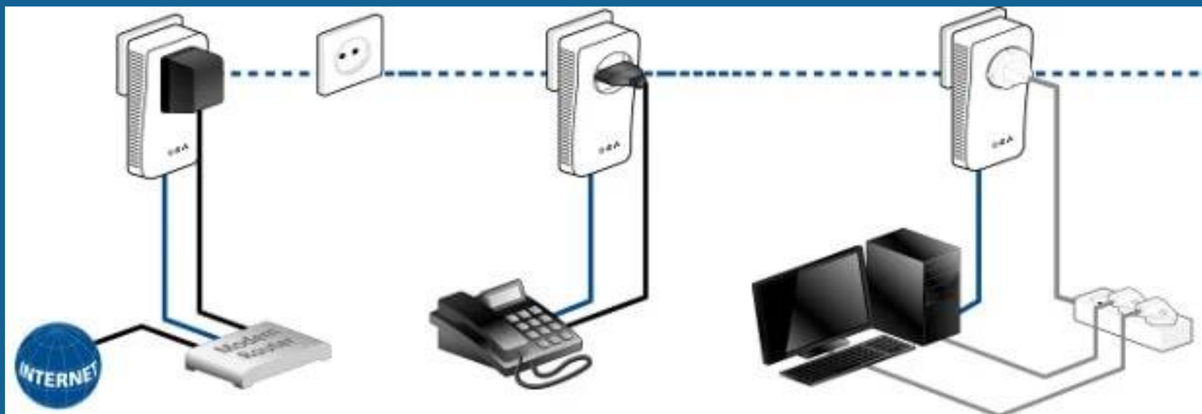
CARACTÉRISATION DU BRUIT DANS LA COMMUNICATION PAR COURANTS PORTEURS DE LIGNE (CPL)

Saïd TELLEZ

N° d'inscription : 17920

LA VILLE



Internet dans le milieu rural

Transition écologique:
- Le villes consomment beaucoup d'énergie
- Réseaux intelligents

PROBLÉMATIQUE

Difficulté : Faire passer du courant ET de l'information

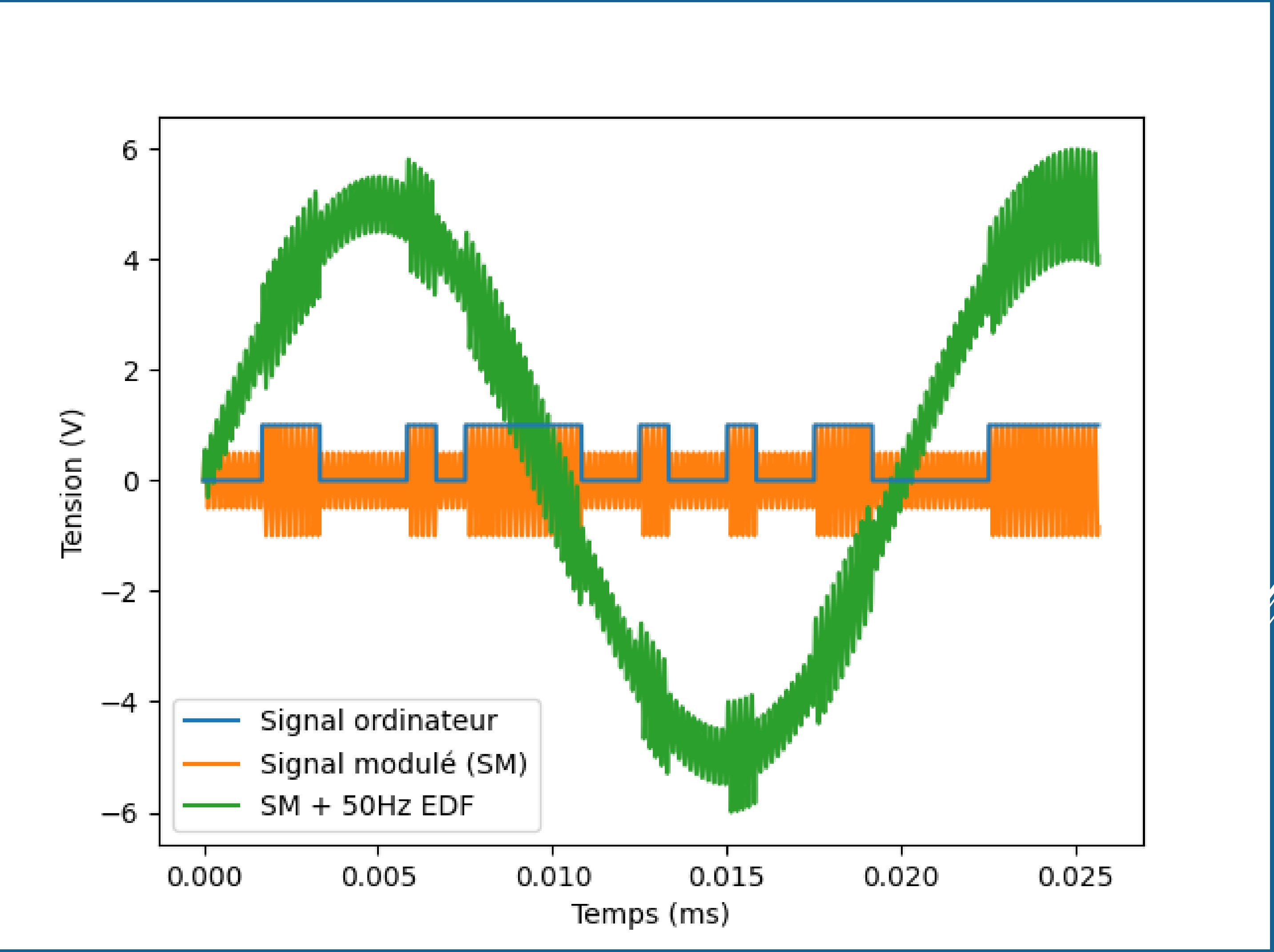
- ➢ Introduction de bruit

Quelles méthodes existe-t-il pour caractériser le bruit mesuré dans un système de communication par Courants Porteurs de Ligne ?

I. Modèle physique
   1. Principe
   2. Circuit
   3. Résultats
II. Méthode des différences finies
   1. Équations de Maxwell
   2. Discrétisation d'EDP
III. Solveur spectral
   1. Transformée de Fourier discrète
   2. Solveur pseudo-spectral
   3. Amélioration (Finger Trees)

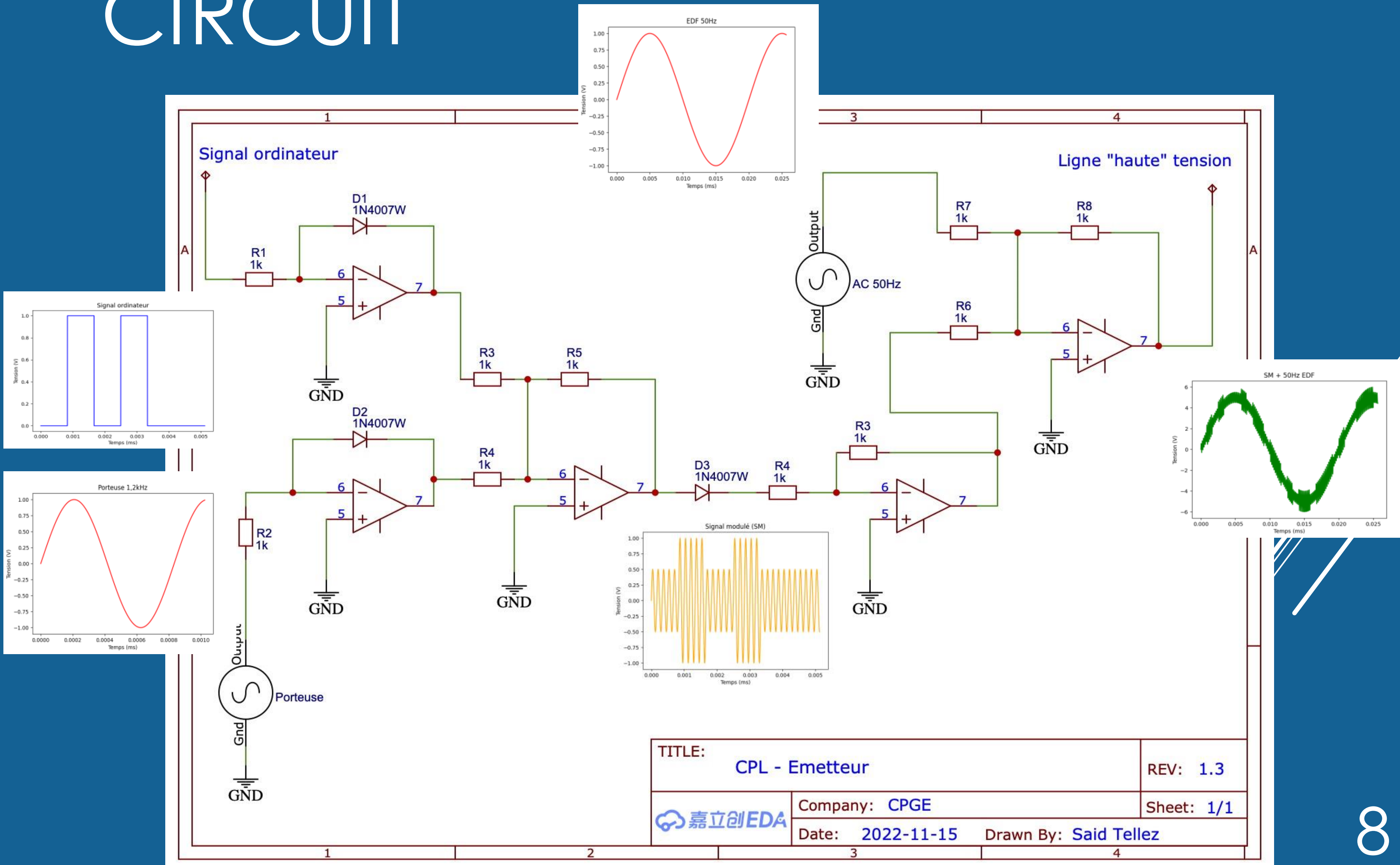
I. **Modèle physique**
   1. **Principe**
   2. **Circuit**
   3. **Résultats**
II. Méthode des différences finies
   1. Équations de Maxwell
   2. Discrétisation d'EDP
III. Solveur spectral
   1. Transformée de Fourier rapide
   2. Solveur pseudo-spectral
   3. Amélioration (Finger Trees)

PRINCIPE

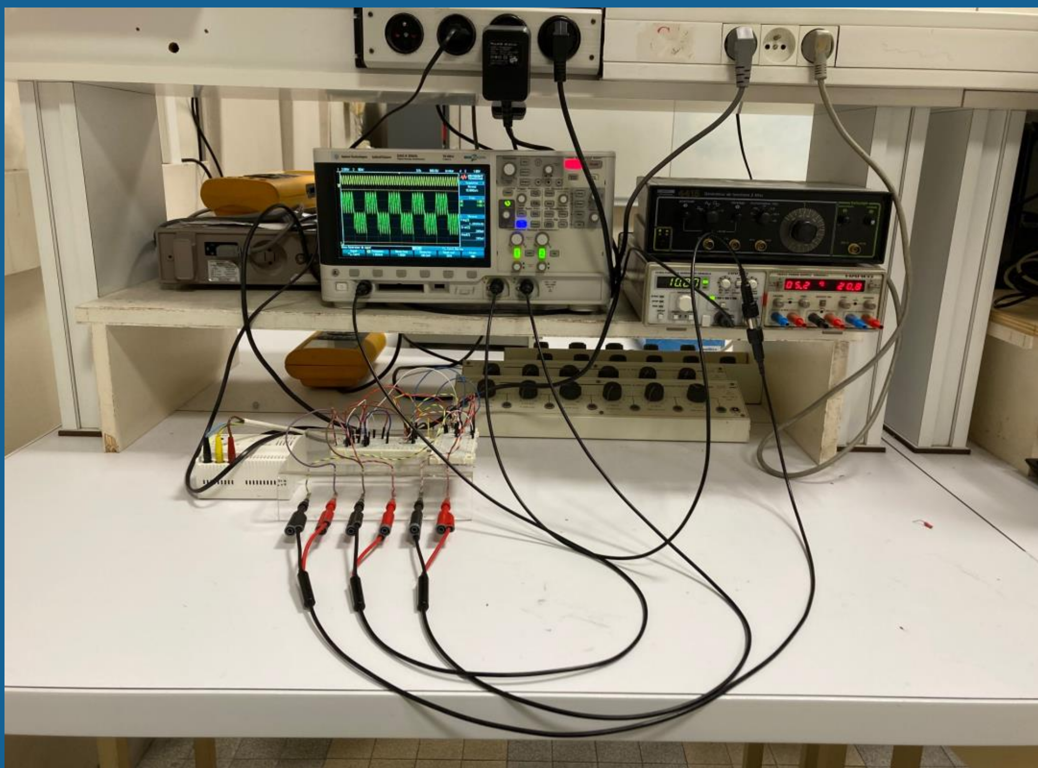

Tension (V)
Temps (ms)
6
4
2
0
−2
−4
−6
0.000
0.005
0.010
0.015
0.020
0.025
Signal ordinateur
Signal modulé (SM)
SM + 50Hz EDF

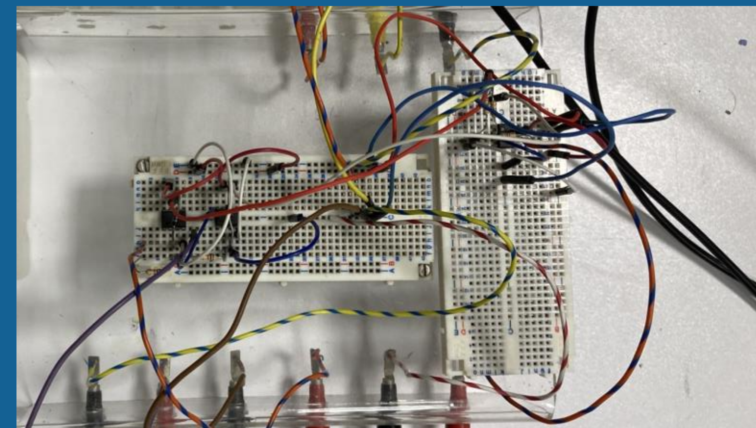
I. **Modèle physique**
   1. **Principe**
   2. **<u>Circuit</u>**
   3. **Résultats**
II. Méthode des différences finies
   1. Équations de Maxwell
   2. Discrétisation d'EDP
III. Solveur spectral
   1. Transformée de Fourier rapide
   2. Solveur pseudo-spectral
   3. Amélioration (Finger Trees)

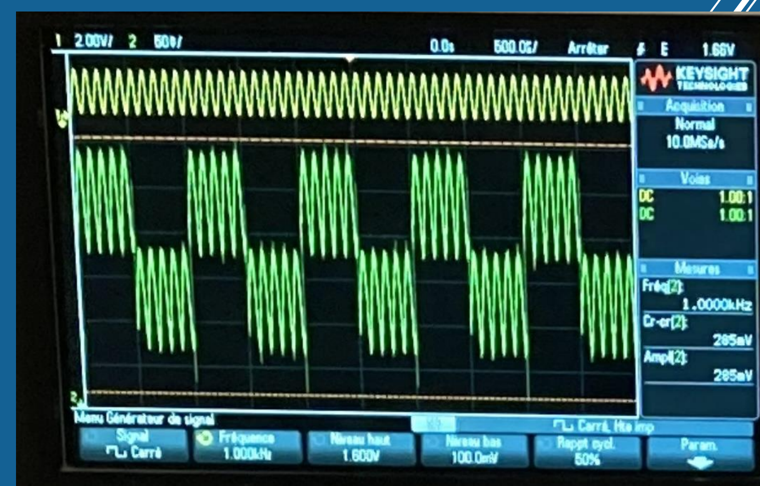
CIRCUIT


Signal ordinateur
Ligne "haute" tension
D1
1N4007W
D2
1N4007W
D3
1N4007W
R1
1k
R2
1k
R3
1k
R4
1k
R5
1k
R6
1k
R7
1k
R8
1k
GND
AC 50Hz
Output
Gnd
Porteuse
EDF 50Hz
Temps (ms)
Tension (V)
Signal ordinateur
Porteuse 1,2kHz
Signal modulé (SM)
SM + 50Hz EDF
TITLE:
CPL - Emetteur
REV: 1.3
Company: CPGE
Sheet: 1/1
Date: 2022-11-15
Drawn By: Said Tellez
嘉立创EDA

CIRCUIT

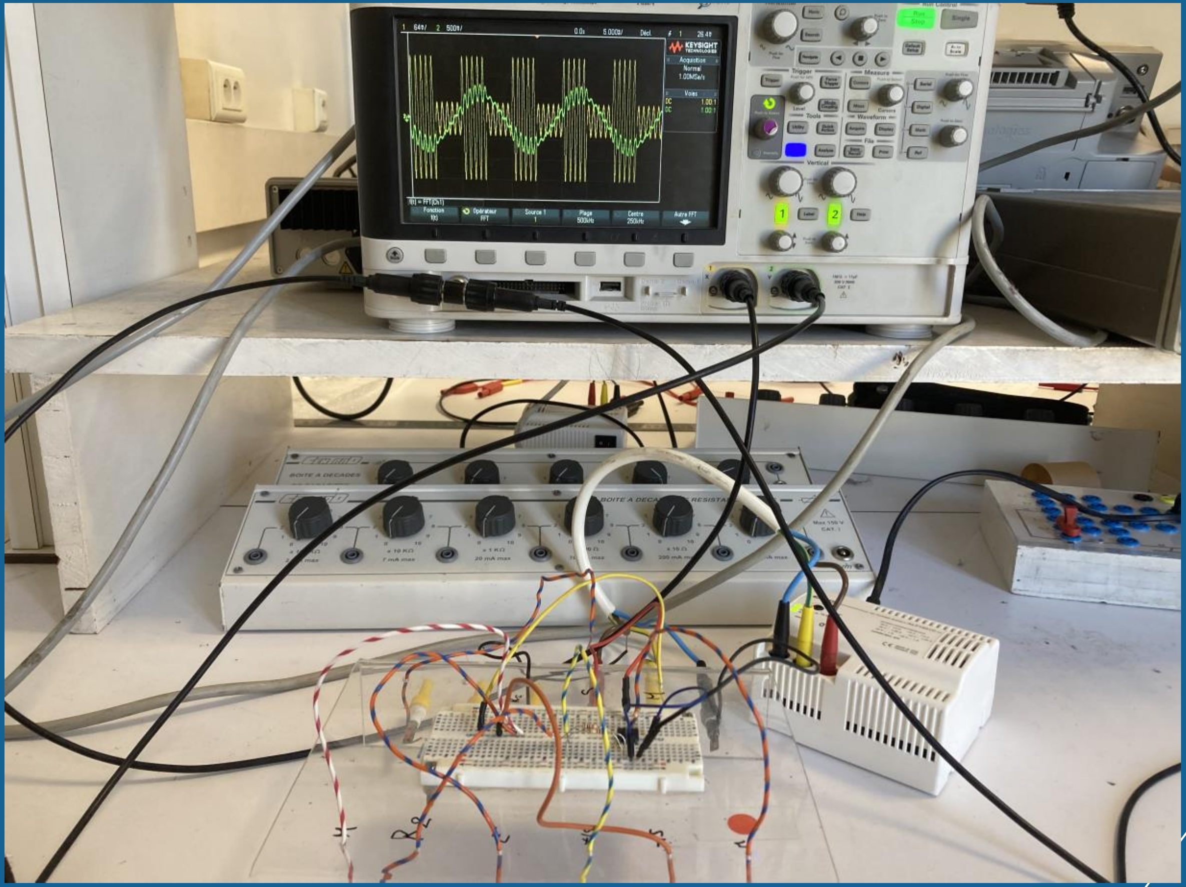





1 2.00V/ 2
0.0s
Arrêter
1.66V
KEYSIGHT
Acquisition
Normal
10.0MSa/s
Voies
DC 1.00:1
DC 1.00:1
Mesures
Fréq(2):
1.0000kHz
Cr-cr(2):
285mV
Ampl(2):
285mV
Menu Générateur de signal
Carré, Hte imp
Signal
Carré
Fréquence
1.000kHz
Niveau haut
1.600V
Niveau bas
100.0mV
Rappt cycl.
50%
Param.

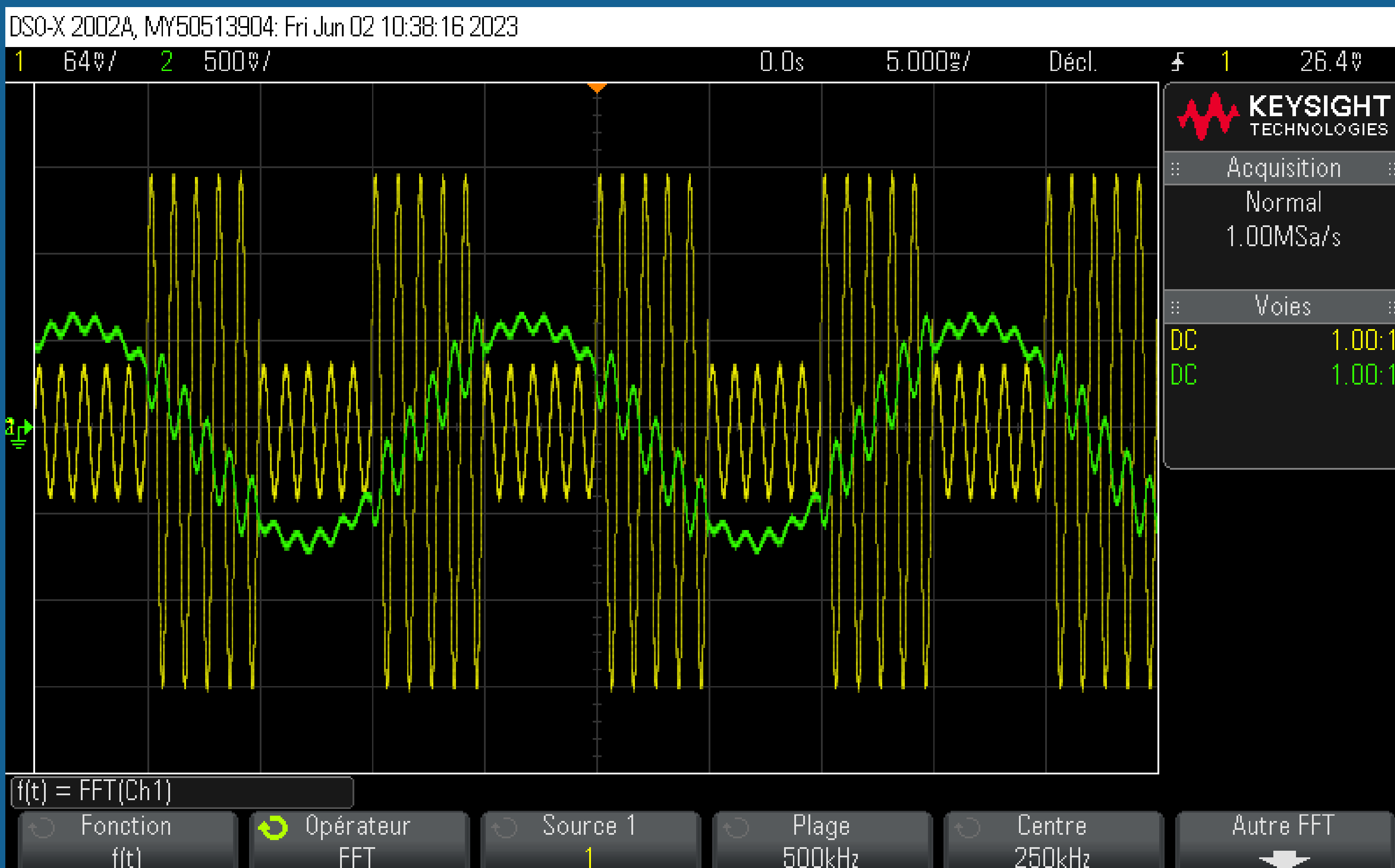
I. **Modèle physique**
   1. **Principe**
   2. **Circuit**
   3. **<u>Résultats</u>**
II. Méthode des différences finies
   1. Équations de Maxwell
   2. Discrétisation d'EDP
III. Solveur spectral
   1. Transformée de Fourier rapide
   2. Solveur pseudo-spectral
   3. Amélioration (Finger Trees)

OSCILLOSCOPE

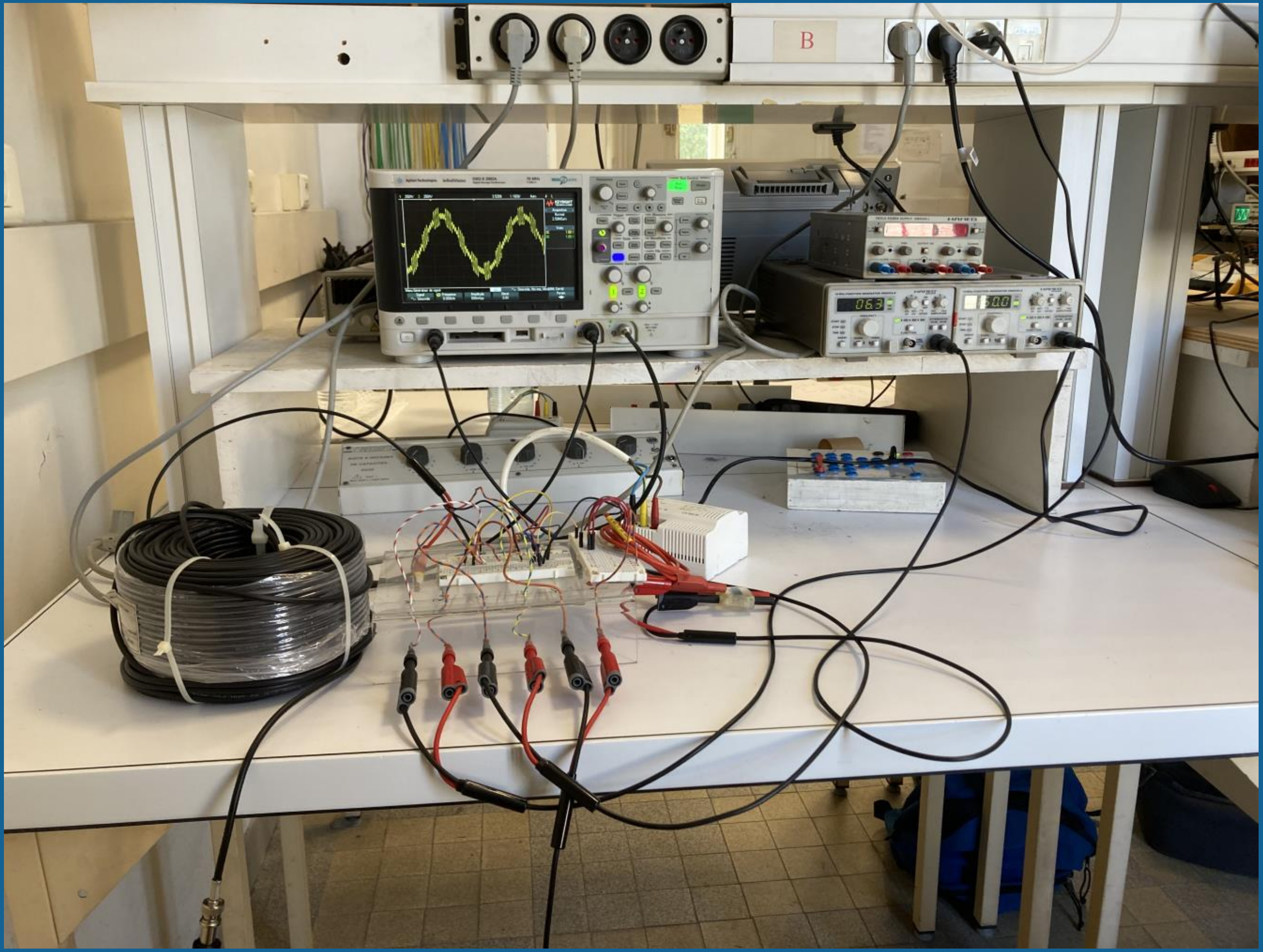


Modulation grâce à l'oscilloscope

OSCILLOSCOPE

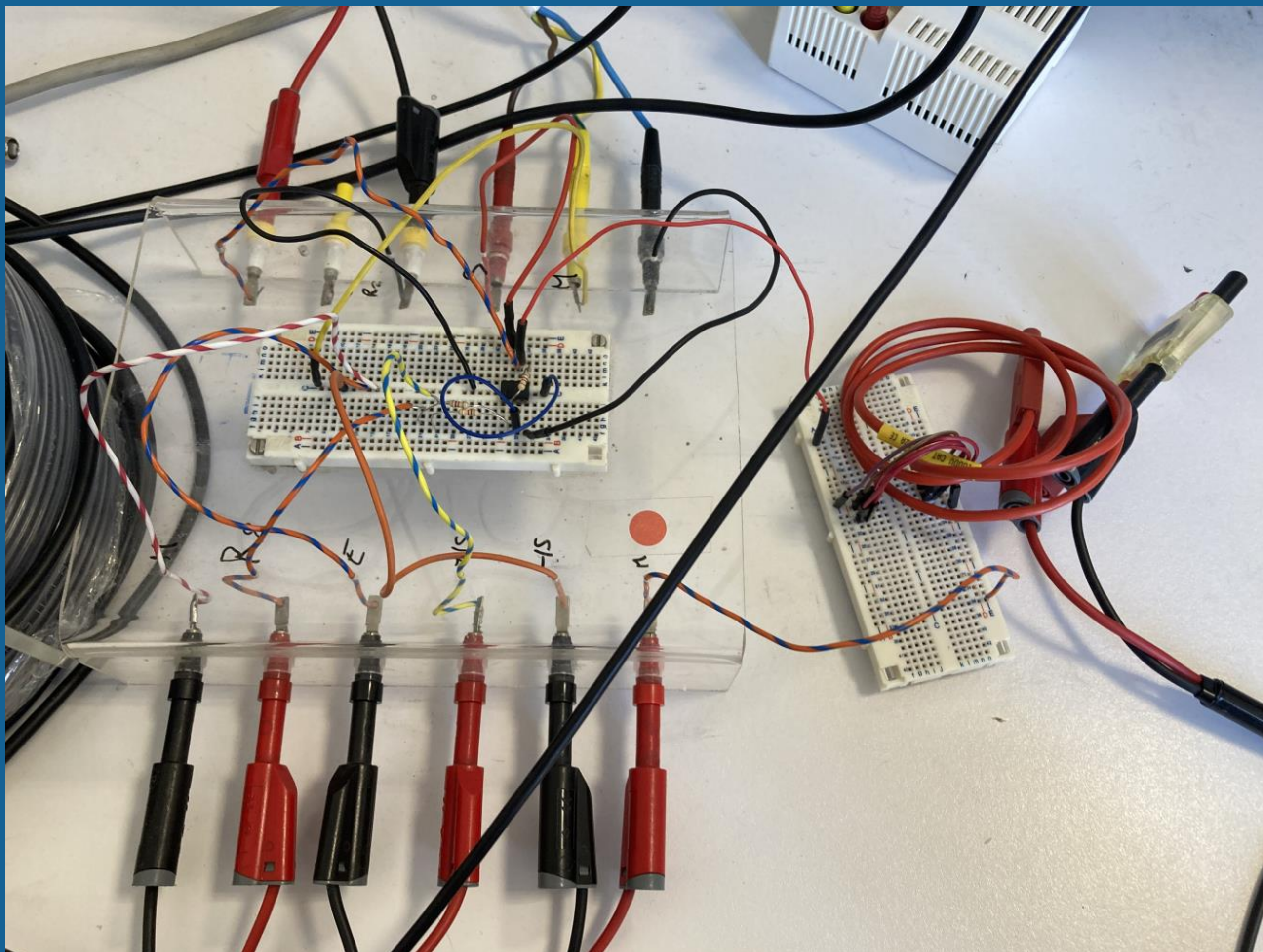


GÉNÉRATION DE BRUIT



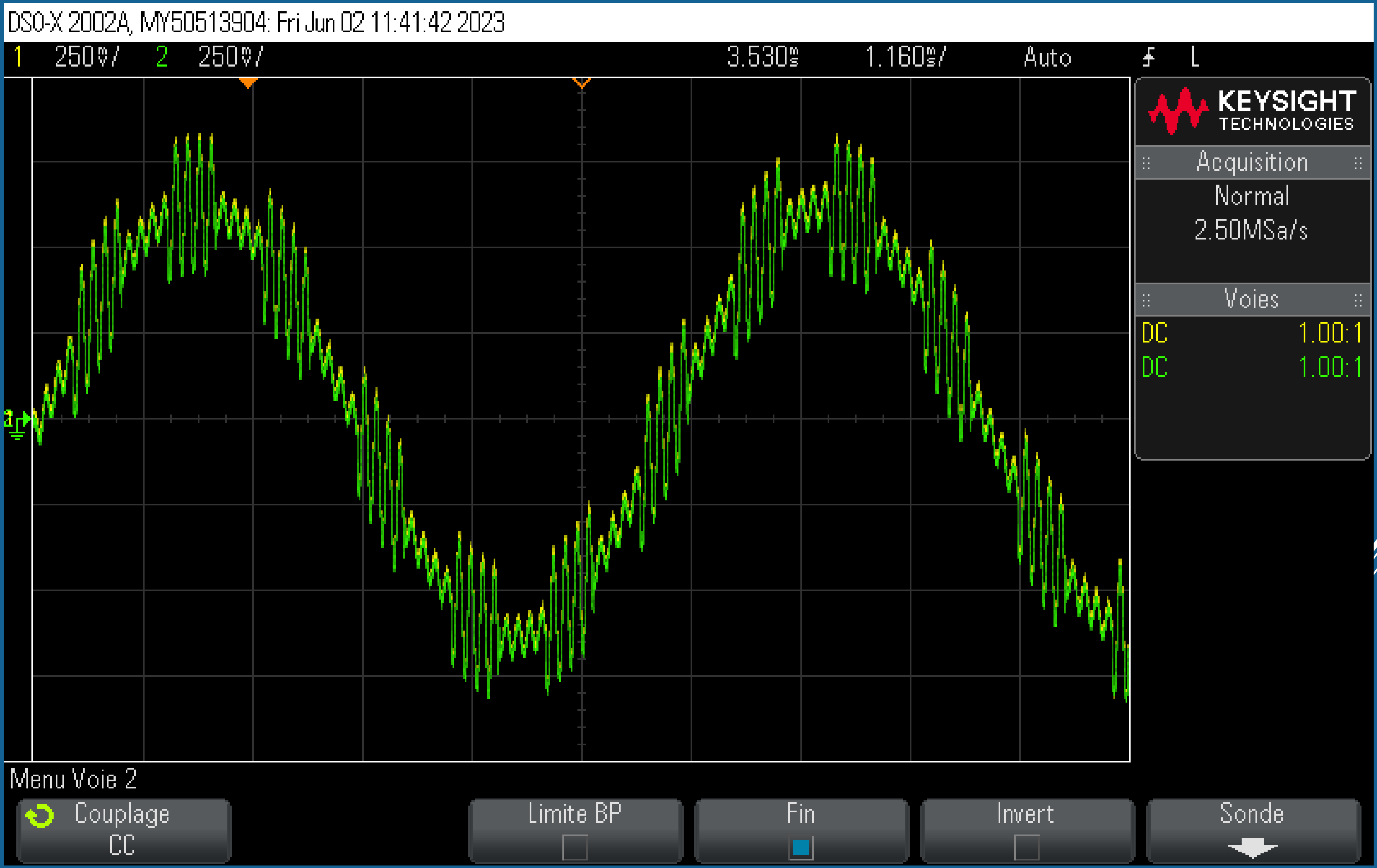
100 m de câble coaxial
+
Plusieurs passages dans la platine d'expérimentation
+
Signal parasite induit

GÉNÉRATION DE BRUIT



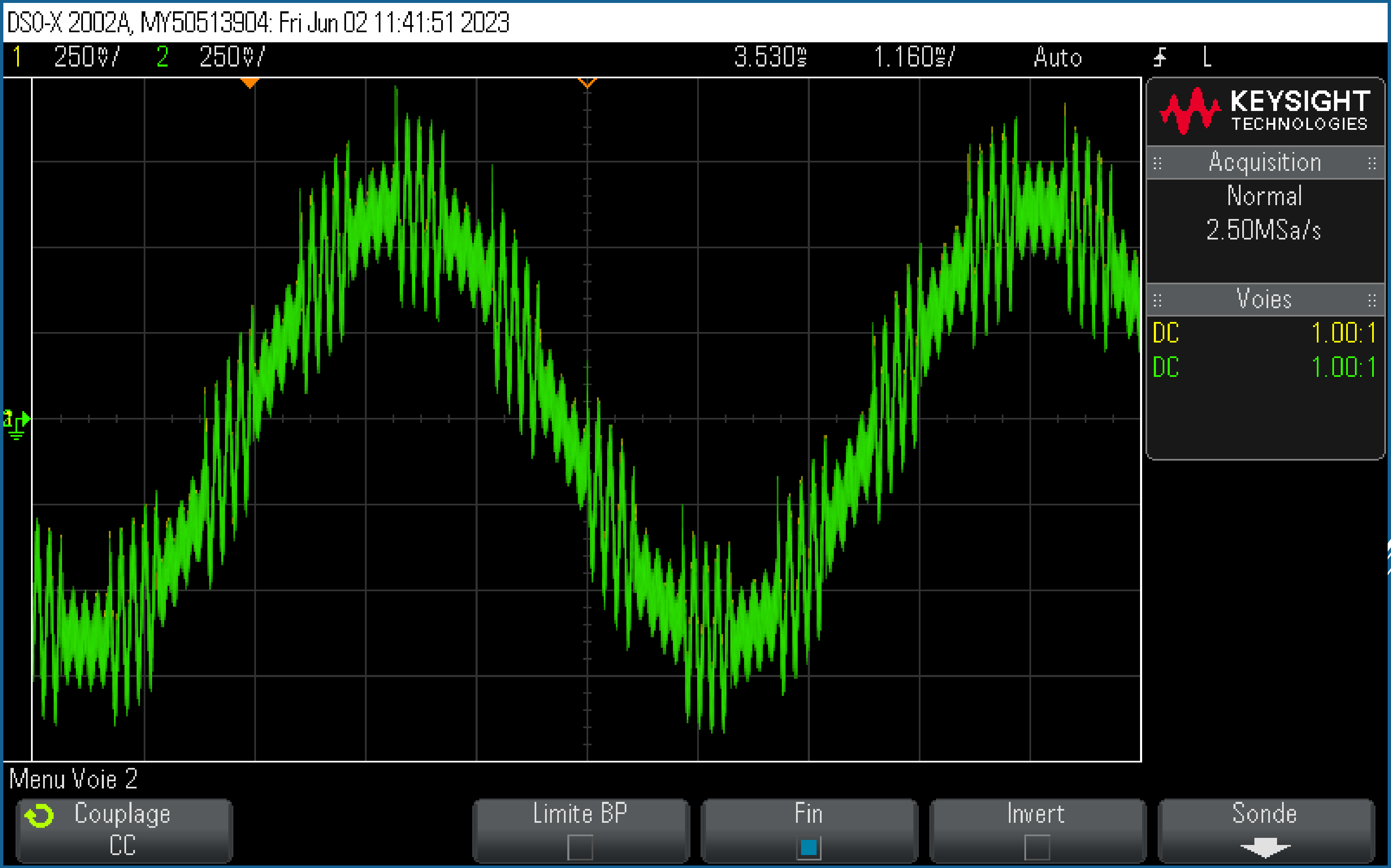
100 m de câble coaxial
+
Plusieurs passages dans la platine d'expérimentation
+
Signal parasite induit

RÉSULTATS



Signal parasite : 100kHz

RÉSULTATS



Signal parasite : 3MHz

I. Modèle physique
   1. Principe
   2. Circuit
   3. Résultats
II. **Méthode des différences finies**
   1. **Équations de Maxwell**
   2. **Discrétisation d'EDP**
III. Solveur spectral
   1. Transformée de Fourier rapide
   2. Solveur pseudo-spectral
   3. Amélioration (Finger Trees)

ÉQUATIONS DE MAXWELL

$$\begin{cases} \mathrm{div}\,\overrightarrow{\mathrm{E}}(\overrightarrow{\mathrm{r}},t) = \dfrac{\rho(\overrightarrow{\mathrm{r}},t)}{\varepsilon_0} \\ \\ \mathrm{div}\,\overrightarrow{\mathrm{B}}(\overrightarrow{\mathrm{r}},t) = 0 \\ \\ \overrightarrow{\mathrm{rot}}\,\overrightarrow{\mathrm{E}}(\overrightarrow{\mathrm{r}},t) = -\dfrac{\partial \overrightarrow{\mathrm{B}}}{\partial t}(\overrightarrow{\mathrm{r}},t) \\ \\ \overrightarrow{\mathrm{rot}}\,\overrightarrow{\mathrm{B}}(\overrightarrow{\mathrm{r}},t) = \mu_0\,\overrightarrow{\mathrm{j}}(\overrightarrow{\mathrm{r}},t) + \mu_0\varepsilon_0\dfrac{\partial \overrightarrow{\mathrm{E}}}{\partial t}(\overrightarrow{\mathrm{r}},t) \end{cases}$$

DISCRÉTISATION

$$\overrightarrow{\mathrm{rot}}\,\vec{\mathrm{a}} = \begin{pmatrix} \frac{\partial a_z}{\partial y} - \frac{\partial a_y}{\partial z} \\ \frac{\partial a_x}{\partial z} - \frac{\partial a_z}{\partial x} \\ \frac{\partial a_y}{\partial x} - \frac{\partial a_x}{\partial y} \end{pmatrix}$$

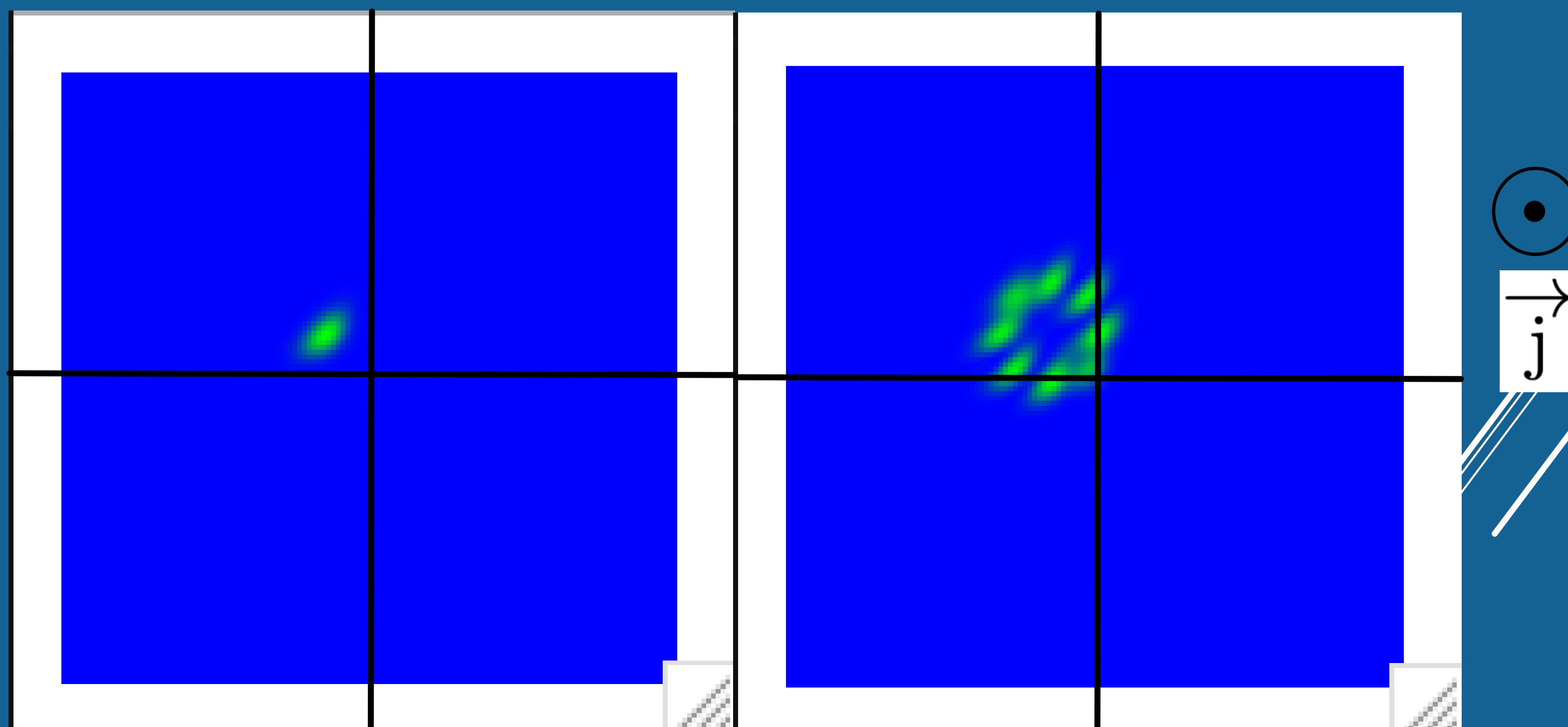
$$\frac{\partial f}{\partial x} \approx \frac{f_{n+1} - f_n}{h}$$

$$\overrightarrow{\mathrm{rot}}\,\vec{\mathrm{a}} \approx \frac{1}{h} \begin{pmatrix} a_z^{(i,j+1,k,n)} - a_z^{(i,j,k,n)} - a_y^{(i,j,k+1,n)} + a_y^{(i,j,k,n)} \\ a_x^{(i,j,k+1,n)} - a_x^{(i,j,k,n)} - a_z^{(i+1,j,k,n)} + a_z^{(i,j,k,n)} \\ a_y^{(i+1,j,k,n)} - a_y^{(i,j,k,n)} - a_x^{(i,j+1,k,n)} + a_x^{(i,j,k,n)} \end{pmatrix}$$

DISCRÉTISATION

$$\frac{\partial f}{\partial x} \approx \frac{f_{n+1} - f_n}{h}$$

Norme du champ magnétique



Fil traversé par un courant

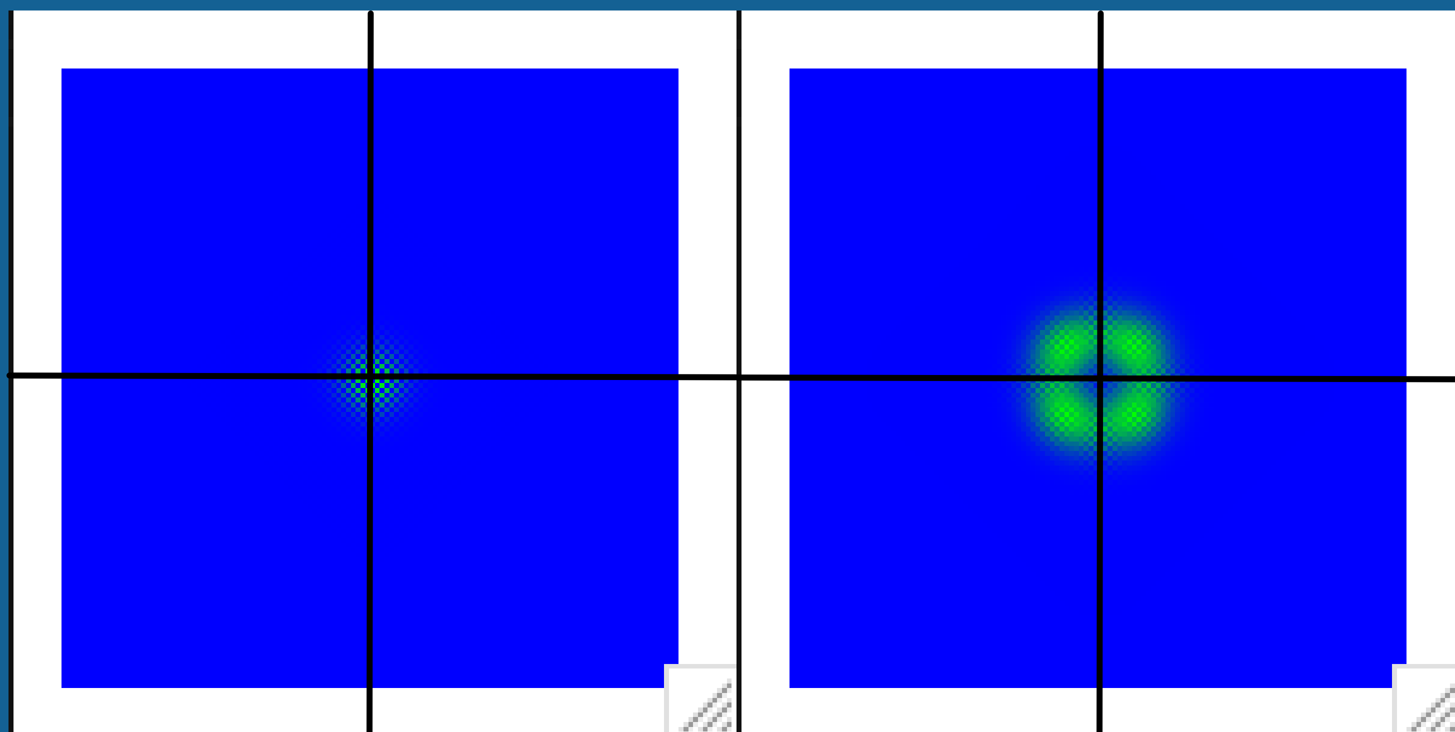
Cylindre de courant

Anisotropie

DISCRÉTISATION

$$\frac{\partial f}{\partial x} \approx \frac{f_{n+1} - f_{n-1}}{2h}$$

Norme du champ magnétique

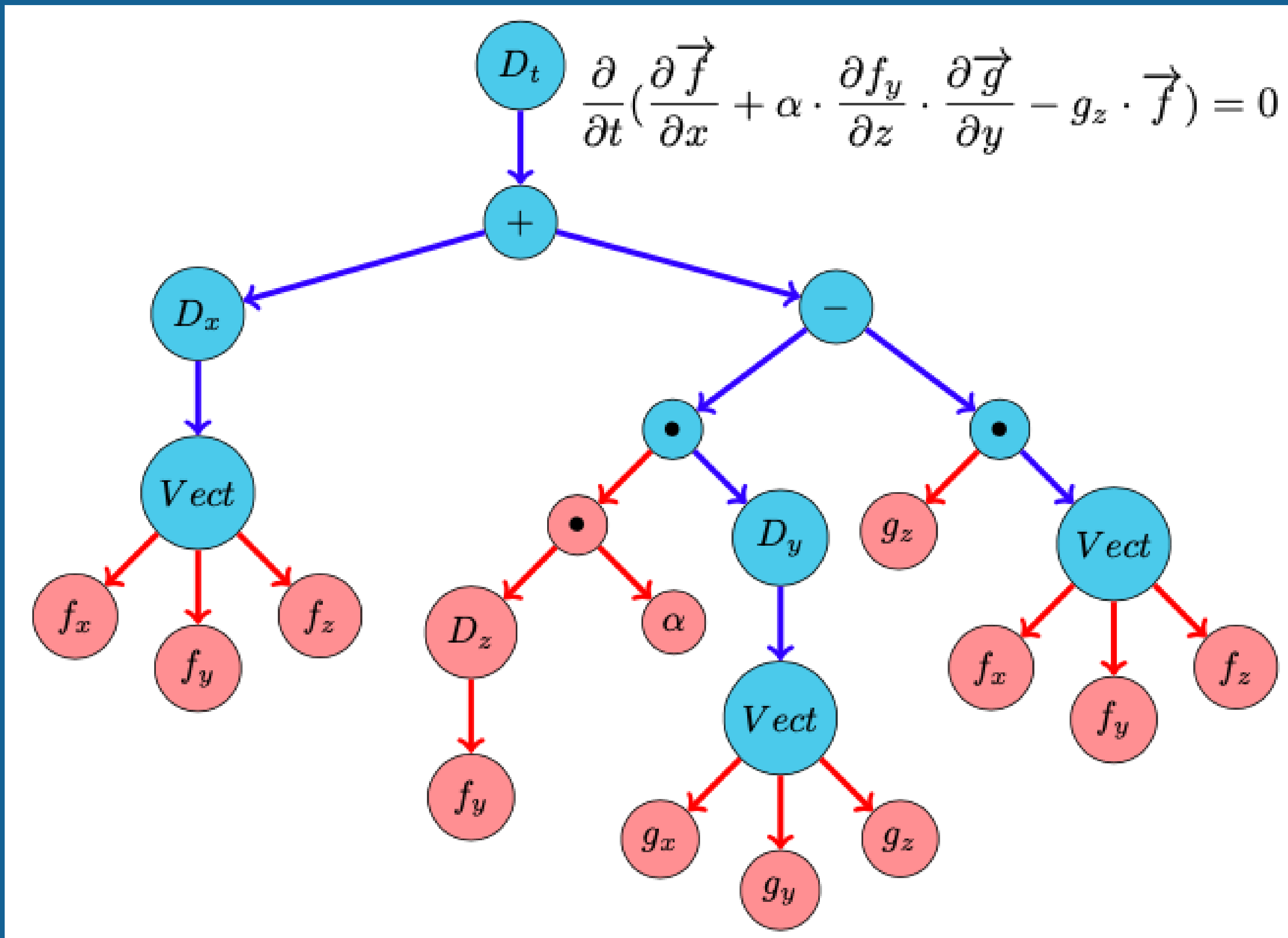


$\odot$ $\vec{j}$

Fil traversé par un courant | Cylindre de courant

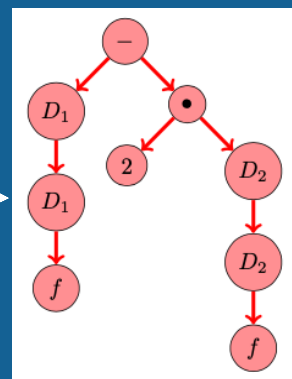
I. Modèle physique
 1. Principe
 2. Circuit
 3. Résultats
II. **Méthode des différences finies**
 1. **Équations de Maxwell**
 2. **Discrétisation d'EDP**
III. Solveur spectral
 1. Transformée de Fourier rapide
 2. Solveur pseudo-spectral
 3. Amélioration (Finger Trees)

REPRÉSENTATION D'EDP



VÉRIFICATION

$$\partial_{1,1} f - 2\partial_{2,2} f = 0$$



$$\begin{pmatrix} i,j & i+1,j & i+2,j \\ -1 & -2 & 1 \end{pmatrix}, \begin{pmatrix} i,j & i,j+1 & i,j+2 \\ -1 & 4 & -2 \end{pmatrix}$$

h = l = 1

$$\frac{f_{i+2,j} - 2f_{i+1,j} + f_{i,j}}{h^2} - 2\frac{f_{i,j+2} - 2f_{i,j+1} + f_{i,j}}{l^2} = 0$$

Équations non couplées

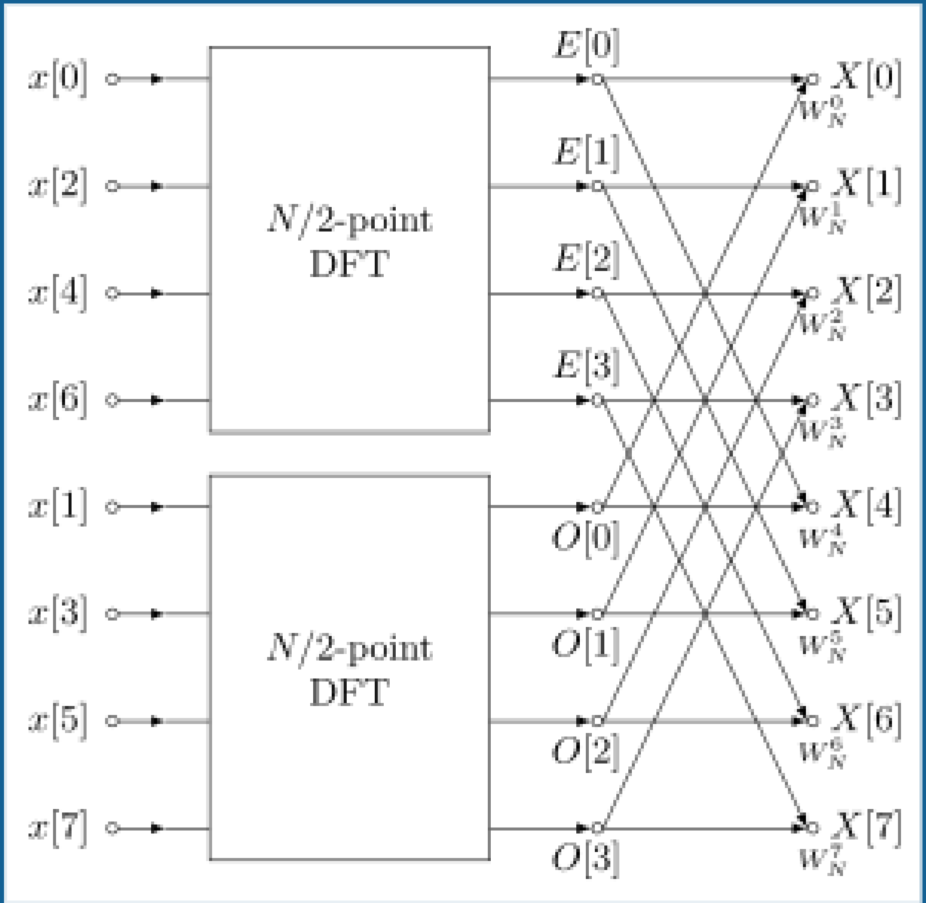
I. Modèle physique
   1. Principe
   2. Circuit
   3. Résultats
II. Méthode des différences finies
   1. Équations de Maxwell
   2. Discrétisation d'EDP
III. **Solveur spectral**
   1. **Transformée de Fourier rapide**
   2. **Solveur pseudo-spectral**
   3. **Amélioration (Finger Trees)**

TRANSFORMÉE DE FOURIER DISCRÈTE

Si f est une fonction de $\mathbb{R}^3$ dans $\mathbb{R}^3$ on définit (sous réserve d'existence):

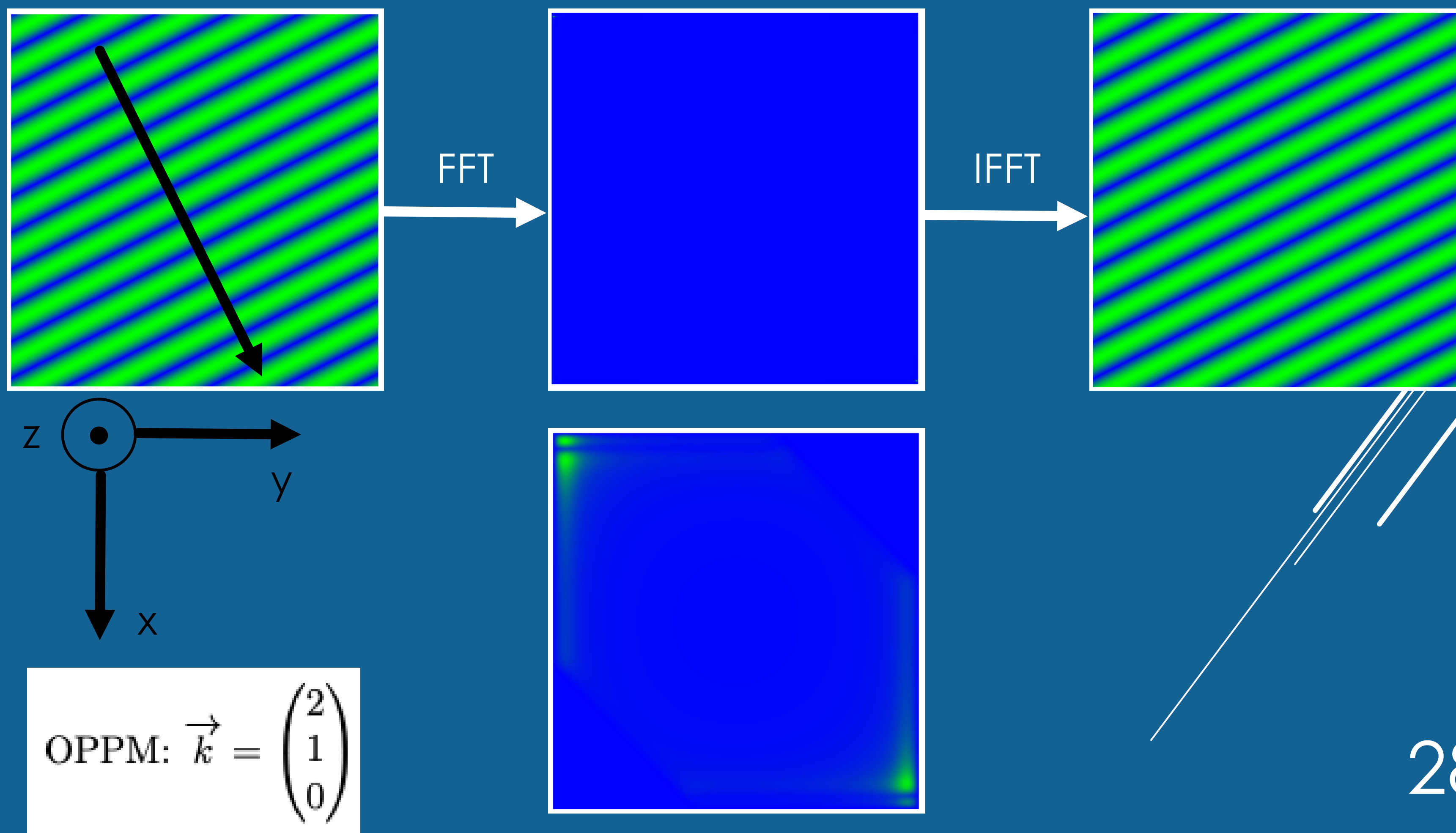
$$\begin{array}{rccc} \mathcal{F}(f): & \mathbb{R}^3 & \longrightarrow & \mathbb{C}^3 \\ & \overrightarrow{k} & \longmapsto & \displaystyle\iiint_{\mathbb{R}^3} exp(-i\overrightarrow{k}\cdot\overrightarrow{r})f(\overrightarrow{r})\,d\tau \end{array}$$

ALGORITHME DE COOLEY-TUKEY



Source: https://en.wikipedia.org/wiki/Cooley–Tukey_FFT_algorithm

ALGORITHME DE COOLEY-TUKEY



I. Modèle physique
   1. Principe
   2. Circuit
   3. Résultats
II. Méthode des différences finies
   1. Équations de Maxwell
   2. Discrétisation d'EDP
III. Solveur spectral
   1. **Transformée de Fourier rapide**
   2. **Solveur pseudo-spectral**
   3. **Amélioration (Finger Trees)**

SOLVEUR PSEUDO-SPECTRAL

$$\begin{cases} i\overrightarrow{\mathrm{k}} \cdot \overrightarrow{\mathcal{E}}(\overrightarrow{\mathrm{k}},t) = \dfrac{\rho(\overrightarrow{\mathrm{k}},t)}{\varepsilon_0} \\ i\overrightarrow{\mathrm{k}} \cdot \overrightarrow{\mathcal{B}}(\overrightarrow{\mathrm{k}},t) = 0 \\ i\overrightarrow{\mathrm{k}} \wedge \overrightarrow{\mathcal{E}}(\overrightarrow{\mathrm{k}},t) = -\dfrac{\partial \overrightarrow{\mathcal{B}}}{\partial t}(\overrightarrow{\mathrm{k}},t) \\ i\overrightarrow{\mathrm{k}} \wedge \overrightarrow{\mathcal{B}}(\overrightarrow{\mathrm{k}},t) = \mu_0 \overrightarrow{\mathcal{J}}(\overrightarrow{\mathrm{k}},t) + \mu_0\varepsilon_0 \dfrac{\partial \overrightarrow{\mathcal{E}}}{\partial t}(\overrightarrow{\mathrm{k}},t) \end{cases}$$

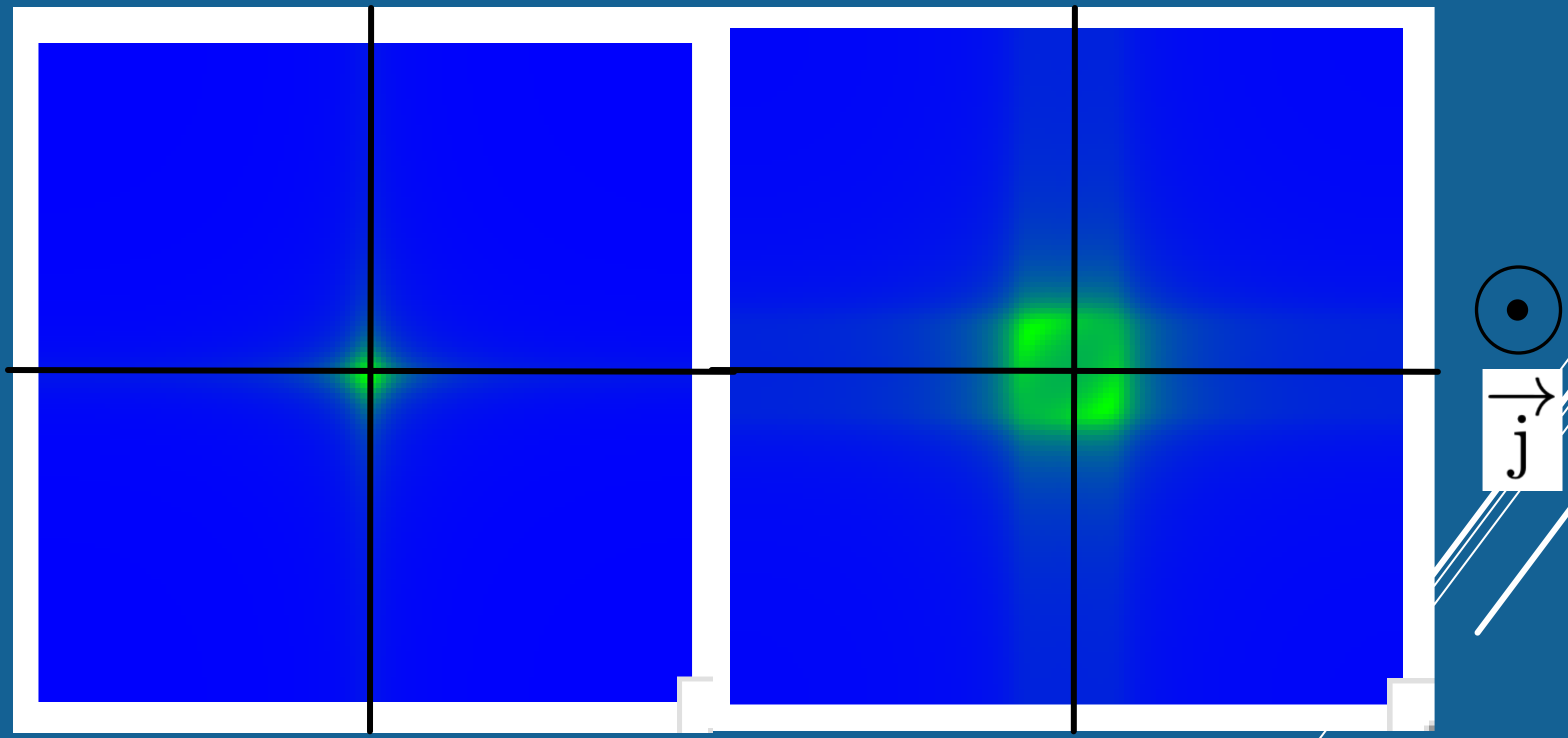
SOLVEUR PSEUDO-SPECTRAL

$$\begin{cases} i\,\overrightarrow{\mathrm{k}} \wedge \overrightarrow{\mathcal{E}^{(\mathrm{n})}}(\overrightarrow{\mathrm{k}}) = -\dfrac{\overrightarrow{\mathcal{B}^{(\mathrm{n+1})}}(\overrightarrow{\mathrm{k}}) - \overrightarrow{\mathcal{B}^{(\mathrm{n})}}(\overrightarrow{\mathrm{k}})}{h} \\ i\,\overrightarrow{\mathrm{k}} \wedge \overrightarrow{\mathcal{B}^{(\mathrm{n})}}(\overrightarrow{\mathrm{k}}) = \mu_0 \overrightarrow{\mathcal{J}^{(\mathrm{n})}}(\overrightarrow{\mathrm{k}}) + \mu_0 \varepsilon_0 \dfrac{\overrightarrow{\mathcal{E}^{(\mathrm{n+1})}}(\overrightarrow{\mathrm{k}}) - \overrightarrow{\mathcal{E}^{(\mathrm{n})}}(\overrightarrow{\mathrm{k}})}{h} \end{cases}$$

Le vecteur densité de courant est imposé
(ce n'est pas une inconnue)

SOLVEUR PSEUDO-SPECTRAL

Norme du champ magnétique



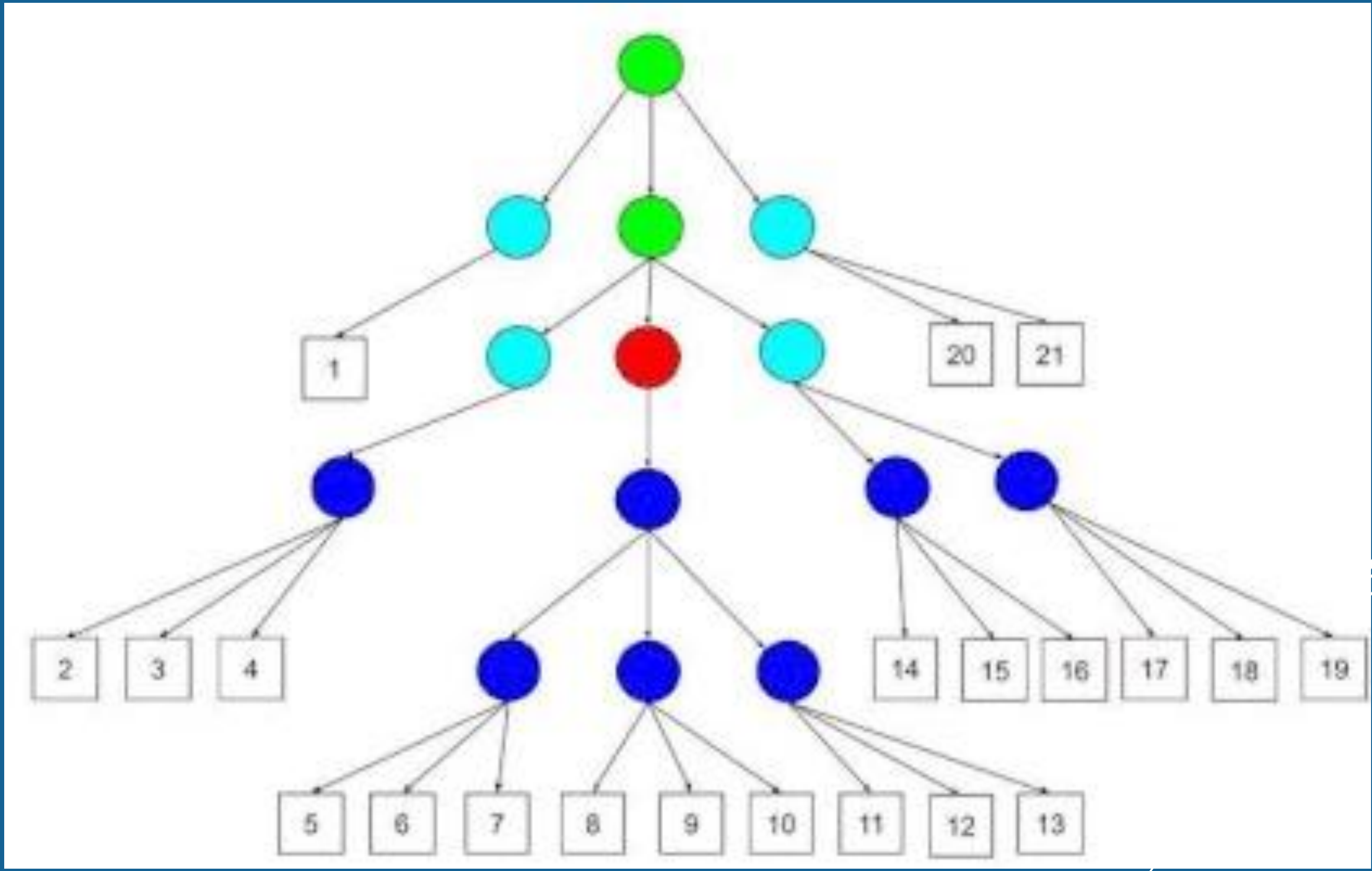
Fil traversé par un courant

Cylindre de courant

Conditions aux limites imposent moins de symétrie

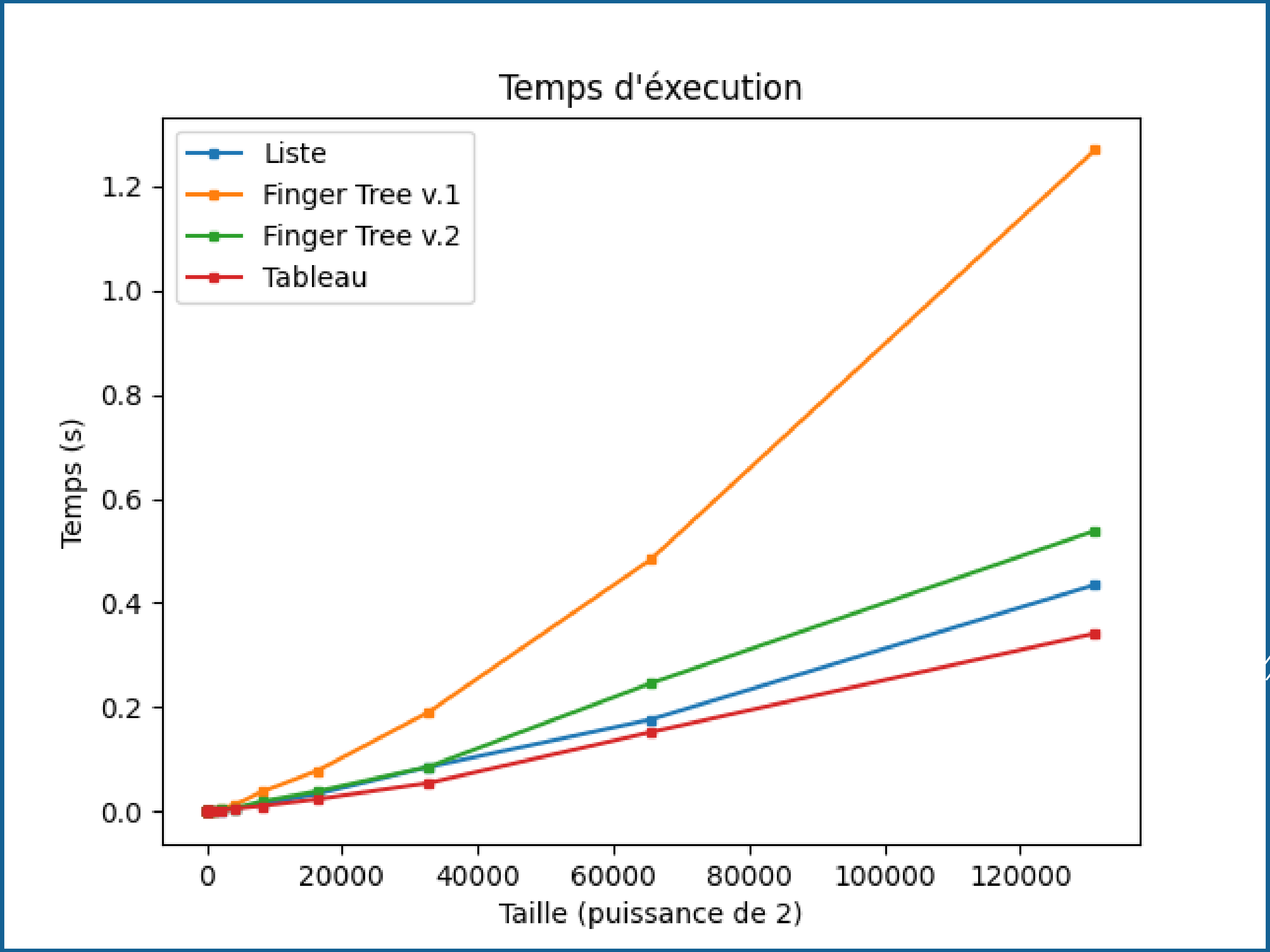
I. Modèle physique
   1. Principe
   2. Circuit
   3. Résultats
II. Méthode des différences finies
   1. Équations de Maxwell
   2. Discrétisation d'EDP
III. Solveur spectral
   1. **Transformée de Fourier rapide**
   2. **Solveur pseudo-spectral**
   3. **Amélioration (Finger Trees)**

FINGER TREES

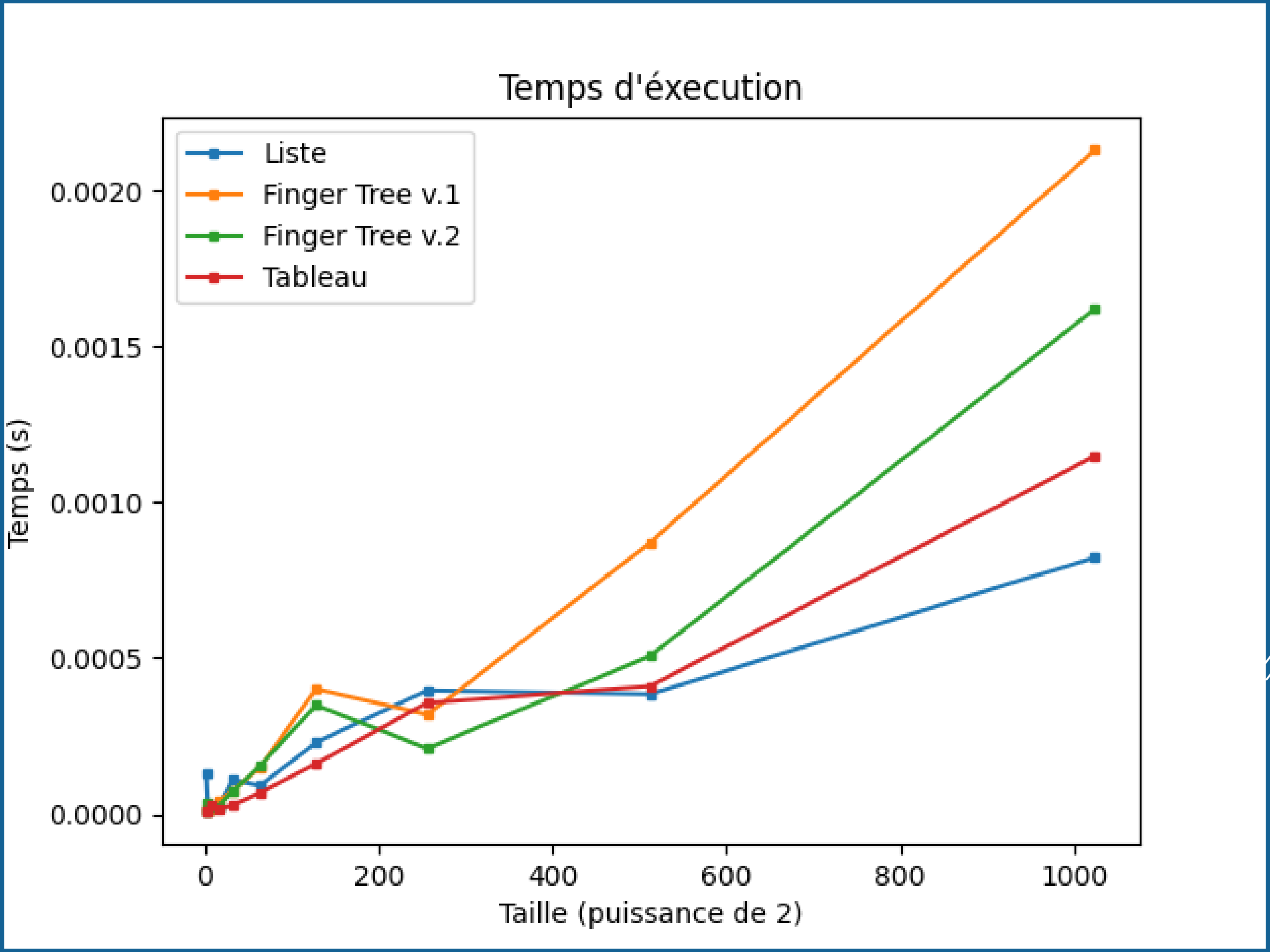


Source: https://en.wikipedia.org/wiki/Finger_tree

COMPLEXITÉS



COMPLEXITÉS



ANNEXES

TYPES: DIFFÉRENCES FINIES

```
type 'a field = {
  mat: 'a array array array array;
  size: int; dx: float }
type em_field = {
  e: float field;
  b: float field;
  mutable j: float field;
  size: int; dx: float}
```

TYPES: REPRÉSENTATION D'EDP

```
type scalar_expr = [
  | `Nul
  | `Unk of int (* Inconnues: Fonctions scalaires; Le paramètre est uniquement un identifiant *)
  | `Const of float
  | `Fun of float -> float
  | `D of int * scalar_expr (* Dérivées partielles: Indices: Temps: 0; x: 1; y: 2; z: 3 *)
  | `Sum of scalar_expr * scalar_expr
  | `Diff of scalar_expr * scalar_expr
  | `Prod of scalar_expr * scalar_expr
]

type expr = [
  | `Vect of scalar_expr array (* Indices: x:0; y: 1; z: 2 *)
  (* Il ne faut plus vérifier la cohérence des dimensions (Cf. v1) *)
  | `D of int * expr (* Dérivées partielles: Indices: Temps: 0; x: 1; y: 2; z: 3 *)
  | `Sum of expr * expr
  | `Diff of expr * expr
  | `Prod of scalar_expr * expr
]
```

TYPES: SOLVEUR SPECTRAL

```
type field = {
  e : Complex.t array array array array; (* Champ électrique (vectoriel) *)
  b : Complex.t array array array array; (* Champ magnétique (vectoriel) *)
  mutable j : Complex.t array array array array; (* Champ vectoriel de densité de courant *)
  size : int; (* Taille de l'espace cubique *)
  dx : float (* Pas de discrétisation spatiale *)
}
```

TYPES: FINGER TREE

```
type _ digit =
  | One: 'a -> 'a digit
  | Two: 'a * 'a -> 'a digit
  | Three: 'a * 'a * 'a -> 'a digit

type _ node =
  | N2: 'a * 'a -> 'a node
  | N3: 'a * 'a * 'a -> 'a node

type _ fingertree =
  | Nil: 'a fingertree
  | Single: 'a -> 'a fingertree
  | More: 'a digit * ('a node) fingertree * 'a digit -> 'a fingertree
```

SIGNATURES

```
utop # #show TIPE.Vect;;
module Vect = TIPE.Vect
module Vect :
  sig
    val ( *.. ) : Complex.t -> Complex.t -> Complex.t
    val ( /.. ) : Complex.t -> Complex.t -> Complex.t
    val ( -.. ) : Complex.t -> Complex.t -> Complex.t
    val ( +.. ) : Complex.t -> Complex.t -> Complex.t
    val norm : float array -> float
    val add : Complex.t array -> Complex.t array -> Complex.t array
    val sub : Complex.t array -> Complex.t array -> Complex.t array
    val scalar : Complex.t -> Complex.t array -> Complex.t array
    val cross_prod : Complex.t array -> Complex.t array -> Complex.t array
    val gen_op : ('a -> 'b -> 'c) -> 'a array -> 'b array -> 'c array
    val gen_scalar : ('a -> 'b -> 'c) -> 'a -> 'b array -> 'c array
    val gen_cross_prod :
      ('a -> 'b -> 'c) -> ('c -> 'c -> 'd) -> 'a array -> 'b array -> 'd array
  end
```

SIGNATURES

```
utop # #show TIPE.Matrix;;
module Matrix = TIPE.Matrix
module Matrix :
  sig
    val iof : float -> int
    val foi : int -> float
    val create_3D : int -> ?m:int -> ?k:int -> 'a -> 'a array array array
    val create_cylinder :
      int -> ?m:int -> ?k:int -> int -> int -> int -> float array array array
    val create_2D : int -> ?m:int -> 'a -> 'a array array
    val add_2D :
      Complex.t array array -> Complex.t array array -> Complex.t array array
    val map2_3D :
      ('a -> 'b -> 'c) ->
      'a array array array -> 'b array array array -> 'c array array array
    val transpose_3D : 'a array array array -> 'a array array array
    val transpose_2D : 'a array array -> 'a array array
    val map : ('a -> 'b) -> 'a array array array -> 'b array array array
    val zip :
      'a array array array * 'a array array array * 'a array array array ->
      'a array array array array
    val unzip :
      'a array array array array ->
      'a array array array * 'a array array array * 'a array array array
    val one_step_scalar_matrix_fft :
      Complex.t array array array -> Complex.t array array array
    val one_step_scalar_matrix_ifft :
      Complex.t array array array -> Complex.t array array array
    val ossm_fft : Complex.t array array array -> Complex.t array array array
    val fft_matrix_3D :
      Complex.t array array array -> Complex.t array array array
    val ifft_matrix_3D :
      Complex.t array array array -> Complex.t array array array
    val fft_matrix_2D : Complex.t array array -> Complex.t array array
    val ifft_matrix_2D : Complex.t array array -> Complex.t array array
    val lines : int -> int -> Complex.t array array
    val sin_mat : int -> int * int -> Complex.t array array
    val random_mat : int -> Complex.t array array
    val show : float array array -> unit
    val show_lin : float array array -> unit
  end
```

SIGNATURES

```
utop # #show TIPE.Lib;;
module Lib = TIPE.Lib
module Lib :
  sig
    val complex_of_int : int * int -> Complex.t
    val complex_of_float : float -> Complex.t
    val ( *.. ) : Complex.t -> Complex.t -> Complex.t
    val print_list' : Complex.t list -> unit
    val print_list : Complex.t list -> unit
    val array2list : 'a array -> 'a list
    val list2array : 'a list -> 'a array
  end
```

SIGNATURES

```
utop # #show TIPE.Fifo;;
module Fifo = TIPE.Fifo
module Fifo :
  sig
    type 'a a23 =
        F of 'a
      | N2 of 'a a23 * 'a a23
      | N3 of 'a a23 * 'a a23 * 'a a23
    type 'a digit =
        One of 'a a23
      | Two of 'a a23 * 'a a23
      | Three of 'a a23 * 'a a23 * 'a a23
    type 'a fingertree =
        Nil
      | Single of 'a a23
      | More of 'a digit * 'a fingertree * 'a digit
    val size23 : 'a a23 -> int
    val heightcomplete : 'a a23 -> int * bool
    val treesize : 'a fingertree -> int
    val size_digit : 'a digit -> int
    val size : 'a fingertree -> int
    val height : 'a fingertree -> int
    val insertleft : 'a a23 -> 'a fingertree -> 'a fingertree
    val insertright : 'a a23 -> 'a fingertree -> 'a fingertree
    exception Empty_fingertree
    val extractleft : 'a fingertree -> 'a a23 * 'a fingertree
    val extractright : 'a fingertree -> 'a a23 * 'a fingertree
    val complete_digit_of_range : int -> 'a digit -> bool
    val complete_of_range : int -> 'a fingertree -> bool
    val cons : 'a -> 'a fingertree -> 'a fingertree
    val snoc : 'a -> 'a fingertree -> 'a fingertree
    val tail : 'a fingertree -> 'a * 'a fingertree
    val init : 'a fingertree -> 'a * 'a fingertree
    val digit2list : 'a digit -> 'a a23 list
    val list2trees : 'a a23 list -> 'a a23 list
    val insert_list_right : 'a fingertree -> 'a a23 list -> 'a fingertree
    val insert_list_left : 'a fingertree -> 'a a23 list -> 'a fingertree
  end
```

SIGNATURES

```
utop # #show TIPE.Fifo_poly;;
module Fifo_poly = TIPE.Fifo_poly
module Fifo_poly :
  sig
    type _ digit =
        One : 'a -> 'a digit
      | Two : 'a * 'a -> 'a digit
      | Three : 'a * 'a * 'a -> 'a digit
    type _ node = N2 : 'a * 'a -> 'a node | N3 : 'a * 'a * 'a -> 'a node
    type _ fingertree =
        Nil : 'a fingertree
      | Single : 'a -> 'a fingertree
      | More : 'a digit * 'a node fingertree * 'a digit -> 'a fingertree
    val insertleft : 'x -> 'x fingertree -> 'x fingertree
    val insertright : 'x -> 'x fingertree -> 'x fingertree
    exception Empty_fingertree
    val extractleft : 'x fingertree -> 'x * 'x fingertree
    val extractright : 'x fingertree -> 'x * 'x fingertree
    val digit2list : 'a digit -> 'a list
    val list2trees : 'a list -> 'a node list
    val insert_list_right : 'a fingertree -> 'a list -> 'a fingertree
    val insert_list_left : 'a fingertree -> 'a list -> 'a fingertree
    val glue : 'x fingertree -> 'x list -> 'x fingertree -> 'x fingertree
    val concat : 'a fingertree -> 'a fingertree -> 'a fingertree
    val list2fingertree : 'a list -> 'a fingertree
    val fingertree2list : 'a fingertree -> 'a list
    val map_digit : ('a -> 'b) -> 'a digit -> 'b digit
    val map_node : ('a -> 'b) -> 'a node -> 'b node
    val map : ('x -> 'y) -> 'x fingertree -> 'y fingertree
    val mapi : (int -> 'a -> 'b) -> 'a fingertree -> 'b fingertree
  end
```

SIGNATURES

```
utop # #show TIPE.Fifo_poly_tail;;
module Fifo_poly_tail = TIPE.Fifo_poly_tail
module Fifo_poly_tail :
  sig
    type _ digit =
        One : 'a -> 'a digit
      | Two : 'a * 'a -> 'a digit
      | Three : 'a * 'a * 'a -> 'a digit
    type _ node = N2 : 'a * 'a -> 'a node | N3 : 'a * 'a * 'a -> 'a node
    type _ fingertree =
        Nil : 'a fingertree
      | Single : 'a -> 'a fingertree
      | More : 'a digit * 'a node fingertree * 'a digit -> 'a fingertree
    val insertleft : 'x -> 'x fingertree -> 'x fingertree
    val insertright : 'x -> 'x fingertree -> 'x fingertree
    exception Empty_fingertree
    val extractleft : 'x fingertree -> 'x * 'x fingertree
    val extractright : 'x fingertree -> 'x * 'x fingertree
    val digit2list : 'a digit -> 'a list
    val list2trees : 'a list -> 'a node list
    val insert_list_right : 'a fingertree -> 'a list -> 'a fingertree
    val insert_list_left : 'a fingertree -> 'a list -> 'a fingertree
    val glue : 'x fingertree -> 'x list -> 'x fingertree -> 'x fingertree
    val concat : 'a fingertree -> 'a fingertree -> 'a fingertree
    val list2fingertree : 'a list -> 'a fingertree
    val fingertree2list : 'a fingertree -> 'a list
    val map_digit : ('a -> 'b) -> 'a digit -> 'b digit
    val map_node : ('a -> 'b) -> 'a node -> 'b node
    val map : ('x -> 'y) -> 'x fingertree -> 'y fingertree
    val iter_digit : ('a -> unit) -> 'a digit -> unit
    val iter_node : ('a -> unit) -> 'a node -> unit
    val iter : ('x -> unit) -> 'x fingertree -> unit
    val size : 'a fingertree -> int
    val mapi_rev_list : (int -> 'a -> 'b) -> 'a list -> 'b list
    val mapi_pure_functional :
      (int -> 'a -> 'b) -> 'a fingertree -> 'b fingertree
    val mapi_digit : ('a -> 'b) -> 'a digit -> 'b digit
    val mapi_node : ('a -> 'b) -> 'a node -> 'b node
    val mapi : (int -> 'a -> 'b) -> 'a fingertree -> 'b fingertree
    val mapi_v2 : (int -> 'a -> 'b) -> 'a fingertree -> 'b fingertree
    val map2_digit : ('a -> 'b -> 'c) -> 'a digit -> 'b digit -> 'c digit
    val map2_node : ('a -> 'b -> 'c) -> 'a node -> 'b node -> 'c node
    val map2 :
      ('x -> 'y -> 'z) -> 'x fingertree -> 'y fingertree -> 'z fingertree
  end
```

SIGNATURES

```
utop # #show TIPE.Fft_list;;
module Fft_list = TIPE.Fft_list
module Fft_list :
  sig
    val divide : 'a list -> 'a list * 'a list * int
    val omega : float -> int -> int -> Complex.t
    val zip :
      float -> int -> int -> 'a list * Complex.t list -> ('a * Complex.t) list
    val add_c : Complex.t * Complex.t -> Complex.t
    val sub_c : Complex.t * Complex.t -> Complex.t
    val fusion :
      float -> int -> Complex.t list -> Complex.t list -> Complex.t list
    val raw_fft : float -> Complex.t list -> Complex.t list
    val fft : Complex.t list -> Complex.t list
    val ifft : Complex.t list -> Complex.t list
  end
```

SIGNATURES

```
utop # #show TIPE.Fft_list_tail;;
module Fft_list_tail = TIPE.Fft_list_tail
module Fft_list_tail :
  sig
    val divide : 'a list -> 'a list * 'a list * int
    val omega : float -> int -> int -> Complex.t
    val zip :
      float -> int -> 'a list -> Complex.t list -> ('a * Complex.t) list
    val add_c : Complex.t * Complex.t -> Complex.t
    val sub_c : Complex.t * Complex.t -> Complex.t
    val fusion :
      float -> int -> Complex.t list -> Complex.t list -> Complex.t list
    val raw_fft : float -> Complex.t list -> Complex.t list
    val fft : Complex.t list -> Complex.t list
    val ifft : Complex.t list -> Complex.t list
  end
```

SIGNATURES

```
utop # #show TIPE.Fft_fifo_v1;;
module Fft_fifo_v1 = TIPE.Fft_fifo_v1
module Fft_fifo_v1 :
  sig
    val divide :
      'a TIPE.Fifo_poly_tail.fingertree ->
      'a TIPE.Fifo_poly_tail.fingertree * 'a TIPE.Fifo_poly_tail.fingertree *
      int
    val omega : float -> int -> int -> Complex.t
    val zip :
      float ->
      int ->
      int ->
      'a TIPE.Fifo_poly_tail.fingertree *
      Complex.t TIPE.Fifo_poly_tail.fingertree ->
      ('a * Complex.t) TIPE.Fifo_poly_tail.fingertree
    val add_c : Complex.t * Complex.t -> Complex.t
    val sub_c : Complex.t * Complex.t -> Complex.t
    val fusion :
      float ->
      int ->
      Complex.t TIPE.Fifo_poly_tail.fingertree ->
      Complex.t TIPE.Fifo_poly_tail.fingertree ->
      Complex.t TIPE.Fifo_poly_tail.fingertree
    val raw_fft :
      float ->
      Complex.t TIPE.Fifo_poly_tail.fingertree ->
      Complex.t TIPE.Fifo_poly_tail.fingertree
    val fft : Complex.t list -> Complex.t list
    val ifft : Complex.t list -> Complex.t list
  end
```

SIGNATURES

```
utop # #show TIPE.Fft_fifo_v2;;
module Fft_fifo_v2 = TIPE.Fft_fifo_v2
module Fft_fifo_v2 :
  sig
    val divide :
      'a TIPE.Fifo_poly_tail.fingertree ->
      'a TIPE.Fifo_poly_tail.fingertree * 'a TIPE.Fifo_poly_tail.fingertree *
      int
    val omega : float -> int -> int -> Complex.t
    val fusion :
      float ->
      int ->
      Complex.t TIPE.Fifo_poly_tail.fingertree ->
      Complex.t TIPE.Fifo_poly_tail.fingertree ->
      Complex.t TIPE.Fifo_poly_tail.fingertree
    val raw_fft :
      float ->
      Complex.t TIPE.Fifo_poly_tail.fingertree ->
      Complex.t TIPE.Fifo_poly_tail.fingertree
    val fft :
      Complex.t TIPE.Fifo_poly_tail.fingertree ->
      Complex.t TIPE.Fifo_poly_tail.fingertree
    val ifft :
      Complex.t TIPE.Fifo_poly_tail.fingertree ->
      Complex.t TIPE.Fifo_poly_tail.fingertree
  end
```

SIGNATURES

```
utop # #show TIPE.Fft_array;;
module Fft_array = TIPE.Fft_array
module Fft_array :
  sig
    val divide : 'a array -> 'a array * 'a array
    val omega : float -> int -> int -> Complex.t
    val raw_fft : float -> Complex.t array -> Complex.t array
    val fft : Complex.t array -> Complex.t array
    val ifft : Complex.t array -> Complex.t array
    val transpose : 'a array array -> 'a array array
    val fft_vect : Complex.t array array -> Complex.t array array
  end
```

SIGNATURES

```
utop # #show TIPE.Expr_gen;;
module Expr_gen = TIPE.Expr_gen
module Expr_gen :
  sig
    type expr =
        Unk of int
      | Fun of string
      | Vect of expr array
      | D of int * expr
      | Sum of expr * expr
      | Diff of expr * expr
      | Prod of expr * expr
    exception Dimension
    val dimension_image : expr -> int
    val dimension_domain : expr -> int
    val diff_class : expr -> int
    val max_index_unk : expr -> int
    val div : expr -> expr
    val grad : expr -> expr
    val rot : expr -> expr
    val laplace_scalar : expr -> expr
    val laplace_vect : expr -> expr
    val eliminate_1D_vectors : expr -> expr
    val extract_vect : expr -> expr array
    val discretization_scalar : float array -> expr -> float array array array
  end
```

SIGNATURES

```
utop # #show TIPE.Expr_gen_v2;;
module Expr_gen_v2 = TIPE.Expr_gen_v2
module Expr_gen_v2 :
  sig
   val extract_vect : expr -> scalar_expr array
   val scalar : expr -> expr -> scalar_expr
   val cross : expr -> expr -> expr
   val grad : scalar_expr -> expr
   val div : expr -> scalar_expr
   val rot : expr -> expr
   val laplace_scalar : scalar_expr -> scalar_expr
   val laplace_vect : expr -> expr
   exception Not_lswcc
   val lswcc_discretization :
     float array -> scalar_expr -> float array array array
   exception Non_linear
   val linear_discretization :
     float array -> scalar_expr -> (float -> float) array array array
   val vect_lswcc_discretization :
     float array -> expr -> float array array array array
 end
```

SIGNATURES

```
utop # #show TIPE.Exe_funs;;
module Exe_funs = TIPE.Exe_funs
module Exe_funs :
  sig
    val test_times_fft1 : int -> unit
    val test_times_fft2 : int -> unit
    val test_correct_fft : unit -> unit
    val test_laplace_ca : unit -> unit
    val test_2D_matrix_fft : unit -> unit
    val test_2D_matrix_fft_2 : unit -> unit
    val test_spect_ca_1 : unit -> unit
    val test_spect_ca_point : unit -> unit
    val test_spect_ca_cyl : unit -> unit
    val test_cell_aut_v2_point : unit -> unit
    val test_cell_aut_v2_cyl : unit -> unit
    val test_cell_aut_v3_point : unit -> unit
    val test_cell_aut_v3_cyl : unit -> unit
```

SIGNATURES

```
utop # #show TIPE.Cell_aut;;
module Cell_aut = TIPE.Cell_aut
module Cell_aut :
  sig
    type 'a field = { mat : 'a array array array array; size : int; }
    val norm_field : float field -> float array array array
    val create : int -> 'a array -> 'a field
    val laplace_op :
      float field ->
      float field ->
      float -> float -> float -> int -> int -> int -> float array
    val update :
      float field -> float field -> float -> float -> float -> float field
    val show : float field -> int -> unit
  end
```

SIGNATURES

```
utop # #show TIPE.Cell_aut_v2;;
module Cell_aut_v2 = TIPE.Cell_aut_v2
module Cell_aut_v2 :
  sig
    type 'a field = {
      mat : 'a array array array array;
      size : int;
      dx : float;
    }
    type em_field = {
      e : float field;
      b : float field;
      mutable j : float field;
      size : int;
      dx : float;
    }
    val ( ++ ) : float array -> float array -> float array
    val ( -- ) : float array -> float array -> float array
    val ( ** ) : float -> float array -> float array
    val ( *^ ) : float array -> float array -> float array
    val norm_field : float field -> float array array array
    val create_f : int -> 'a array -> float -> 'a field
    val create_em_f : int -> float array -> float -> em_field
    val rot : float field -> int -> int -> int -> float array
    val update_e :
      float field ->
      float field ->
      float field ->
      int -> int -> int -> float -> float -> float -> float array
    val update_b :
      float field -> float field -> int -> int -> int -> float -> float array
    val update : em_field -> em_field -> float -> float -> float -> unit
    val show : float field -> int -> unit
    val show_lin : float field -> int -> unit
  end
```

SIGNATURES

```
utop # #show TIPE.Cell_aut_v3;;
module Cell_aut_v3 = TIPE.Cell_aut_v3
module Cell_aut_v3 :
  sig
    type 'a field = {
      mat : 'a array array array array;
      size : int;
      dx : float;
    }
    type em_field = {
      e : float field;
      b : float field;
      mutable j : float field;
      size : int;
      dx : float;
    }
    val ( ++ ) : float array -> float array -> float array
    val ( -- ) : float array -> float array -> float array
    val ( ** ) : float -> float array -> float array
    val ( *^ ) : float array -> float array -> float array
    val norm_field : float field -> float array array array
    val create_f : int -> 'a array -> float -> 'a field
    val create_em_f : int -> float array -> float -> em_field
    val rot : float field -> int -> int -> int -> float array
    val update_e :
      em_field -> int -> int -> int -> float -> float -> float -> float array
    val update_b : em_field -> int -> int -> int -> float -> float array
    val update : em_field -> em_field -> float -> float -> float -> unit
    val show : float field -> int -> unit
    val show_lin : float field -> int -> unit
  end
```

SIGNATURES

```
utop # #show TIPE.Cell_aut_spect;;
module Cell_aut_spect = TIPE.Cell_aut_spect
module Cell_aut_spect :
  sig
    type field = {
      e : Complex.t array array array array;
      b : Complex.t array array array array;
      mutable j : Complex.t array array array array;
      size : int;
      dx : float;
    }
    val create : int -> Complex.t array -> float -> field
    val ( *.. ) : Complex.t -> Complex.t -> Complex.t
    val ( /.. ) : Complex.t -> Complex.t -> Complex.t
    val ( -.. ) : Complex.t -> Complex.t -> Complex.t
    val ( +.. ) : Complex.t -> Complex.t -> Complex.t
    val ( ++ ) : Complex.t array -> Complex.t array -> Complex.t array
    val ( -- ) : Complex.t array -> Complex.t array -> Complex.t array
    val ( ** ) : Complex.t -> Complex.t array -> Complex.t array
    val ( *^ ) : Complex.t array -> Complex.t array -> Complex.t array
    val i : Complex.t
    val ( *** ) : ('a -> 'b) -> ('c -> 'a) -> 'c -> 'b
    val norm_field :
      Complex.t array array array array -> float array array array
    val update : field -> field -> field -> float -> float -> float -> unit
    val show : Complex.t array array array -> unit
  end
```