultat avec la convolution et la cross-corrélation ?

Performance ⇔

imfilter_time (generic function with 1 method)

```
1 function imfilter_time(n, alg)
2     A = rand(n, n)
3     B = rand(n, n)
4     return @elapsed imfilter(A, B, alg)
5 end
```



```

1 let
2   n = 2 .^ (1:6)
3   fir_time(n) = imfilter_time(n, Algorithm.FIR())
4   fft_time(n) = imfilter_time(n, Algorithm.FFT())
5   plot(n, fir_time ◦ Int, label = "Finite Impulse Response (FIR)")
6   plot!(n, fft_time ◦ Int, label = "Fast Fourier Transform (FFT)")
7 end

```

```

3x3 OffsetArray{::Matrix{Int64}, -1:1, -1:1} with eltype Int64 with indices -1:1×-1:1:
 0  1  0
 1 -4  1
 0  1  0

```

```

1 convert(AbstractArray, Kernel.Laplacian())

```

Autre kernels ↗



```
1 imfilter(mandrill, Kernel.Laplacian())
```

Différents kernels permettent d'analyser des aspects différents d'une image. Le kernel à utilisé peut aussi être **appris**, c'est la base des **Convolutional Neural Networks** (CNNs) !

1.1.2.2 Illustration : Le produit de polynômes ↩

```
deg = 4
```

```
1 deg = 4
```

```
p = 0.232635935010149 + 0.6095209235337684·x + 0.4261674149695569·x2 + 0.48166465675489234·x3
```

```
1 p = Polynomial(rand(deg))
```

```
q = 0.5515524604124381 + 0.7123983679987089·x + 0.731546288271667·x2 + 0.8692015741050767·x3
```

```
1 q = Polynomial(rand(deg))
```

$0.12831092233519573 + 0.5019122254869954 \cdot x + 0.8394593522358733 \cdot x^2 + 1.217364587584689 \cdot x^3 + 1.184694852183686 \cdot x^4 + 0.7227853797645183 \cdot x^5 + 0.4186636778421339 \cdot x^6$

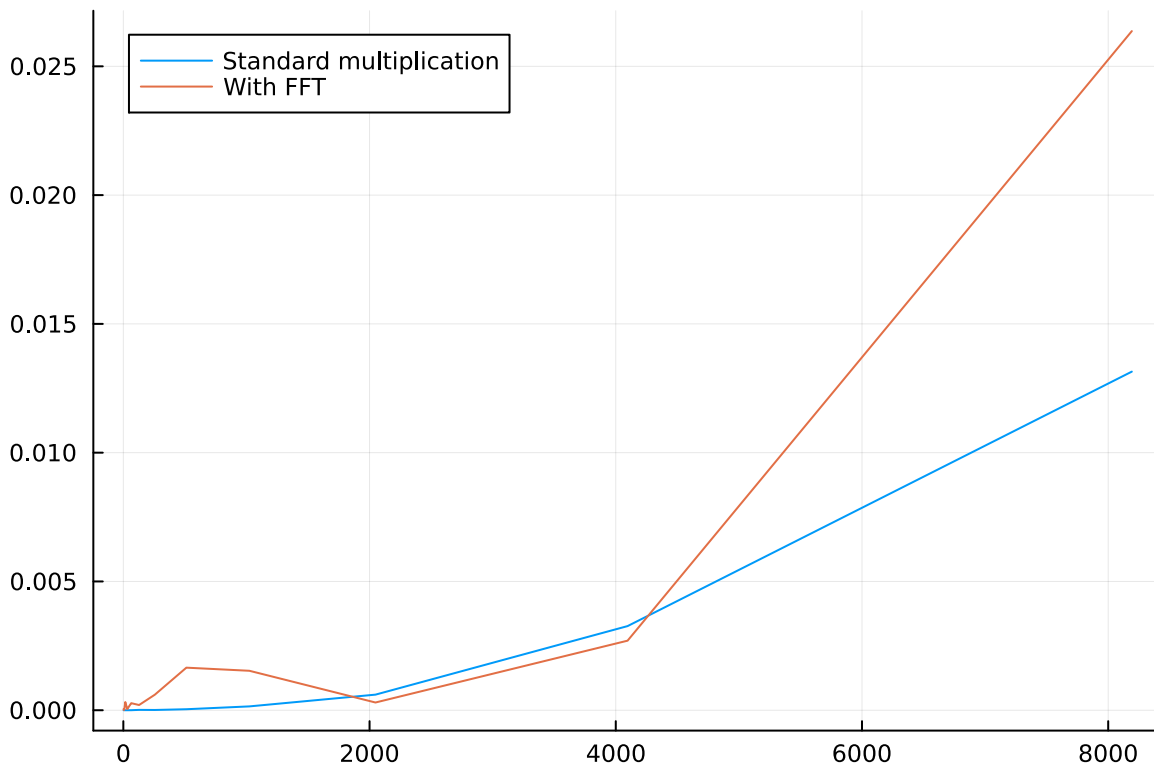
```
1 p * q
```

$0.1283109223351957 + 0.5019122254869953 \cdot x + 0.8394593522358733 \cdot x^2 + 1.2173645875846888 \cdot x^3 + 1.184694852183686 \cdot x^4 + 0.7227853797645183 \cdot x^5 + 0.418663677842134 \cdot x^6$

```
1 Polynomials.poly_multiplication_fft(p, q)
```

poly_time (generic function with 1 method)

```
1 function poly_time(n, with_fft::Bool)
2     p = Polynomial(rand(n))
3     q = Polynomial(rand(n))
4     @elapsed if with_fft
5         Polynomials.poly_multiplication_fft(p, q)
6     else
7         p * q
8     end
9 end
```



```

1 let
2   n = 2 .^ (1:13)
3   std_time(n) = poly_time(n, false)
4   fft_time(n) = poly_time(n, true)
5   plot(n, std_time ◦ Int, label = "Standard multiplication")
6   plot!(n, fft_time ◦ Int, label = "With FFT")
7 end

```

► Comment utiliser cela pour calculer le produit de grand nombres ?

1.1.2 Définition de la transformée de Fourier ⇄

La transformée de Fourier continue et son inverse:

$$F(\xi) = \int_{-\infty}^{\infty} f(t) e^{-i2\pi\xi t} dt$$

$$f(t) = \int_{-\infty}^{\infty} F(\xi) e^{i2\pi\xi t} d\xi$$

La transformée de Fourier discrète et son inverse:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi \frac{k}{N} n}$$

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{i2\pi \frac{k}{N} n}$$

Rajouter linearité : TODO

► **Que vaut la distance entre deux valeurs successive en temporel et en fréquentiel pour les signaux discrets ?**

TODO : même dimension en temporel et fréquentiel

► **Que vaut le signal en dehors de ses N points ?**

1.1.3 Le replis spectral ⇔

On vient de voir que le signal est extrapolé périodiquement par la transformée de fourier discrète. Si on prend un signal de 0 à $t_{\max}$ et qu'on divise la fréquence d'échantillonnage de sa transformée de fourier par 2. Ça signifie que le signal sera 2 fois moins long.

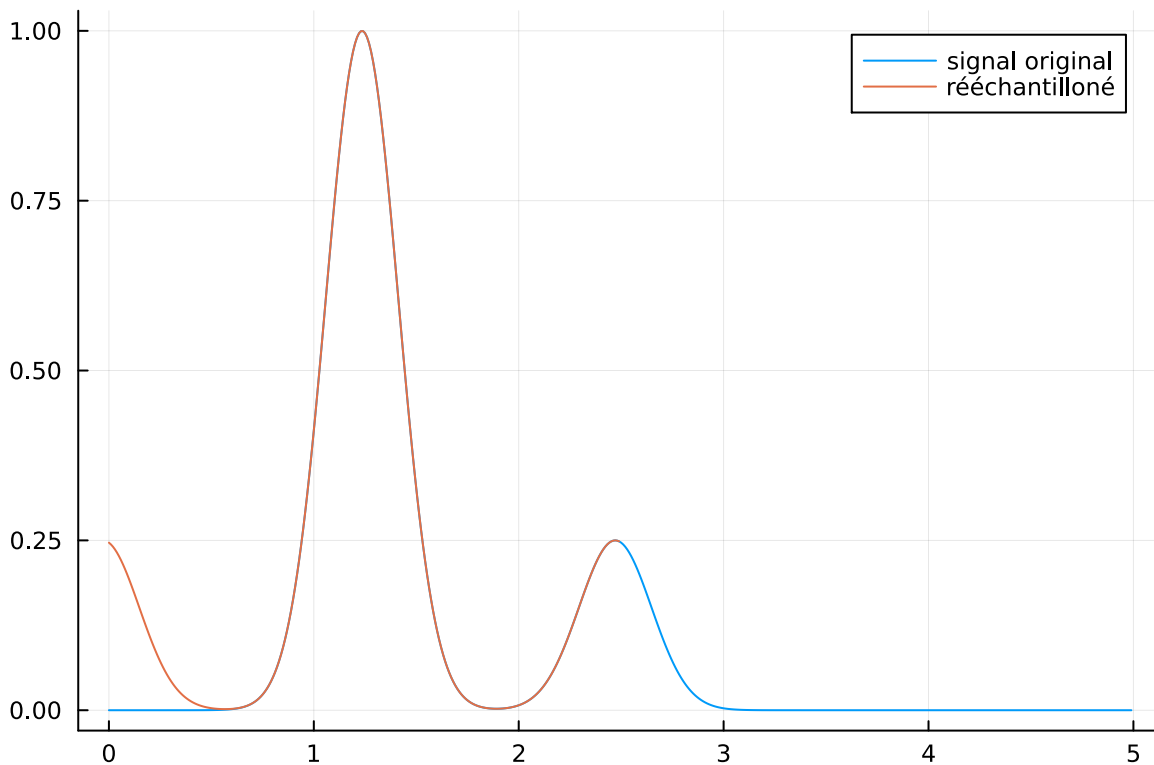
► **Que va-t-il advenir de la deuxième moitié du signal ?**

Illustrons cela avec la fonction $\exp(-(t-\text{centre})^2)$ qui est initialement échantillonnée avec 512 échantillons de 0 à $t_{\max}$ et qu'on ré-échantillonne ensuite à $512 / \text{ratio}$ en fréquentiel, ce qui divise $t_{\max}$ par ratio également.

$t_{\max}$ =  5.0

centre =  2.470967741935484

ratio =  2



1.2 Fast Fourier Transform ⇔

Attardons-nous à présent sur l'algorithme utilisé pour calculer la DFT (transformée de Fourier discrète). On a vu précédemment que cet algorithme a une complexité de $\Omega(n \log(n))$. C'est parmi le top 10 des meilleurs algorithmes du 20^e siècle ! Son Implémentation la plus rapide est dans la librairie FFTW : the Fastest Fourier Transform in the West ! Comment est-elle la plus rapide ? Voir [ici](#). L'algorithme est initialement découvert par Gauss en 1805 puis redécouvert par Cooley et Tukey en 1965.

La DFT est en fait une évaluation d'un polynôme aux différentes racines N-èmes de l'unité:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi \frac{k}{N} n}$$

$$X_k = \sum_{n=0}^{N-1} x_n z_{k,N}^n \quad z_{k,N} = e^{-i2\pi \frac{k}{N}}$$

La DFT peut aussi être vue comme un produit matriciel:

$$X = \begin{bmatrix} 1 & 1 & \dots & 1 & 1 \\ 1 & z_N & \dots & z_N^{N-2} & z_N^{N-1} \\ 1 & z_N^2 & \dots & z_N^{2(N-2)} & z_N^{2(N-1)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & z_N^{(N-1)2} & \dots & z_N^{(N-1)(N-2)} & z_N^{(N-1)(N-1)} \end{bmatrix} x$$

Un produit matriciel a complexité $\Omega(n^2)$. Ça ne nous donne pas une complexité de $\Omega(n \log n)$, il va falloir utiliser la structure assez spéciale de cette matrice...

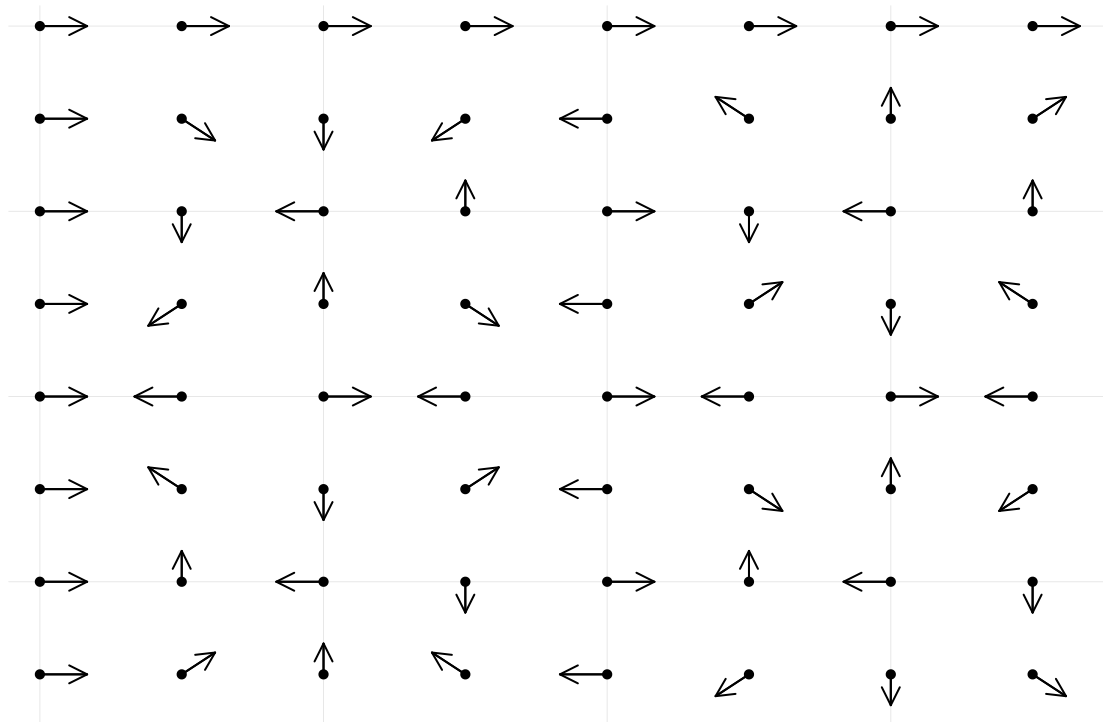
On a acquis une intuition géométrique sur les racines z_N . On va s'en servir pour visualiser cette matrice.

```
8x8 Matrix{ComplexF64}:
1.0+0.0im      1.0+0.0im      1.0+0.0im ... 1.0+0.0im      1.0+0.0im
1.0+0.0im      0.707107-0.707107im  0.0-1.0im      0.0+1.0im      0.707107+0.707107im
1.0+0.0im      0.0-1.0im      -1.0-0.0im      -1.0-0.0im      0.0+1.0im
1.0+0.0im     -0.707107-0.707107im  0.0+1.0im      0.0-1.0im     -0.707107+0.707107im
1.0+0.0im      -1.0-0.0im      1.0-0.0im      1.0-0.0im      -1.0-0.0im
1.0+0.0im     -0.707107+0.707107im  0.0-1.0im ... 0.0+1.0im     -0.707107-0.707107im
1.0+0.0im      0.0+1.0im      -1.0-0.0im      -1.0-0.0im      0.0-1.0im
1.0+0.0im      0.707107+0.707107im  0.0+1.0im      0.0-1.0im      0.707107-0.707107im
```

1 [F\(8\)](#)

► **Qu'observe-t-on dans la matrice ci-dessous ?**

Besoin d'un indice ? ☐



```
1 arrows(8, colored_arrows)
```

On peut le vérifier numériquement également:

```
partie_mauve = 4x4 Matrix{ComplexF64}:
 1.0+0.0im  1.0+0.0im  1.0+0.0im  1.0+0.0im
 1.0+0.0im  0.0-1.0im -1.0-0.0im  0.0+1.0im
 1.0+0.0im -1.0-0.0im  1.0-0.0im -1.0-0.0im
 1.0+0.0im  0.0+1.0im -1.0-0.0im  0.0-1.0im
```

```
1 partie_mauve = F(8)[1:4, 1:2:8]
```

```
partie_bleue = 4x4 Matrix{ComplexF64}:
 1.0+0.0im  1.0-0.0im  1.0-0.0im  1.0-0.0im
 1.0+0.0im  0.0-1.0im -1.0-0.0im  0.0+1.0im
 1.0+0.0im -1.0-0.0im  1.0-0.0im -1.0-0.0im
 1.0+0.0im  0.0+1.0im -1.0-0.0im  0.0-1.0im
```

```
1 partie_bleue = F(8)[5:8, 1:2:8]
```

```
true
```

```
1 partie_mauve == partie_bleue
```

```
partie_verte = 4x4 Matrix{ComplexF64}:
      1.0+0.0im      1.0+0.0im      ...      1.0+0.0im
      0.707107-0.707107im -0.707107-0.707107im  0.707107+0.707107im
      0.0-1.0im      0.0+1.0im      0.0+1.0im
      -0.707107-0.707107im  0.707107-0.707107im -0.707107+0.707107im
```

```
1 partie_verte = F(8)[1:4, 2:2:8]
```

```
partie_rouge = 4x4 Matrix{ComplexF64}:
      -1.0-0.0im      -1.0-0.0im      ...      -1.0-0.0im
      -0.707107+0.707107im  0.707107+0.707107im -0.707107-0.707107im
      0.0+1.0im      0.0-1.0im      0.0-1.0im
      0.707107+0.707107im -0.707107+0.707107im  0.707107-0.707107im
```

```
1 partie_rouge = F(8)[5:8, 2:2:8]
```

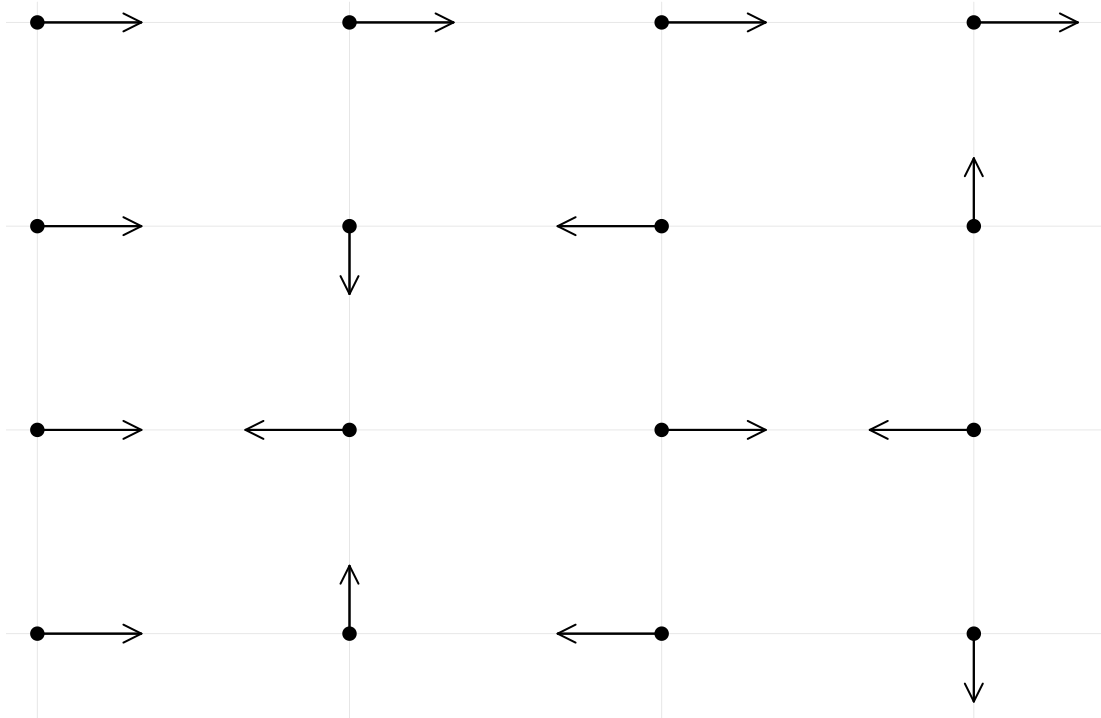
```
true
```

```
1 partie_verte == -partie_rouge
```

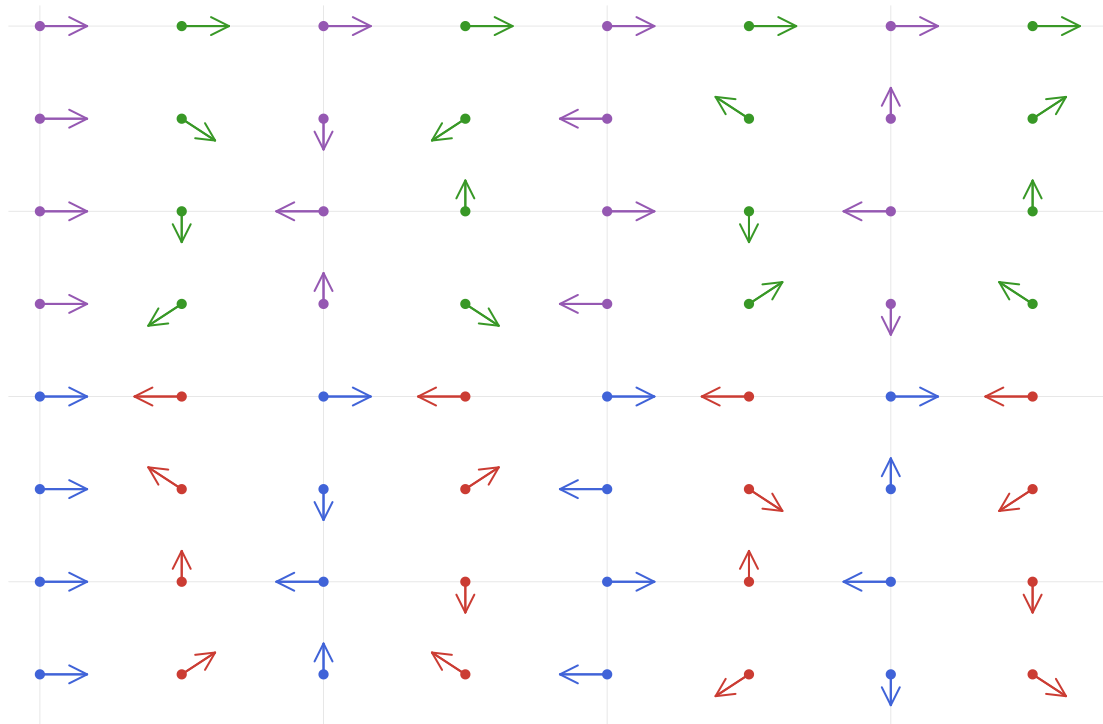
```
true
```

```
1 isapprox(0.0, 1e-300, atol = 1e-8)
```

► Qu'observe-t-on en comparant cela à la matrice 2 fois plus petite ?



```
1 arrows(4, false)
```



Vérifions le à nouveau numériquement:

```
4x4 Matrix{ComplexF64}:
 1.0+0.0im  1.0-0.0im  1.0-0.0im  1.0-0.0im
 1.0+0.0im  0.0-1.0im -1.0-0.0im  0.0+1.0im
 1.0+0.0im -1.0-0.0im  1.0-0.0im -1.0-0.0im
 1.0+0.0im  0.0+1.0im -1.0-0.0im  0.0-1.0im
```

```
1 partie_bleue
```

```
4x4 Matrix{ComplexF64}:
 1.0+0.0im  1.0+0.0im  1.0+0.0im  1.0+0.0im
 1.0+0.0im  0.0-1.0im -1.0-0.0im  0.0+1.0im
 1.0+0.0im -1.0-0.0im  1.0-0.0im -1.0-0.0im
 1.0+0.0im  0.0+1.0im -1.0-0.0im  0.0-1.0im
```

```
1 F(4)
```

```
true
```

```
1 partie_bleue == F(4)
```

```
4x4 Matrix{ComplexF64}:
 1.0+0.0im          1.0+0.0im  ...  1.0+0.0im
 0.707107-0.707107im -0.707107-0.707107im  0.707107+0.707107im
 0.0-1.0im          0.0+1.0im          0.0+1.0im
 -0.707107-0.707107im  0.707107-0.707107im -0.707107+0.707107im
```

```
1 partie_verte
```

```
0.7071067811865476 - 0.7071067811865475im
```

```
1 cispi(-2/8)
```

```
►[1.0+0.0im, 0.707107-0.707107im, 2.22045e-16-1.0im, -0.707107-0.707107im]
```

```
1 cispi(-2/8) .^ collect(0:3)
```

```
4×4 Matrix{ComplexF64}:
```

```
      1.0+0.0im      1.0+0.0im      ...      1.0+0.0im
      0.707107-0.707107im  -0.707107-0.707107im  0.707107+0.707107im
      2.22045e-16-1.0im  -2.22045e-16+1.0im  -2.22045e-16+1.0im
      -0.707107-0.707107im  0.707107-0.707107im  -0.707107+0.707107im
```

```
1 cispi(-2/8) .^ (0:3) .* F(4)
```

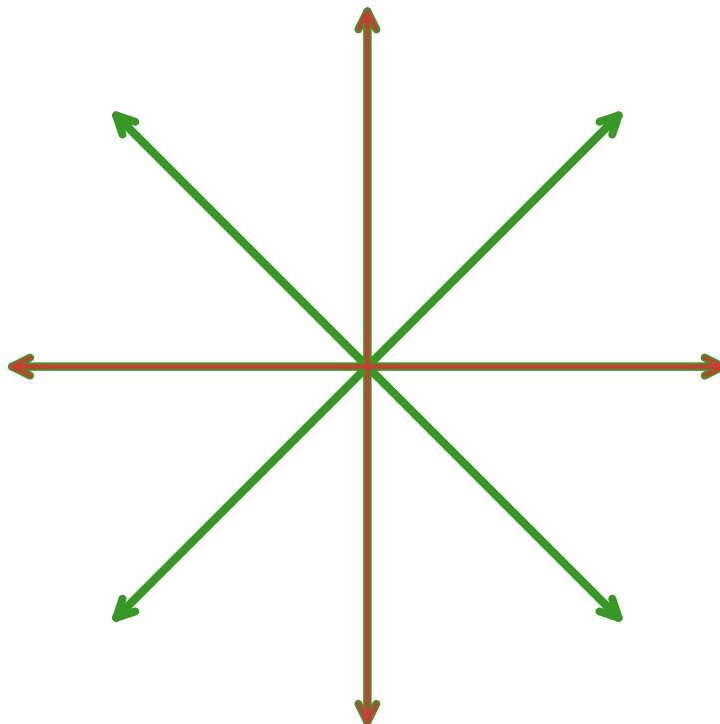
```
4×4 BitMatrix:
```

```
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
```

```
1 isapprox.(real.(partie_verte), real.(cispi(-2/8) .^ (0:3) .* F(4)), atol=1e-8)
```

En se rappelant que l'intuition géométrique des N racines de l'unité, on comprend pourquoi les $N/2$ racines sont un sous-ensembles des N racines.

max_N = 8



Utilitaires ⇌

1 using Plots

Precompiling Plots...

```
2053.9 ms ✓ ColorVectorSpace
3012.0 ms ✓ ColorSchemes
10273.2 ms ✓ PlotUtils
2938.0 ms ✓ PlotThemes
4296.4 ms ✓ RecipesPipeline
62581.0 ms ✓ Plots
2769.9 ms ✓ Plots → UnitfulExt
7 dependencies successfully precompiled in 85 seconds. 178 already precompiled.
```

1 using PlutoUI, FFTW, PortAudio, Images, TestImages, ImageFiltering

Precompiling PortAudio...

```
797.9 ms ✓ argp_standalone_jll
811.5 ms ✓ fts_jll
992.8 ms ✓ GSL_jll
1108.4 ms ✓ ICU_jll
1157.4 ms ✓ Ncurses_jll
922.4 ms ✓ Libtool_jll
866.5 ms ✓ libcap_jll
918.2 ms ✓ SoXResampler_jll
1270.5 ms ✓ FLAC_jll
1306.2 ms ✓ BerkeleyDB_jll
987.7 ms ✓ obstack_jll
873.7 ms ✓ alsa_jll
942.1 ms ✓ Readline_jll
1195.4 ms ✓ Libical_jll
934.0 ms ✓ libportaudio_jll
2065.4 ms ✓ Polynomials → PolynomialsFFTWExt
1473.5 ms ✓ Elfutils_jll
1361.2 ms ✓ libsndfile_jll
991.6 ms ✓ Gdbm_jll
1177.3 ms ✓ BlueZ_jll
1152.8 ms ✓ GStreamer_jll
2011.4 ms ✓ SBC_jll
3172.9 ms ✓ DSP
1516.3 ms ✓ SampledSignals
4356.8 ms ✓ PulseAudio_jll
1573.9 ms ✓ alsa_plugins_jll
3732.0 ms ✓ PortAudio
27 dependencies successfully precompiled in 16 seconds. 122 already precompiled.
```

1 using Polynomials, LinearAlgebra

play_sound (generic function with 1 method)

```
1 function play_sound(sound, samplerate)
2     # For the static version, there is no sound so we just make the buttons
3     # do nothing
4     if PortAudio.Pa_GetDefaultOutputDevice() != PortAudio.LibPortAudio.paNoDevice
5         PortAudioStream(0, 2; samplerate) do stream
6             write(stream, real.(sound))
7         end
8     end
9 end
```

```
1 begin
2     struct Join
3         list
4         Join(a) = new(a)
5         Join(a, b, args...) = Join(tuple(a, b, args...))
6     end
7     function Base.show(io::IO, mime::MIME"text/html", d::Join)
8         for el in d.list
9             show(io, mime, el)
10        end
11    end
12 end
```

```
1 begin
2     struct HTMLTag
3         tag::String
4         parent
5     end
6     function Base.show(io::IO, mime::MIME"text/html", d::HTMLTag)
7         write(io, "<", d.tag, ">")
8         show(io, mime, d.parent)
9         write(io, "</", d.tag, ">")
10    end
11 end
```

f (generic function with 1 method)

```
1 f(i, j, n) = cispi(-2 * i * j / n)
```

F (generic function with 1 method)

```
1 function F(n)
2     A = [
3         f(k, m, n)
4         for k in 0:(n-1), m in 0:(n-1)
5     ]
6 end
```

```

8×8 Matrix{ComplexF64}:
 1.0+0.0im    1.0+0.0im    1.0+0.0im ...    1.0+0.0im    1.0+0.0im
 1.0+0.0im    0.707107-0.707107im  0.0-1.0im    0.0+1.0im    0.707107+0.707107im
 1.0+0.0im    0.0-1.0im    -1.0-0.0im    -1.0-0.0im    0.0+1.0im
 1.0+0.0im   -0.707107-0.707107im  0.0+1.0im    0.0-1.0im   -0.707107+0.707107im
 1.0+0.0im    -1.0-0.0im    1.0-0.0im    1.0-0.0im    -1.0-0.0im
 1.0+0.0im   -0.707107+0.707107im  0.0-1.0im ...    0.0+1.0im   -0.707107-0.707107im
 1.0+0.0im    0.0+1.0im    -1.0-0.0im    -1.0-0.0im    0.0-1.0im
 1.0+0.0im    0.707107+0.707107im  0.0+1.0im    0.0-1.0im    0.707107-0.707107im
1 F(8)

```

arrows (generic function with 2 methods)

```

1 function arrows(n, use_color = false)
2     plot(showaxis = false)
3     for i in 0:(n-1)
4         for j in 0:(n-1)
5             if use_color
6                 color = Colors.JULIA_LOGO_COLORS[1 + iseven(j) * 2 + (2i < n)]
7             else
8                 color = :black
9             end
10            scatter!([j], [-i], markersize = 8 / sqrt(n), markerstrokewidth = 0,
11                    legend = nothing; color)
12            c = f(i, j, n) / 3 # Divide by 3 so that arrow is not too long in plot
13            plot!([j, j + real(c)], [-i, -i + imag(c)], arrow = true, legend =
14                nothing; color)
15        end
16    end
17    plot!()
18 end

```

draw_roots! (generic function with 1 method)

```

1 function draw_roots!(N; kws...)
2     for i in 1:N
3         b, a = sincospi(2 * i / N)
4         plot!([0, a], [0, b]; arrow = true, label = nothing, kws...)
5     end
6 end

```

qa (generic function with 1 method)

```

1 function qa(question, answer)
2     return HTMLTag("details", Join(HTMLTag("summary", question), answer))
3 end

```