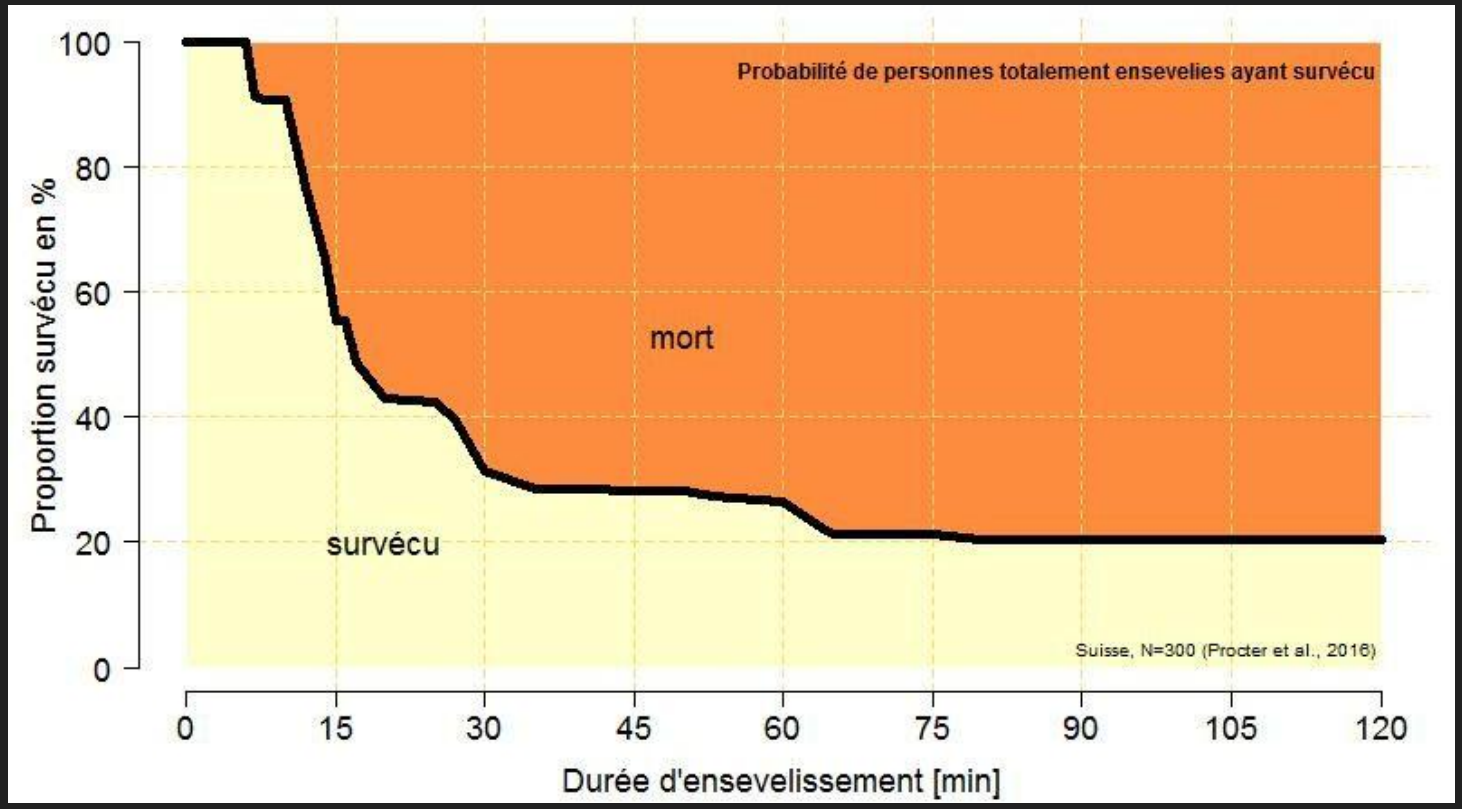
Théophile VEILLON
N° de candidat : 41729

Comment protéger un skieur hors piste ?

Le besoin :

un moyen efficace autonome de prévenir les secours en cas d'accident d'avalanche.



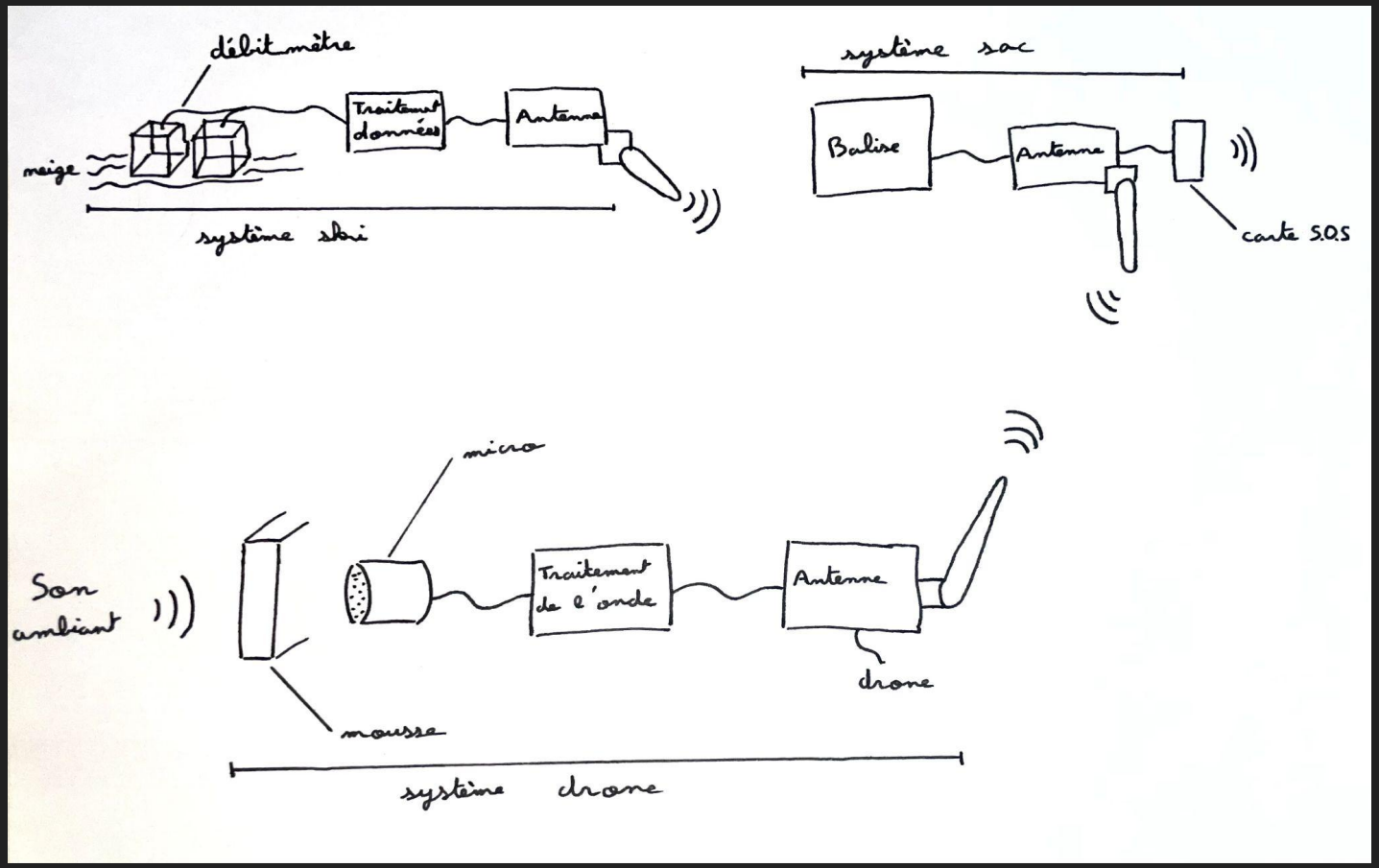
Probabilité de personnes totalement ensevelies ayant survécu

source : SLF

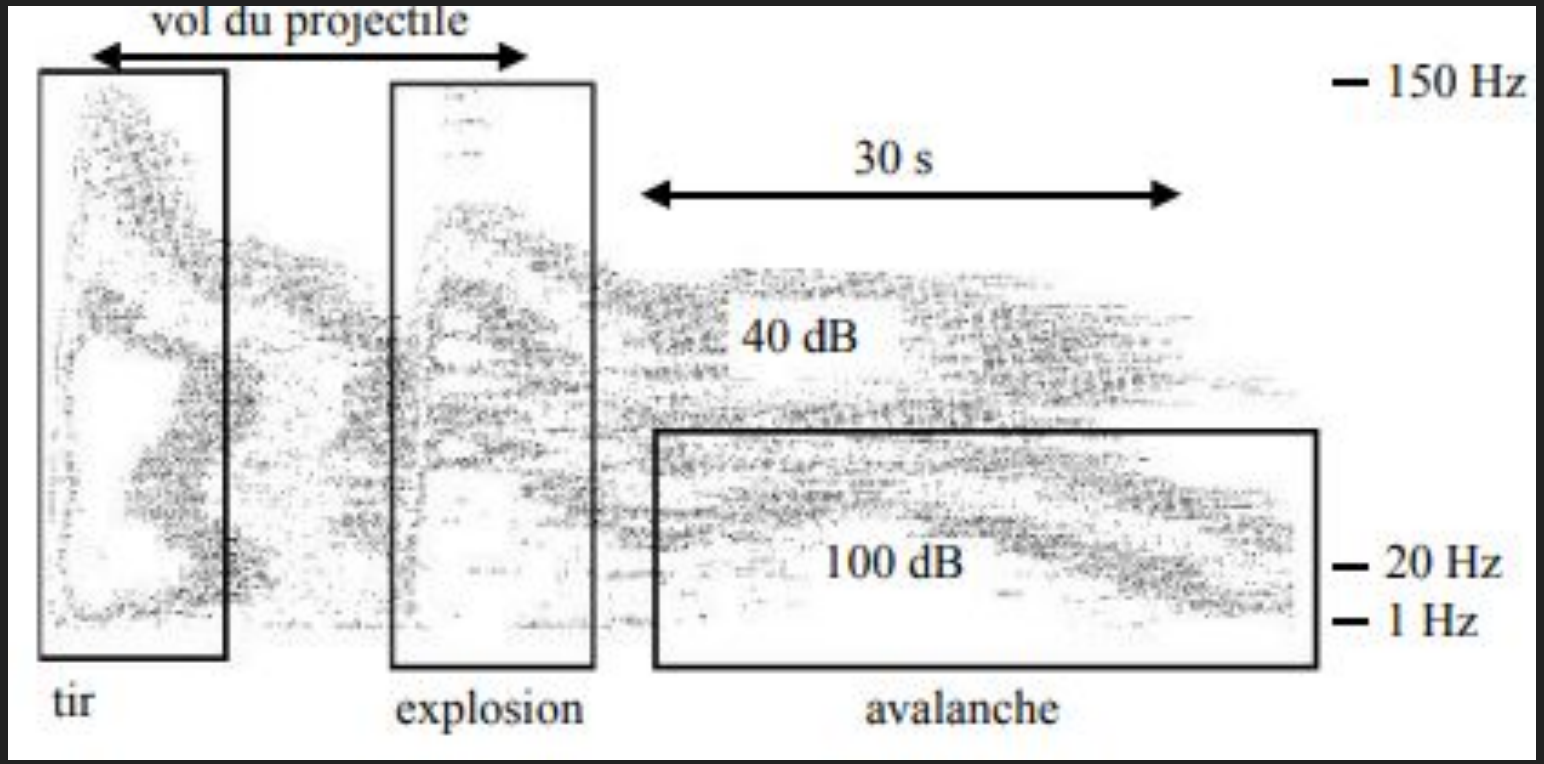
La solution proposée :

Système :

- micro
- balise
- débitmètre

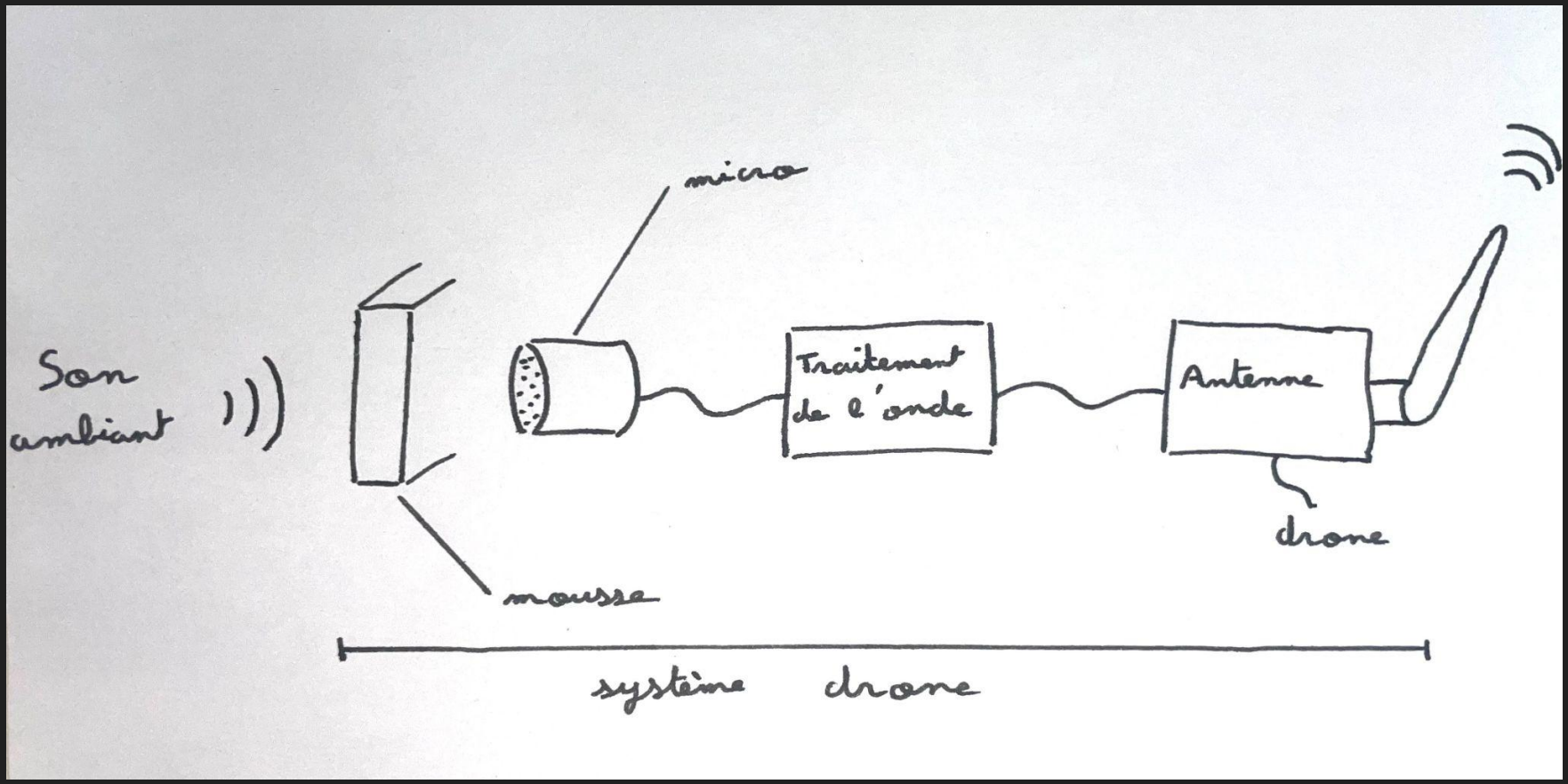


Captation et traitement du son émis par une avalanche



Sonogramme d'une avalanche déclenché artificiellement
Source : IAV

Filtrage de l’onde sonore



système de captation de l’onde sonore émise par l’avalanche

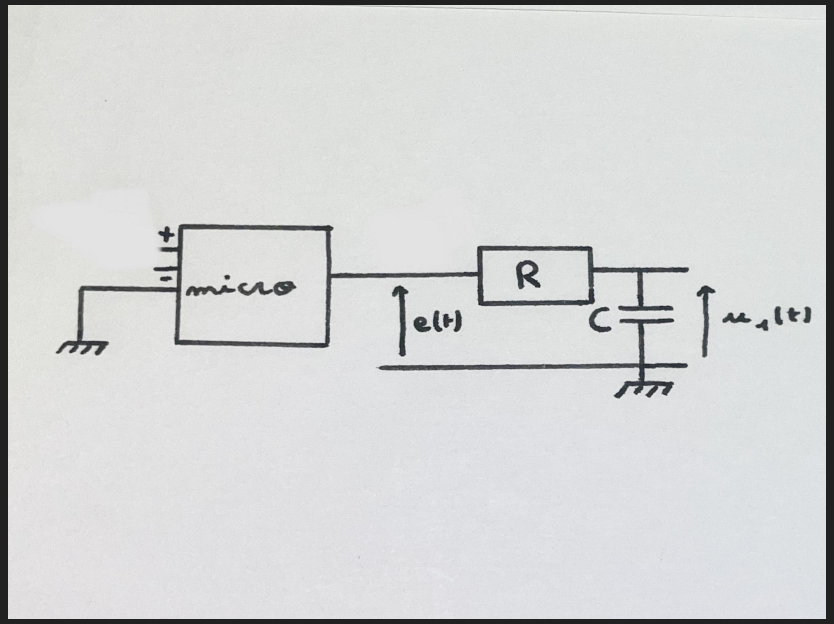
Numérisation de l’onde sonore

f_e = 200 MHz

$f_e > 2f_{max}$ avec f_{max} = 100Hz

En pratique, le CAN LTC 2058 réalise cette opération parfaitement.

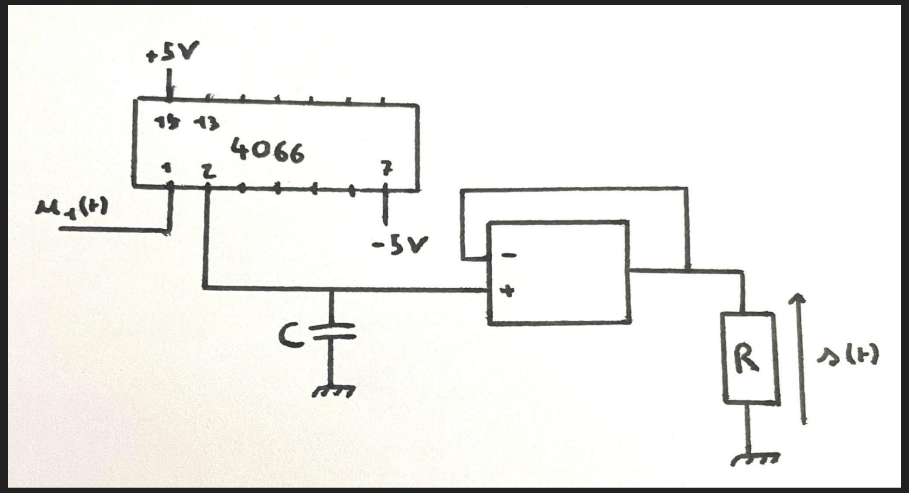
Modélisation :

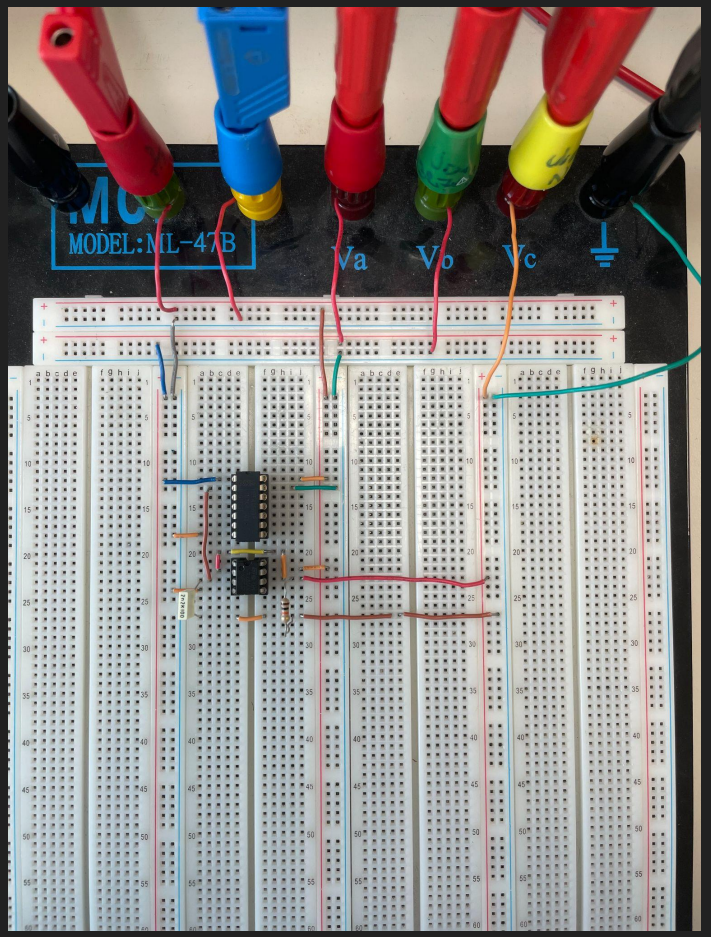


filtre anti-repliement / modélisation d'une mousse acoustique

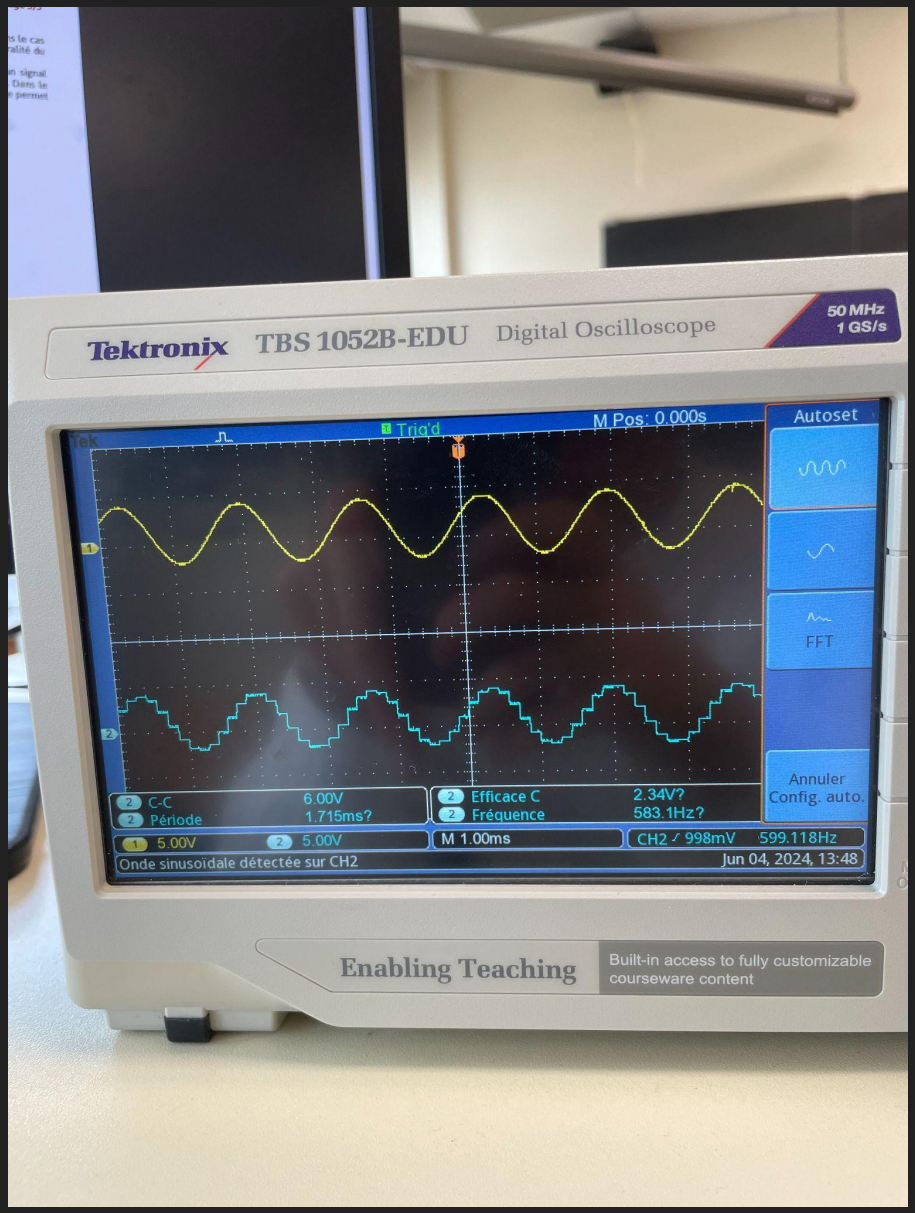
La fréquence de coupure de ce filtre est $f_c = 1/2\pi RC$

Pour avoir une fréquence de coupure autour de 200 Hz, on peut prendre R=1kΩ et C = 80 µF.





CAN échantillonneur / bloqueur


Tektronix TBS 1052B-EDU Digital Oscilloscope
50 MHz
1 GS/s
Trig'd
M Pos: 0.000s
Autoset
FFT
Annuler
Config. auto.
C-C 6.00V
Période 1.715ms?
Efficace C 2.34V?
Fréquence 583.1Hz?
5.00V
5.00V
M 1.00ms
CH2 ⟋ 998mV
599.118Hz
Onde sinusoïdale détectée sur CH2
Jun 04, 2024, 13:48
Enabling Teaching
Built-in access to fully customizable courseware content

Algorithme des k plus proches voisins

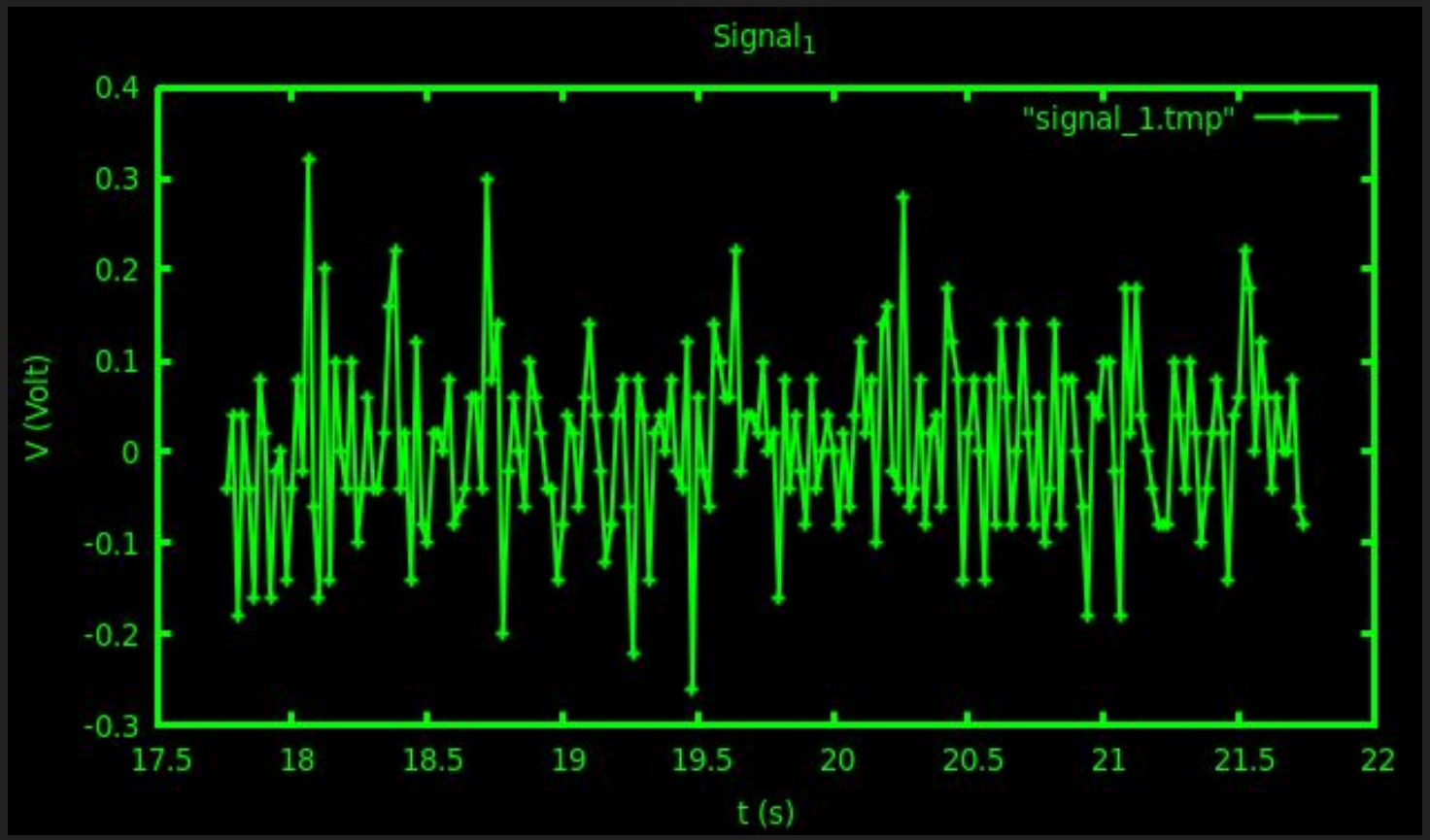
Algorithme décisionnel :

Entrée : spectre d’une onde quelconque

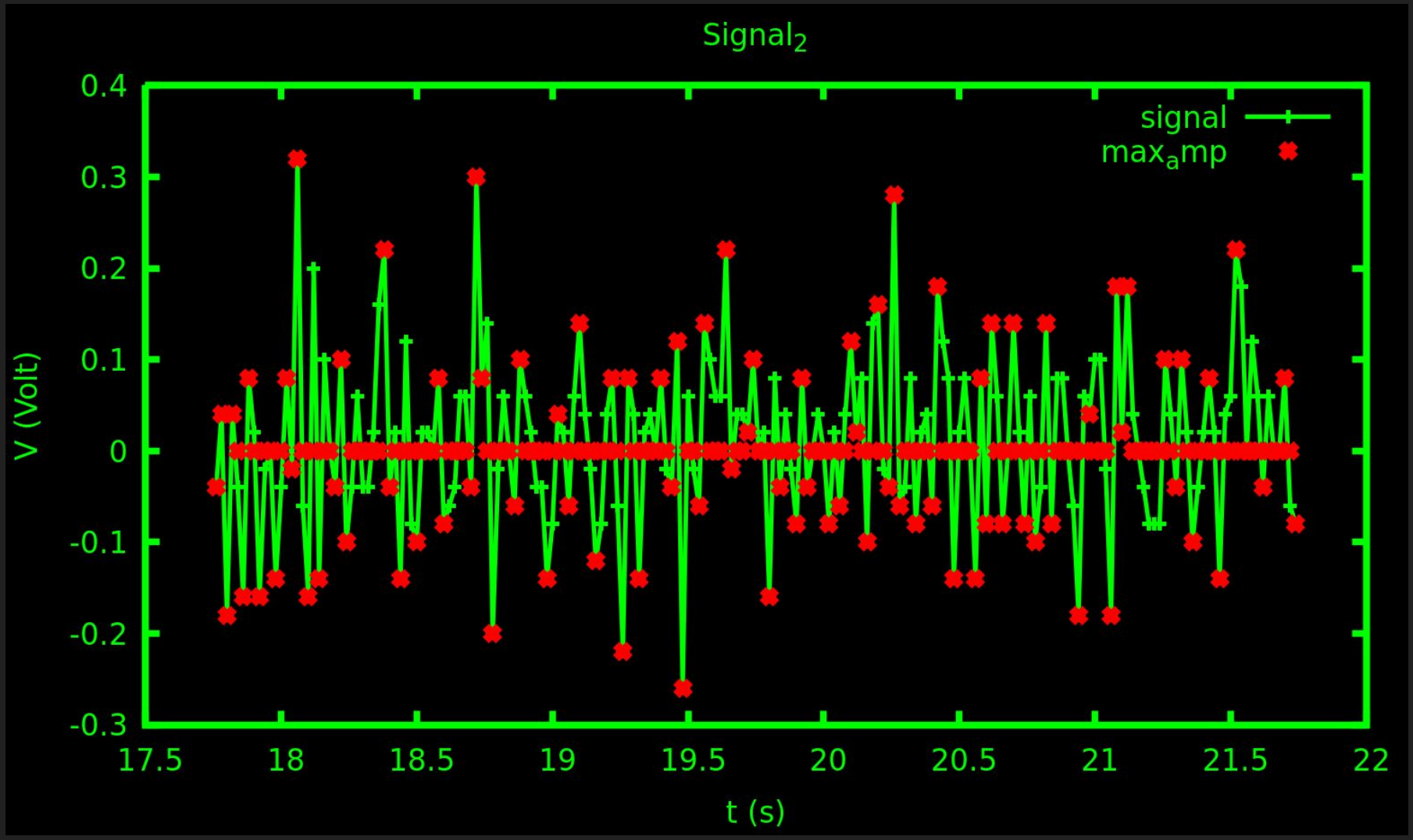
Sortie : un booléen

Question : est ce que le spectre donnée en entrée correspond à un spectre d’avalanche ?

Choix de la distance



signal d’avalanche via gnuplot



principe de l’algorithme

Piste d’amélioration

- les arbres D-dimensionnelles
- augmenter la base de données

ANNEXE

- Poursuite avec le principe de calcul liée au glissement de terrain
- code

Détection d'un glissement de terrain

Pour des température entre -11°C et -1°C,

$$h^4 = \frac{3cr^2\eta^2v^2}{2L\rho}$$

h = épaisseur de film d'eau

c = 3.05 MPa le prapport surface de contact/charge

r = 0.001 m le rayon de contact

η = 10 Pa.s la viscosité dynamique de l'eau

v = 16 m/s La vitesse moyenne du skieur

L = 334 J/g la chaleur latente de fusion de la neige

ρ = 100 Kg/m^3 la masse volumique de la neige poudreuse

$\Longrightarrow$ h = 0.004 m soit h = 0.4 cm

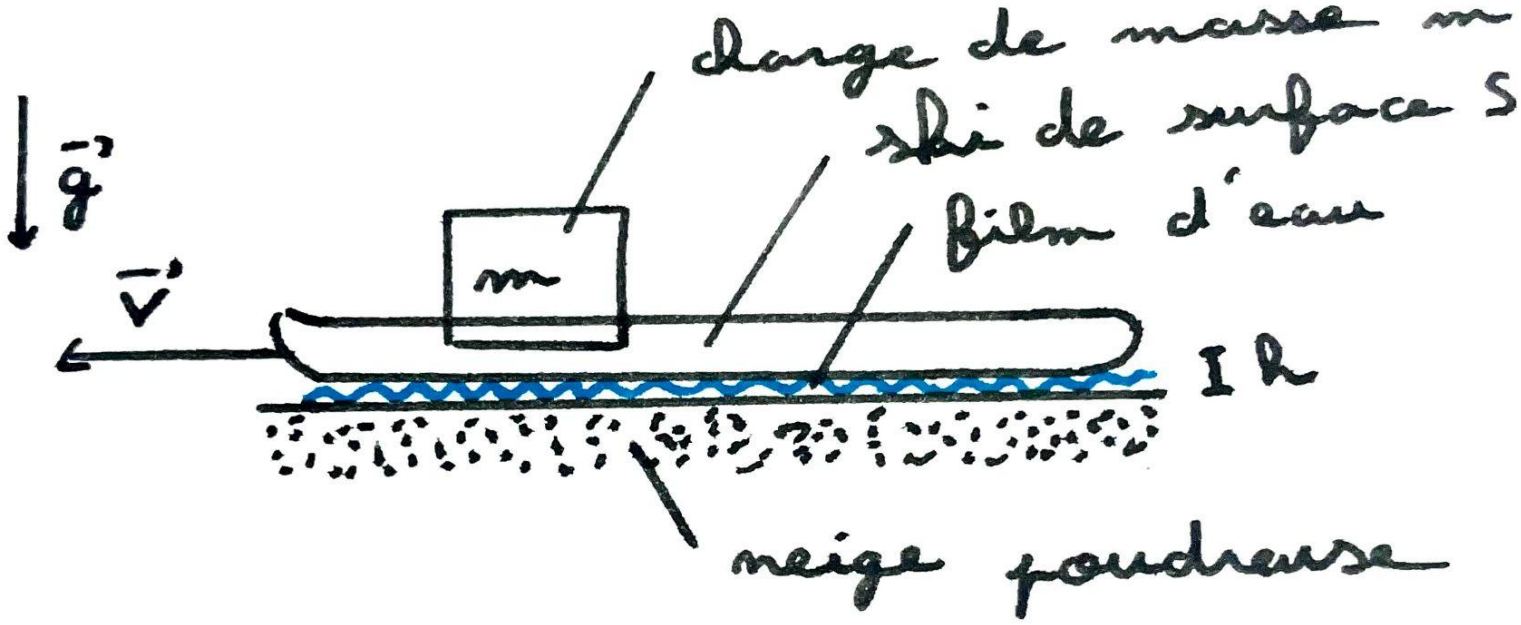
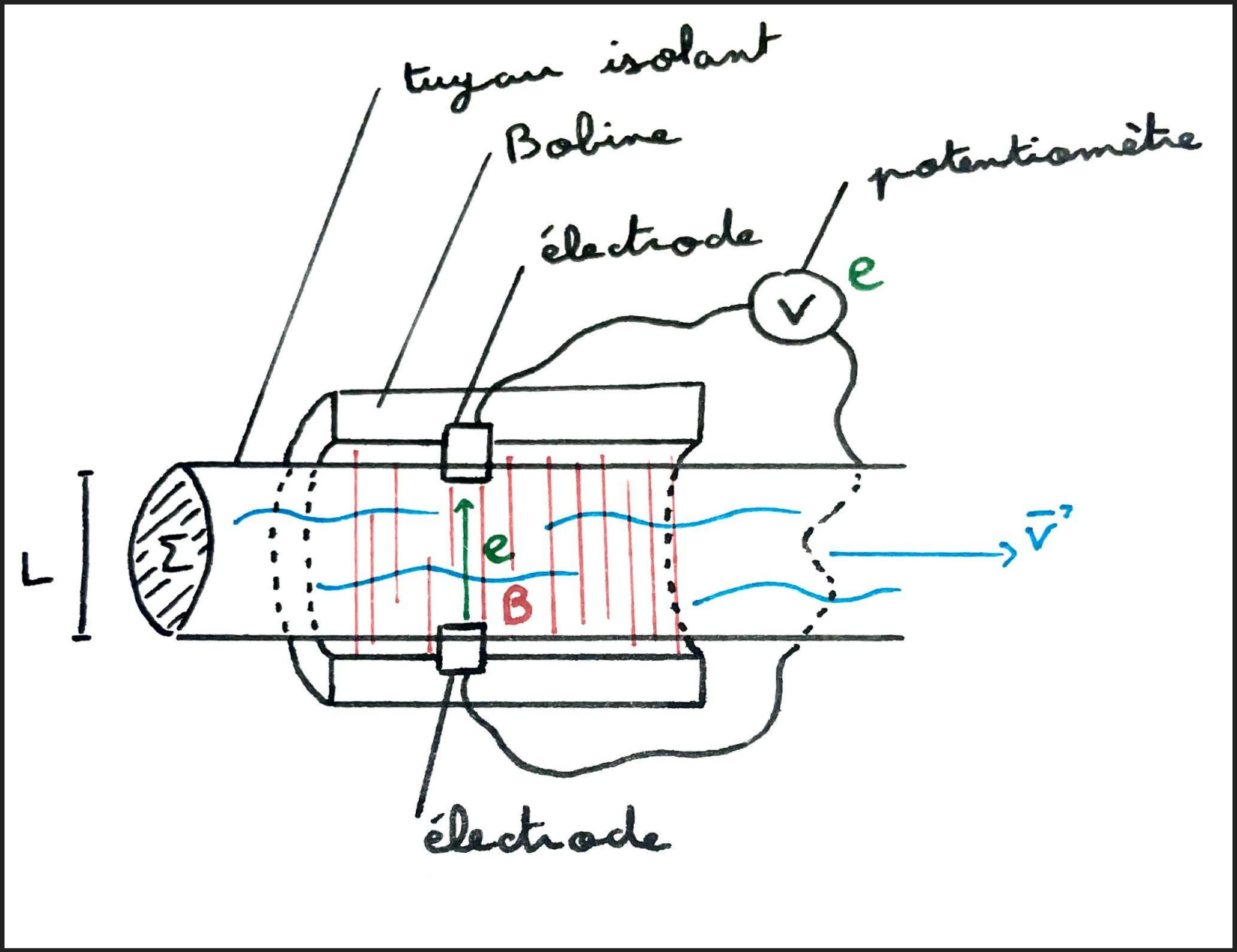


Schéma de l'interaction ski neige



débitmètre électromagnétique circulaire

comment fonctionne un débitmètre linéaire

La mesure se fonde sur la loi de Lenz-Faraday. Pour une surface Σ de contour L, un champ constant B perpendiculaire à l'écoulement de l'eau, elle s'énonce :

$$e = L\vec{v}.\vec{B}$$

Preuve :

Loi de Lenz-Faraday :

or,

$$e = -\frac{d}{dt}\Phi$$

$$\Phi = \iint_{\Sigma} \vec{B}.\vec{dS}$$

D’après la règle d’intégration de Leibniz dans un espace à trois dimension dépendant du temps,

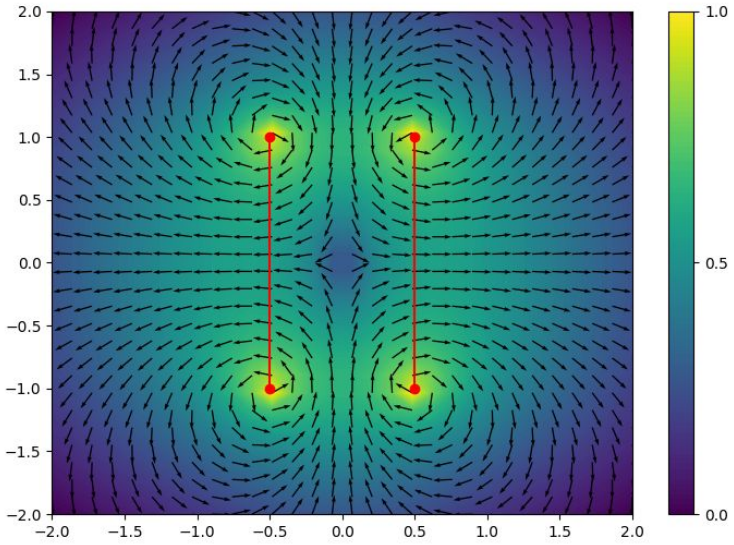
$$\frac{d}{dt}\iint_{\Sigma}\vec{B}.d\vec{S} = \iint_{\Sigma}\frac{\partial}{\partial t}\vec{B}.d\vec{S} - \oint_{\partial\Sigma}\vec{v}\times\vec{B}.\vec{dl}$$

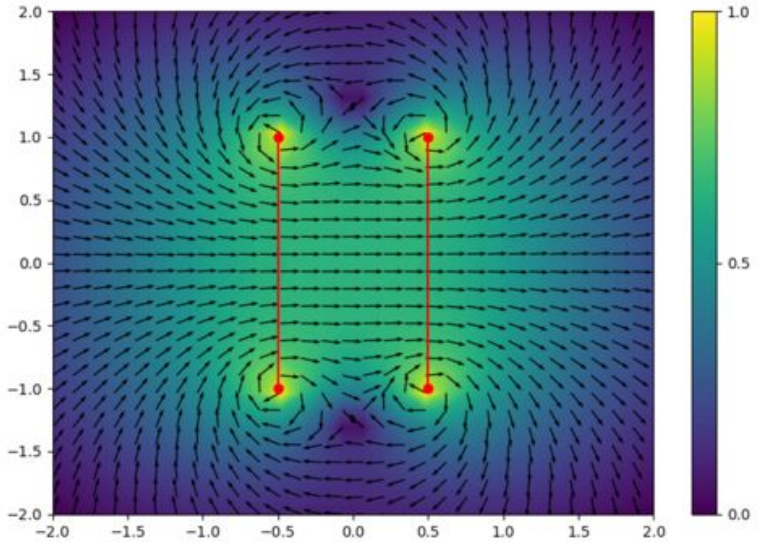
or, le champ B est indépendant du temps donc,

Ainsi, on obtient,

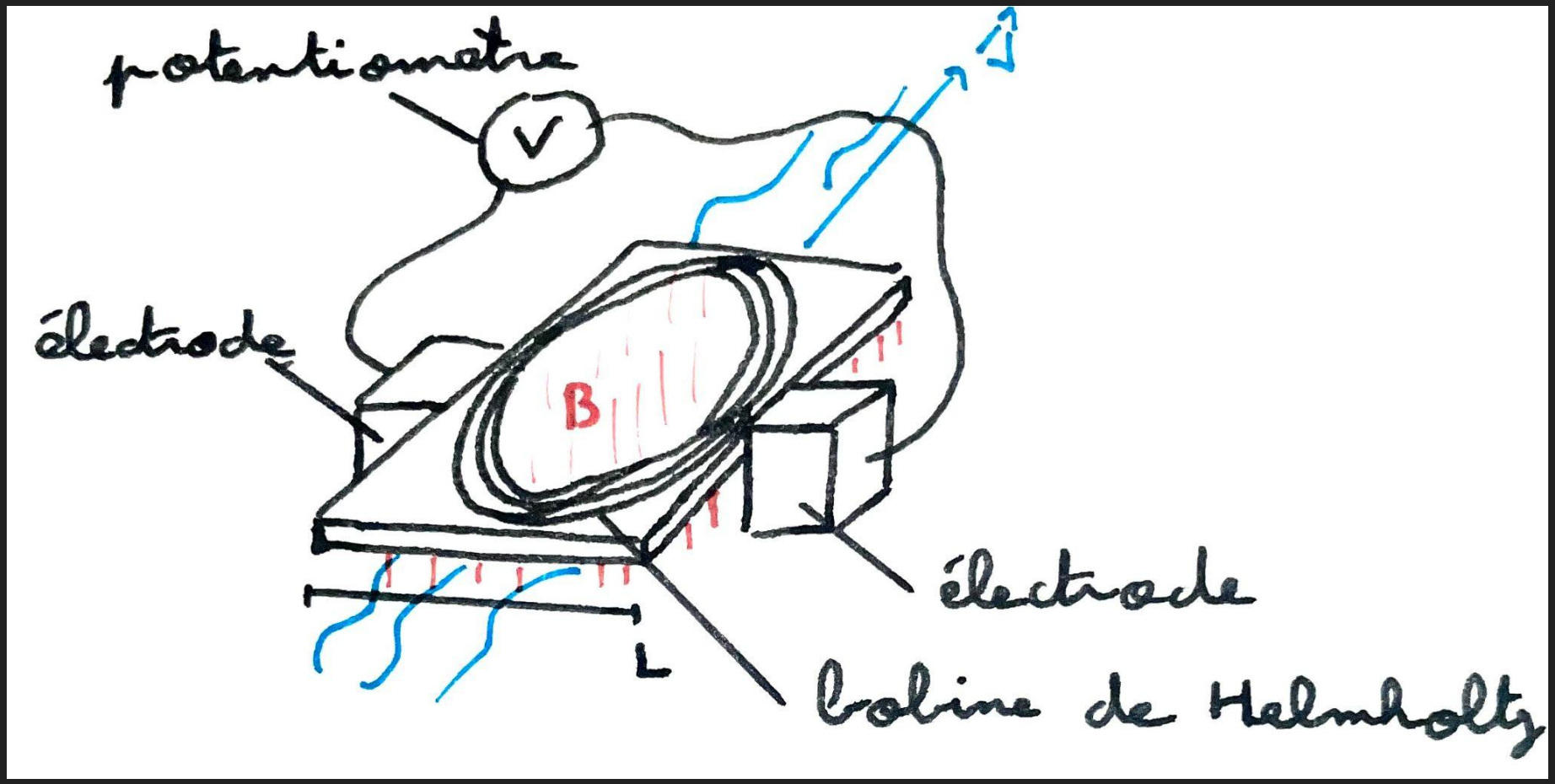
$$e = \oint_{\partial\Sigma}\vec{v}.\vec{B}.\vec{dl}$$

$$e = LvB$$





Graphe du champ magnétique créé par une bobine de Helmholtz
Source : wikipédia



Prototype de débitmètre électromagnétique linéaire

Choix de B

La conductivité d'une eau minérale, comme c'est le cas ici est environ

σ = 2378 µS.cm^{-1}

or,

$$\sigma = \frac{IL}{eA}$$

avec A la section de contacte entre l'électrode et l'eau.

On a établi qu'il y a 0.4cm d'eau et une électrode fait environ 0.5 cm de large donc A = 0.2cm²

On peut raisonnablement prendre L = 6 cm et I = 10^{-3} A

$$B = \frac{I}{v\sigma A}$$

Il nous faut donc une bobine de Helmholtz de 1.3x10^{-3} Tesla

Définition

example.js

```
 struct point_s
{
    double val;
    double tmp;
};
typedef struct point_s point;

struct donnees_s
{
    double amp;
    double freq;
};
typedef struct donnees_s donnees;

struct signal_s
{
    point* amplitude;
    int n;
};
typedef struct signal_s signal;
```

csv -> signal

example.js

```c
/* csv_to_signal lit un signal du format CSV et le transforme en un
signal du type enregistrement "signal" */
signal csv_to_signal (const char* fichier)
{
    FILE* fptr = fopen(fichier, "r");
    int n = nb_ligne(fichier);
    point* amplitude = malloc(n*sizeof(point));

    // sauter la description du csv (aller ligne 19)
    int nligne = 0;
    int c;
    while (nligne < 18) {
        if(c=fgetc(fptr) == '\n') {
            nligne += 1;
        }
    }
```

example.js

```c
    // lecture des valeur
    float tmp, val;
    char chaine[10000];
    for (int i = 0; i<n && chaine =/= NULL; i+=1) {
        fscanf(fptr, ",,,%f,   %f,", &tmp, &val);
        amplitude[i].val = val;
        amplitude[i].tmp = tmp;
        fgets(chaine, 10000, fptr);
        //printf("lecture de la ligne %d : tmp = %f val = %f\n", i, tmp, val);
    }

    // on ferme le fichier
    fflush(fptr);
    fclose(fptr);

    signal y = {.amplitude = amplitude, .n=n-18 };
    return y;
}
```

Traitement du signal

example.js

```
// traite_signal renvoie un tableau de données [amplitude_moyenne ; frequence]
double* traite_signal (signal* s, int k)
{
    double amplitude, periode, frequence, delta_t;

    double moy_signal = 0, moy_amplitude = 0;
    double moy_amp_pos = 0, moy_amp_neg = 0;

    // nombre de point du signal
    int n = (*s).n;

    // tableau glissant de k point pour "lisser le signal"
    double tmp[k];
    int conteur_k = 0;

    // nombre de maximum/ minimum locaux (nombre de "période" du signal)
    int n_moy_pos = 0, n_moy_neg = 0;

    for (int i=0 ; i<n ; i+=1){

        double yi = (*s).amplitude[i].val;

        moy_signal += yi;

        tmp[conteur_k] = yi;
        conteur_k = (conteur_k + 1) %k;

        // calcul des maximum et minimum "très locaux"
        double max_i = 0, min_i = 0;
        for(int i=0 ; i<k ; i+=1)
        {
            if (tmp[i] > max_i) max_i = tmp[i];
            if (min_i > tmp[i]) min_i = tmp[i];
        }

        // repérage des sommet du signal
        if ((*s).amplitude[i-1].val < yi && (*s).amplitude[i+1].val < yi && yi ⩾ max_i )
        {
            moy_amp_pos += val_abs(yi);
            n_moy_pos += 1;
        }
        if (yi < (*s).amplitude[i-1].val && yi < (*s).amplitude[i+1].val && yi ⩽ min_i )
        {
            moy_amp_neg += val_abs(yi);
            n_moy_neg += 1;
        }
    }

    // calcul des moyennes
    moy_signal = moy_signal/(*s).n;
    moy_amp_neg = moy_amp_neg/n_moy_neg;
    moy_amp_pos = moy_amp_pos/n_moy_pos;
    moy_amplitude = (moy_amp_pos + moy_amp_neg)/2;

    // calcul des grandeurs phisiques
    amplitude = moy_amplitude - moy_signal;
    delta_t = (*s).amplitude[n].tmp - (*s).amplitude[0].tmp;
    periode = 2*delta_t / (n_moy_pos + n_moy_neg);
    frequence = periode;

    // renvoie des données
    double* donnees = malloc(2*sizeof (double));
    donnees[0] = amplitude;
    donnees[1] = frequence;
    return donnees;
}
```

k_nn

example.js

```
bool k_nn (signal* s, const char* fichier)
{
    double* donnees_s = traite_signal(s, 1);

    // lecture de la base de données
    int n_d = 7;
    donnees* donnees_aval = malloc(n_d+1*sizeof(donnees));

    FILE* fptr = fopen(fichier, "r");

    // obtention des points
    donnees_aval[0].amp = donnees_s[0];
    donnees_aval[0].freq = donnees_s[1];

    for (int i=1 ; i<n_d+1 ; i+=1) fscanf(fptr, "%lf %lf", &donnees_aval[i].amp, &donnees_aval[i].freq);

```

example.js

```
// calcule du barycentre du cluster d'avalanche
    double barycentre_amp = 0, barycentre_freq = 0;
    for (int i=1 ; i<n_d+1 ; i+=1) {
        barycentre_amp = barycentre_amp + donnees_aval[i].amp;
        barycentre_freq = barycentre_freq + donnees_aval[i].freq;
    }
    barycentre_amp = barycentre_amp/n_d;
    barycentre_freq = barycentre_freq/n_d;
    donnees bar = {.amp = barycentre_amp, .freq = barycentre_freq};

    // si la distance du signal s au barycentre est "courte" alors on peut conclure que c'est celui d'une avalanche
    if (dist(donnees_aval[0], bar) < 155) {
        // on ferme le fichier et free
        free(donnees_aval);
        return true;
    }
    else {
        // on ferme le fichier et free
        free(donnees_aval);
        return false;
    }
}
```