Code ton synthé JavaScript

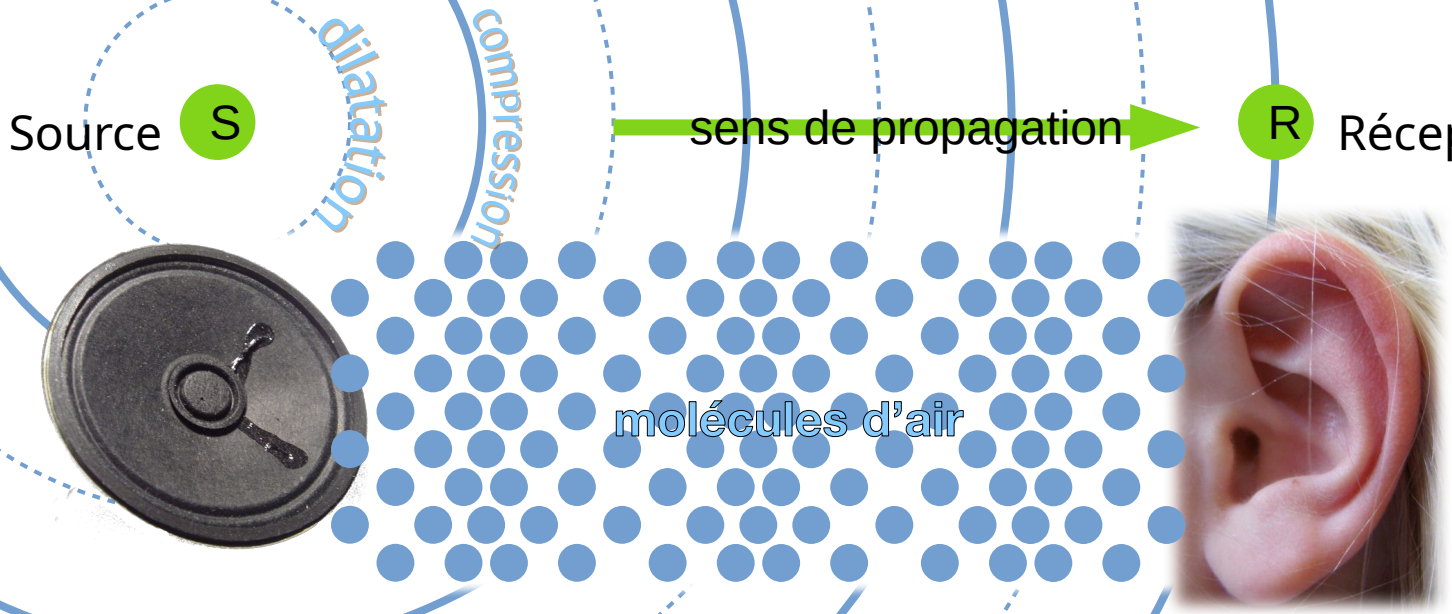
Introduction ludique à la synthèse audio numérique

par Emmanuel RK

Qu’est-ce que le son ?

Le son est la sensation auditive que produit la vibration mécanique de l’air.

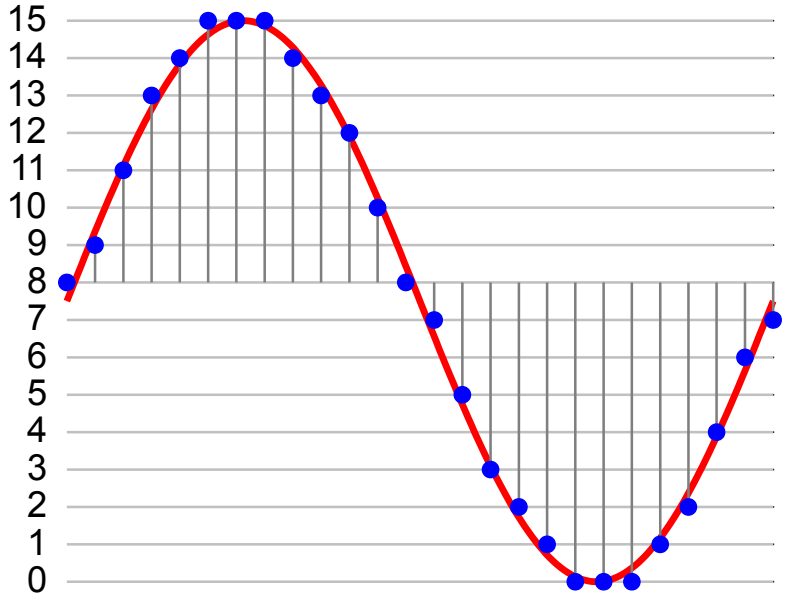
L’onde sonore se propage avec transport d’énergie mais sans déplacement de matière.



Codage du son

Pour numériser un son, le signal électrique provenant par exemple d'un micro est d'abord échantillonné (on prend N mesures par seconde, ex : 44100 pour un CD) puis quantifié, c'est-à-dire représenté avec un nombre fini de valeurs.

C'est ce qu'on appelle le son PCM pour *Pulse Code Modulation*.



Synthèse audio numérique

Pour la synthèse audio numérique, on effectue l’opération inverse : on calcule la valeur des échantillons afin de produire un son.

Pour la suite on va s’appuyer sur une micro webapp permettant de coder son synthé :

https://shadocko.github.io/synthtoy/

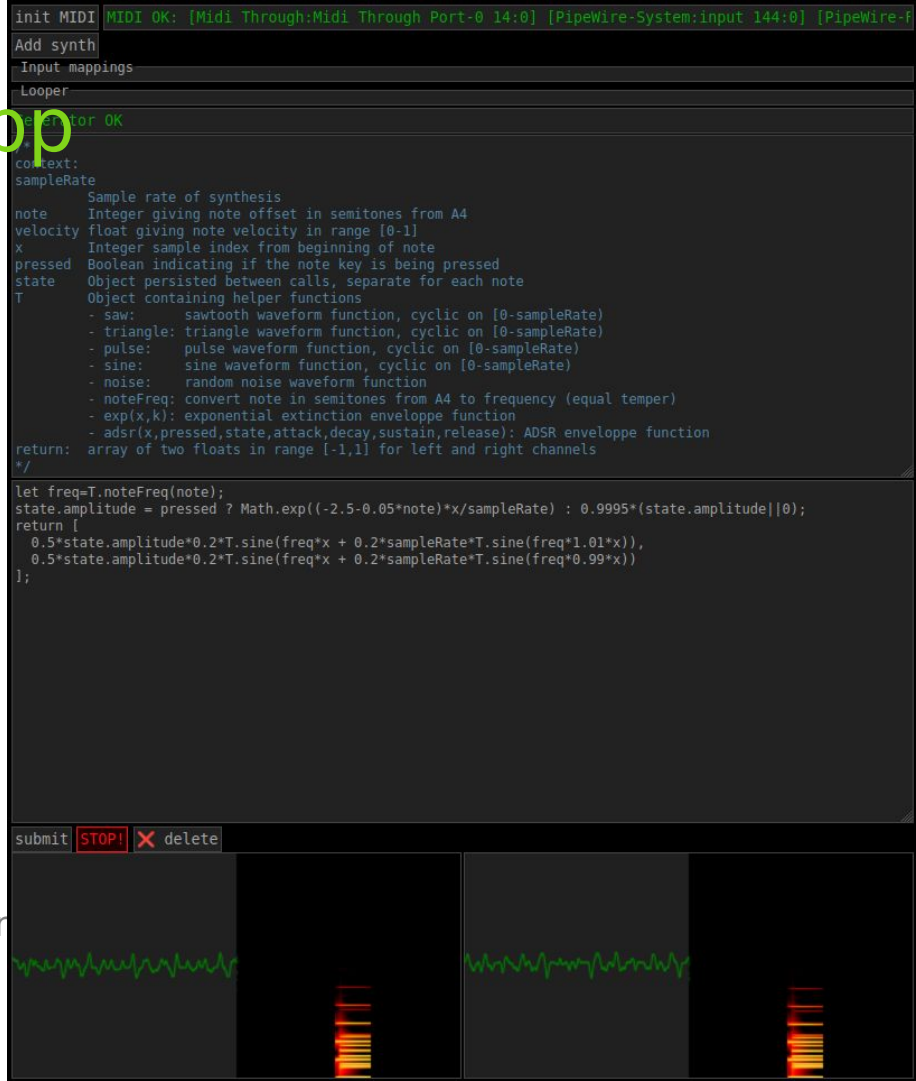
Présentation de la webapp

La *webapp* propose d'utiliser le clavier de l'ordinateur ou un périphérique MIDI comme clavier musical.

Elle permet d'écrire le corps d'une fonction JavaScript appelée pour chaque échantillon, dès-lors qu'une note est jouée.

La fonction doit retourner un tableau de deux valeurs comprises entre -1 et 1 : une pour le canal gauche et une pour le canal droit.

Remarque : ne pas s'attarder sur le code de la webapp qui n'est pas des plus beaux à voir, il a été écrit sur un coin de table en mode "code jetable" pour le fun...



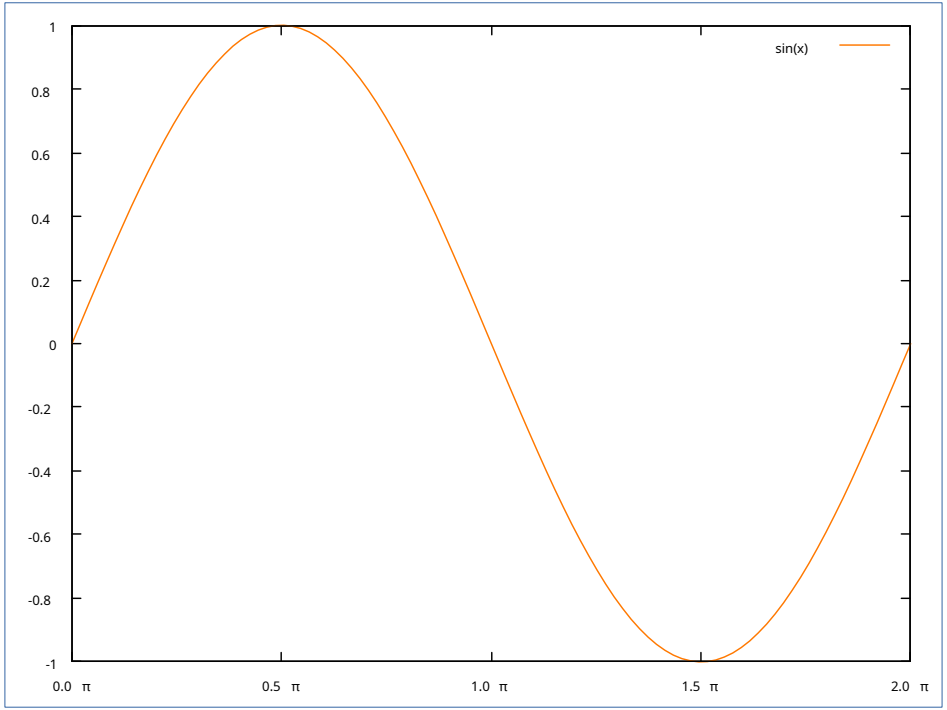
Hello World! - premier son synthétique

On va chercher à produire une sinusoïde pure à 440 Hz (la note LA) lorsqu'une touche est enfoncée.

Le taux d'échantillonnage cible est transmis à la fonction dans le paramètre `sampleRate`, `x` désigne le numéro de l'échantillon (0 à l'appui sur la touche, puis 1, 2, 3, etc.) et `pressed` indique l'état pressé ou non de la touche.

La fonction JavaScript `Math.sin()` est cyclique avec une période de 2π.

La formule est donc : $\sin\left(\frac{440 \times 2\pi x}{sampleRate}\right)$



Tentons cela en JavaScript...

Soluce et boîte à outils

Voici une solution "simple" :

```
let y = pressed ? Math.sin(440*x*2*Math.PI/sampleRate) : 0;
return [y, y];
```

Cependant la fonction reçoit aussi un paramètre "boîte à outils" `T` qui propose une fonction sinusoïde avec une période de `sampleRate`, ce qui simplifie encore davantage l'écriture :

```
let y = pressed ? T.sine(440*x) : 0;
return [y, y];
```

Jouer une gamme

La note que l'on perçoit est directement liée à la fréquence principale du signal sonore, appelée fondamentale.

En multipliant ou en divisant par deux la fréquence fondamentale, on passe à l'octave supérieur ou inférieur.

Notre fonction de synthétiseur logiciel reçoit la note à jouer via le paramètre `note`, exprimé en demi-tons depuis le LA 440Hz.

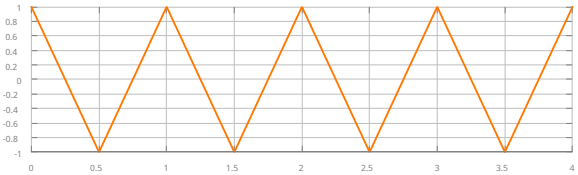
La musique occidentale à laquelle nous sommes habitués utilise majoritairement la gamme dodécaphonique (à 12 demi-tons). Pour calculer la fréquence fondamentale de la note dans cette gamme, avec un tempérament égal, on appliquera la formule $440 \times 2^{(note/12)}$, directement implémentée dans la fonction `T.noteFreq(note)` de la boîte à outils.

Faisons l'expérience avec notre synthé logiciel !

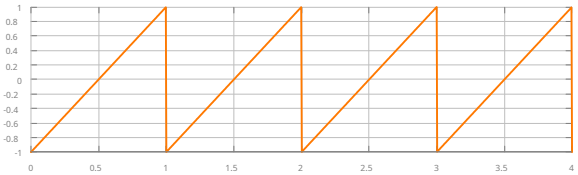
Remarque : on peut aussi s'amuser avec d'autres gammes musicales.

D’autres formes d’onde

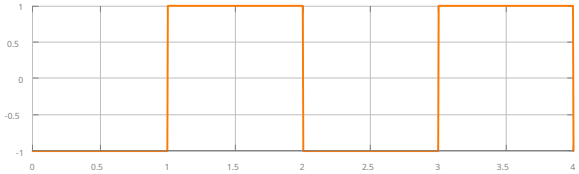
L’onde triangulaire : `T.triangle()`



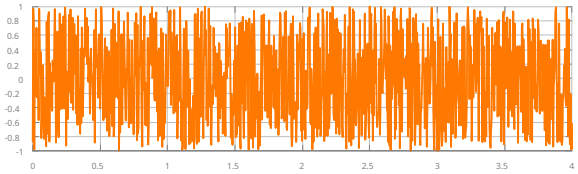
L’onde en dents de scie : `T.saw()`



L’onde en créneaux : `T.pulse()`



Le bruit blanc : `T.noise()`



La boîte à outils comporte d’autres formes d’onde prédéfinies courantes en synthèse audio.
Elles ont toutes une période de `sampleRate`.

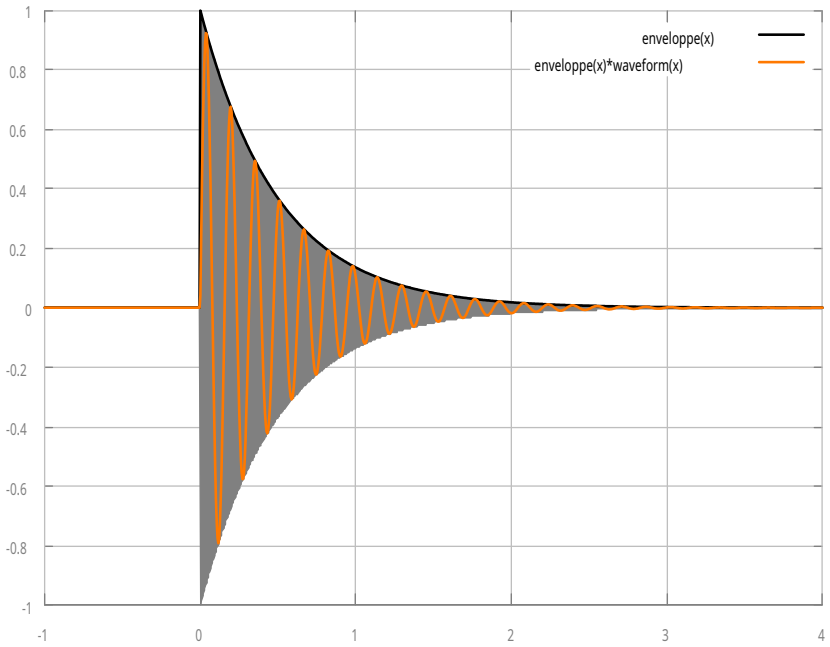
Blip ! Blop ! Pshh ! Flashback années 80 garanti :-)

Forme d’onde et enveloppe

Jusque-là, nous avons déclenché le son en mode "tout ou rien" avec un résultat tout juste digne d’un jeu électronique d’autrefois.

En synthèse audio, on va souvent avoir recours à des fonctions d’enveloppe qui varient plus lentement dans le temps pour moduler d’autres paramètres du son, par exemple l’amplitude de la forme d’onde.

Que se passe-t-il si l’on multiplie nos formes d’onde précédentes par $e^{(-2*x/sampleRate)}$? (`Math.exp(…)`)



Enveloppe ADSR

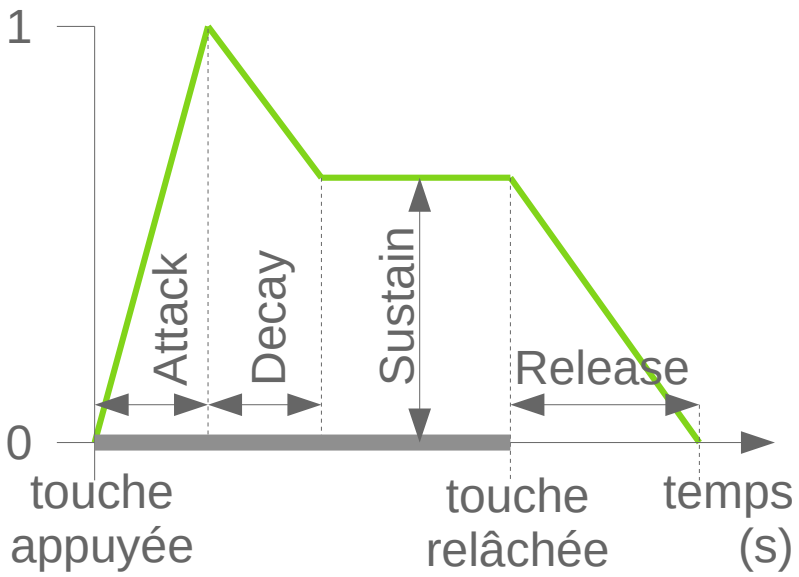
L’enveloppe ADSR pour Attack, Decay, Sustain, Release est l’une des plus utilisées.

C’est une fonction linéaire par morceaux. Saurez-vous la coder ?

Astuce : le paramètre `state` est un objet qui permet de conserver un état entre les appels à la fonction.

Sinon il y a :

```
T.adsr(x,pressed,state,a,d,s,r)
```



Synthèse additive

La synthèse additive consiste à construire des sons par simple superposition de formes d'ondes primitives.

En additionnant des harmoniques pondérées au son principal (c'est à dire des sons dont la fréquence est un multiple entier de la fondamentale), on peut par exemple construire un son convainquant d'orgue, de cloche ou de marimba.

```
// Enveloppe Attack-Release simple
if (state.waspressed && !pressed)
  state.release = x;
state.waspressed = pressed;
let amplitude = pressed ?
  Math.min(1, x*20/sampleRate) :
  Math.max(0, 1-(x-state.release)*20/sampleRate);
// Accumulation d'harmoniques pondérées
const h = [1, 3, 4, 8, 16];
const w = [1., 0.2, 0.5, 0.2, 0.1];
let y = 0.;
let freq=T.noteFreq(note);
for(let i in h) {
  y += w[i] * T.sine(x*freq*h[i]);
}
// Prise en compte de vélocité et enveloppe
y *= 0.2*velocity*amplitude;
return [ y, y ];
```

Modulation de fréquence

La synthèse audio par modulation de fréquence ou synthèse FM consiste à altérer la fréquence du son dans le temps, en fonction d'un autre signal lui-aussi périodique. Pour des formes d'onde $signal_{porteuse}$ et $signal_{modulation}$ toutes deux de période N, on peut exprimer le signal modulé comme suit (temps t en secondes) :

$$signal_{FM}(t) = signal_{porteuse}(f_{porteuse} . t . N + N . poids_{modulation} . signal_{modulation}(f_{modulation} . t . N))$$

NB : dans la webapp, le produit $t.N$ correspond simplement au numéro d'échantillon x.

En utilisant une fréquence de modulation $f_{modulation}$ proche de la porteuse $f_{porteuse}$ (typiquement un multiple proche de 1 de cette dernière), cette technique permet d'obtenir des sons très riches en harmoniques avec seulement deux oscillateurs.

Il est également possible de chaîner les modulations, c'est-à-dire d'utiliser un signal FM pour moduler la fréquence d'un autre signal.

La synthèse FM est omniprésente dans la musique électronique.

Un peu de stéréo s’il-vous-plaît ?

Jusque-là, on s’est contenté de générer le même signal pour les canaux gauche et droite. Il est possible de donner l’impression que le son provient d’une direction donnée en déphasant légèrement les deux signaux.

Astuce : on obtient un effet "son-spacial-pour-pas-cher™" en utilisant de la synthèse FM avec des fréquences de modulation différentes pour les deux canaux.

```
// Enveloppe Attack-Release lente
if (state.pressed && !pressed)
  state.release = x;
state.pressed = pressed;
let amplitude = pressed ?
  Math.min(1, x*2/sampleRate) :
  Math.max(0, 1-(x-state.release)/sampleRate);
// Prise en compte de la vélocité
amplitude *= 0.4*velocity;
// Synthèse FM + spacialisation-du-pauvre
let freq=T.noteFreq(note);
return [
  amplitude * T.triangle(
    freq*x + sampleRate*.5*T.sine(freq*1.01*x) ),
  amplitude * T.triangle(
    freq*x + sampleRate*.5*T.sine(freq/1.01*x) )
];
```

LFO

Outre le nom d'un groupe d'électro des années 90, c'est surtout l'acronyme de Low Frequency Oscillator.

L'idée est cette fois de moduler des propriétés du son en fonction d'un signal basse fréquence, inaudible en tant que tel (<20Hz) : amplitude, phase, poids d'une modulation de fréquence…

Vibrato, tremolo, *flange* (effet "cassette restée au soleil"), distorsion, les possibilités sont nombreuses.

```
let amplitude =
T.adsr(x,pressed,state,0.05,0,1,0.1)
                * 0.2 * velocity;
let freq = T.noteFreq(note);
let lfo1 = T.sine(1.3*x);
let lfo2 = T.sine(0.7*x);
x += 0.002 * sampleRate * (0.5*lfo1 + lfo2);
return [
  amplitude * T.triangle(
    freq*x + 0.1*sampleRate*T.sine(freq*x*1.01) ),
  amplitude * T.triangle(
    freq*x + 0.1*sampleRate*T.sine(freq*x/1.01) )
];
```

Sons arpégés

Les micro-ordinateurs et consoles de jeu des années 80 avaient un nombre très limité de voix audio.

Une technique courante pour donner l'impression d'avoir plus de voix était d'alterner rapidement entre plusieurs fréquences, par exemple les 3 notes d'un accord.

On peut reproduire cet effet en synthèse audio numérique mais il faut veiller à la continuité du signal.

« Les années 80 ont appelé, elles demandent à récupérer leur NES… »

```
let freq=T.noteFreq(note);
let c = Math.floor(50*x/sampleRate) % 3;
switch(c) {
case 0:
  // On utilise la fréquence telle-quelle
  break;
case 1:
  // On utilise la tierce majeure
  freq *= Math.pow(2., 4./12.);
  break;
case 2:
  // On utilise la quinte
  freq *= Math.pow(2., 7./12.);
}
let amplitude =
  0.3*T.adsr(x,pressed,state,0.05,2,0,0.1);
let fm1=freq*1.01;
let fm2=freq/1.01;
// On maintient la continuité de la phase
state.phi1 = (state.phi1|0) + freq
  + 0.2*sampleRate*T.sine(fm1*x)/fm1;
state.phi2 = (state.phi2|0) + freq
  + 0.2*sampleRate*T.sine(fm2*x)/fm2;
return [
  amplitude*T.triangle(state.phi1),
  amplitude*T.triangle(state.phi2)
];
```

Merci

Y’en a un peu plus, je vous l’mets quand-même ?

```
// E-piano
let freq=T.noteFreq(note);
let amplitude = 0.2 * velocity
  * T.adsr(x,pressed,state,.001,0,1,0.25)
  * T.exp(x, 2.5+0.05*note);
return [
  amplitude * T.sine(
    freq*x + 0.2*sampleRate*T.sine(freq*1.01*x)),
  amplitude * T.sine(
    freq*x + 0.2*sampleRate*T.sine(freq*0.99*x))
];
```

```
// Cloches
const H = [1, 2, 3, 4, 5];
const W = [ 0.5, 0.1, 0.5, 0.1, 0.1 ];
const E = [ 5, 10., 1., 1., .5 ];
let y = 0.;
let freq = T.noteFreq(note);
for( let j in H )
  y += W[j] * T.sine(x*freq*H[j])
       * Math.exp(-E[j]*x/sampleRate);
y *= 0.4*velocity;
return [y, y];
```

```
// Marimba
const H = [1, 2, 3, 4];
const W = [1, 0.1, 0.5, 0.1];
const E = [8., 12., 16., 24.];
const A = [500.,500.,500.,1000.];
let y = 0.;
let freq = T.noteFreq(note);
for( let j in H )
  y += W[j] * T.sine(x*freq*H[j])
       * Math.min( Math.exp(-E[j]*x/sampleRate),
                   A[j]*x/sampleRate );
y *= 0.4*velocity;
return [y, y];
```