

TP JS 3 - Courbes

D compressez l'archive d pos e sur Moodle pour ce TP. Le dossier r sultant contient diff rents fichiers   r utiliser ou   compl ter. Pensez   consulter le site [MDN](#).

On propose une page web permettant de repr senter graphiquement diff rentes fonctions math matiques (voir Figure 1). L'impl mentation JS s'appuie sur les canevas HTML et l'API [Canvas](#) en se limitant aux fonctionnalit s permettant le [trac  de lignes](#).

La page permet au visiteur de

- s lectionner et param trer des fonctions,
- les  chantillonner sur un intervalle de valeurs,
- dessiner chaque courbe en reliant les ordonn es des points d'abscisses successives de l' chantillon,
- animer chaque courbe segment par segment selon une fr quence choisie.

La trace de l'ex cution JS est consign e dans les diff rents onglets de la console : objets utilis s pour le trac  et l'historisation (onglet [Journaux](#)), points hors-limites ([Avertissements](#)), lev es d'exceptions ([Erreurs](#)),  chantillons g n r s et autres variables ([D bogueur](#)).

La page comprend :

- un formulaire scind  en trois panneaux,
- un canevas sur lequel sont dessin s les courbes,
- un tableau faisant office d'historique (log) et listant toutes les fonctions dessin es.

Le panneau "Rep re" sert   ajuster les dimensions du rep re (coordonn es maximum) dans lequel s'inscriront les courbes et, optionnellement,   surimposer une grille. Le panneau "Echantillonnage" sert   ajuster la taille des  chantillons (nombre de points)   g n rer pour chaque fonction ainsi que la fr quence du tra age. Le panneau "Fonctions" sert   param trer les fonctions et   d clencher leur trac  en cochant la case correspondante.

Chaque courbe est dessin e avec une couleur tir e al atoirement et l'expression math matique de la fonction est ajout e au tableau. Si la fr quence f choisie est nulle (valeur par d faut), les courbes seront dessin es l'une apr s l'autre sans d lai d'affichage entre segments successifs. Sinon, les courbes seront dessin es de mani re concurrente (en "parall le") avec un d lai de f dixi mes de secondes entre segments successifs (voir l'instantan  en Figure 2).

Ce [d monstrateur](#) vous aidera   visualiser ce qui est attendu.

Le dossier d compress  contient les fichiers suivants :

- **courbes.html** : le fichier HTML de la page web.
- **courbes.css** : la feuille de styles.
- **courbes.js** : le fichier principal JS.
- ** chantillonneur.js** : le module d' chantillonnage.
- **traceur.js** : le module de dessin.
- ** chantillonneur_proto.js** : module obfusqu  pour l' chantillonnage.
- **traceur_proto.js** : module obfusqu  pour le dessin.

Le fichier principal **courbes.js** implémente les fonctionnalités suivantes :

1. construction d'un traceur pour dessiner repère, courbes et grille,
2. tracé par défaut de la fonction identité par l'IIFE **test**,
3. appel de la fonction **tracer** qui trace une courbe au cochage d'une case,
4. appel de la fonction **regénérerTracés** qui régénère canevas et courbes à chaque clic sur **REGENERER**.

Le traceur est une instance de la **classe** **Traceur** construite à partir du **canevas** de la page, d'un contexte 2D, des marges internes du canevas qui délimiteront le cadre (20 pixels en horizontal/vertical), et des valeurs maximum autorisées en abscisse et ordonnée pour les fonctions. L'IIFE **test** fournit la recette pour tracer et journaliser une fonction (ici, la fonction $f(x) = x$). Elle délègue ces opérations au traceur par invocation de la méthode **dessiner** dont l'un des arguments est le descripteur de la fonction qui contient notamment son implémentation JS sous forme de callback. Cette méthode délègue l'échantillonnage à une instance de la classe **Echantillon** et le tracé à une instance de **Traceur** en invoquant la méthode **tracer**.

Le code qui vous est fourni est fonctionnel grâce à l'import de **modules** obfusqués. Vous devrez écraser différentes méthodes et fonctions qui font appel à ces modules pour répondre aux questions.

Exercice 1. Echantillonnage

Le module **Echantillonneur** exporte la classe **Echantillon**. Chaque instance d'**Echantillon** modélise un échantillon de fonction et contient les champs suivants :

- la fonction à échantillonner sous forme de callback **f**,
- la taille **n** de l'échantillon,
- l'intervalle de valeurs **minMaxX** sur lequel est échantillonnée la fonction,
- une méthode statique **abscisse(k,n,minMaxX)** qui calcule le **k**-ième point dans l'intervalle **minMaxX** sous-divisé en **n-1** intervalles de même taille,
- l'échantillon de points résultant, nommé **points**, stocké sous la forme d'un tableau de **n** paires $[[x, f(x)] \mid i=1..n, x=abscisse(i,n,minMaxX)]$.

1. Réimplémentez la propriété **points**.

Exercice 2. Tracé et journalisation

Le tracé d'une fonction **f** s'effectue en convertissant chaque point **(x,f(x))** d'un échantillon par une paire de coordonnées (appelée "transformé" du point) qui sont exprimées en pixels relativement au coin supérieur gauche du canevas (voir l'API **Canvas**).

Les instances de la classe **Traceur** implémentent des conversions obéissant aux règles suivantes :

- les points autorisés **(x,f(x))** appartiennent au domaine $D = [-maxXY.X, +maxXY.X] \times [-maxXY.Y, +maxXY.Y]$ où **maxXY** est donné,
- les transformés autorisés appartiennent au rectangle **R** délimité par les marges internes du canevas,
- le transformé du point **0=(0,0)** apparaît au centre de **R** et correspond à l'origine du repère,
- le transformé d'un point **(x,y)** est obtenu en redimensionnant **x** et **y** du fait du changement d'échelle entre **D** et **R** puis par décalage relativement au transformé du point **0**,
- tout point en dehors du domaine **D** est omis du tracé,
- tout transformé en dehors du rectangle **R** est omis du tracé,
- toute omission donne lieu à un avertissement en console.

Un objet **Traceur** encapsule :

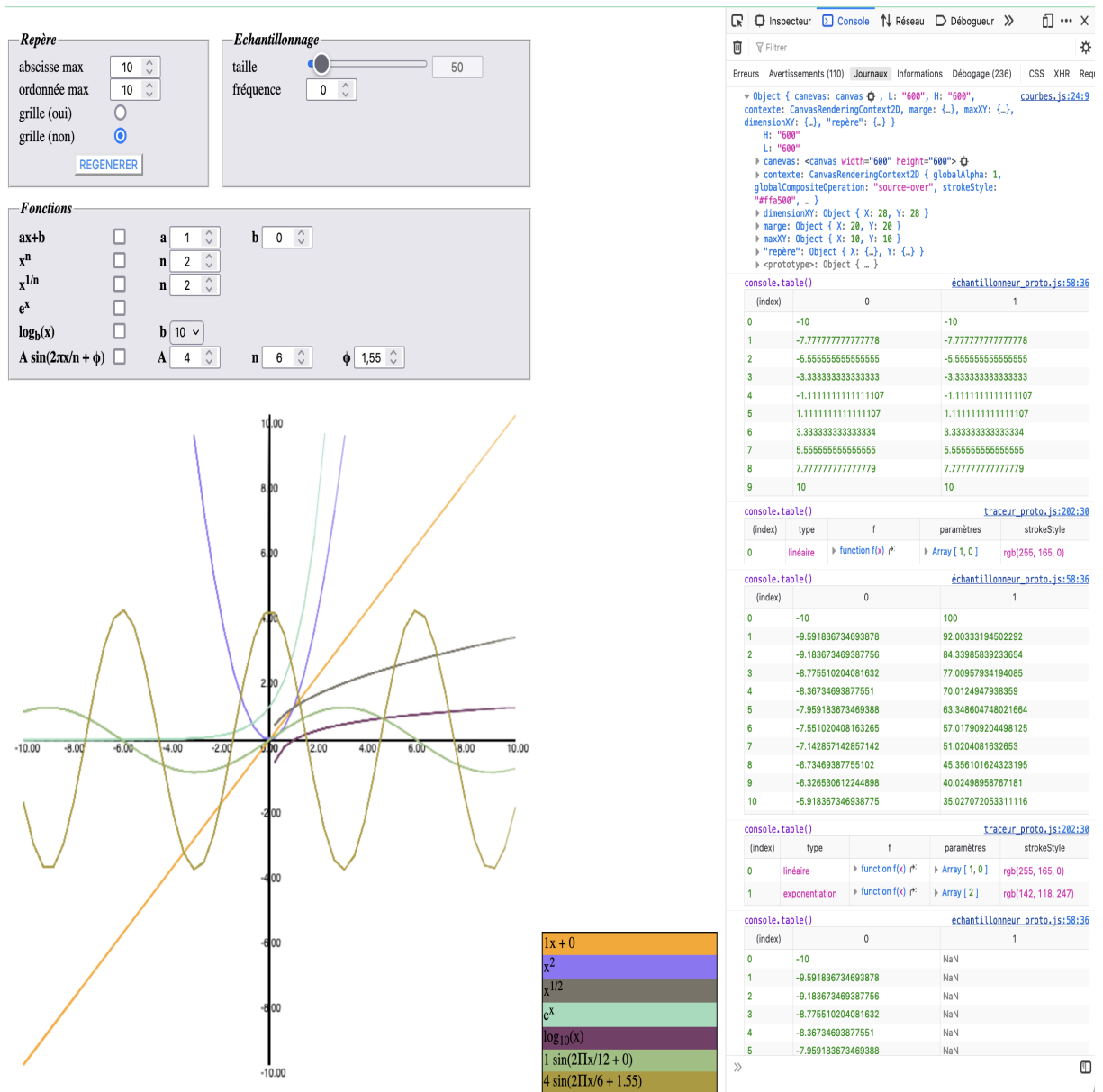
- un canevas **canevas** sur lequel dessiner,
- sa largeur **L** définie par l'attribut **HTML width**, sa hauteur **L** définie par l'attribut **HTML height**, et ses marges internes **marge** qui déterminent conjointement les dimensions du rectangle **R**,
- son contexte 2D **contexte**,
- les valeurs maximum **maxXY** déterminant le domaine **D**,

- les coefficients de redimensionnement **rapportXY** basés sur les ratios entre R et D,
 - les transformés des points extrêmes des axes du repère et de son centre (**repère**).
1. Réimplémentez la méthode **transformer(x,y)** qui calcule et renvoie le transformé (u,v) d'un point (x,y) au format {"X": u, "Y": v}. La méthode renvoie {"X": false, "Y": false} pour tout point ou tout transformé hors-limites. Par exemple, un point est hors-limite si la fonction est évaluée en dehors de son domaine de valeurs (valeur NaN) ou si l'évaluation donne lieu à un débordement (valeurs +/-Infinity).
 2. Réimplémentez la méthode **tracerGrille** qui dessine la grille correspondant aux graduations des axes du repère. Inspirez-vous de la méthode **tracerRepère** qui dessine le repère. Les axes de la grille sont tracés en **pointillés** et en noir (voir Figure 2).
 3. Réimplémentez la méthode **tracer(P,style)** qui dessine la courbe de l'échantillon de points P en utilisant la couleur CSS **style**. La méthode omet le tracé des segments reliant tout point de valeur {"X": false, "Y": false} et émet un avertissement le cas échéant. La méthode lève une exception si les abscisses des points consécutifs du tableau P ne sont pas ordonnées, cad. ne vérifient pas $P[i].X < P[i+1].X$ ($1 \leq i < n$).
 4. Réimplémentez la méthode **dessiner(n, meta, log)** qui (1) génère un échantillon de taille n pour la fonction décrite par **meta**, (2) trace sa courbe, et (3) journalise son expression dans le tableau **log** et l'affiche dans le tableau HTML. L'échantillonnage s'appuie sur la classe **Echantillon**. Le tracé utilise la méthode **tracer(P,style)** en considérant deux cas de figure :
 1. A fréquence nulle, le tracé s'effectue de manière synchrone.
 2. A fréquence non nulle, le tracé s'effectue segment par segment par appel périodique à **tracer(P,style)**. Pour ce faire, utilisez **setInterval** en veillant à lier "l'objet **this**" à ces appels (voir **Le problème de this**)!

La journalisation et l'affichage dans le tableau HTML suppose de construire l'expression algébrique de la fonction à partir de son nom et des paramètres stockés dans le descripteur **meta** : privilégiez les **littéraux de gabarits** (*template literals*).

Exercice 3. Ecouteurs

1. Réimplémentez la fonction **tracer(traceur,log)** dans **courbes.js** qui trace la courbe d'une fonction *f* lorsque la case correspondante est cochée. La fonction procède en 3 étapes :
 1. Elle construit le descripteur de *f* à partir des paramètres renseignés dans le formulaire. Consultez le descripteur **meta_f** initialisé dans l'IIFE **test** pour le format attendu. La fonction *f* sera à implémenter à l'aide des méthodes et constantes de l'objet **Math**. La couleur CSS sera aléatoirement générée sous forme de chaîne de caractères du type "rgb(12,255,0)" par appel à la fonction **rgb** qui est à implémenter.
 2. Elle appelle **traceur.dessiner(n, meta_f, log)** pour le tracé.
 3. Elle laisse la case cochée pendant 2 secondes avant de la décocher.
2. Réimplémentez la fonction **regénérerTracés(log)** qui, à chaque clic sur **REGENERER**, régénère le canevas et les courbes tracées. La fonction procède en 4 étapes :
 1. Suppression et remplacement du canevas existant par un canevas de même taille.
 2. Construction d'une instance de **Traceur** pour ce canevas selon les coordonnées maximum renseignées dans le formulaire et avec des marges de 20 pixels.
 3. Traçage du repère, et éventuellement de la grille si le bouton radio a été coché, à l'aide de l'objet **traceur**.
 4. Traçage de toutes les courbes des fonctions stockées dans l'historique **log**.



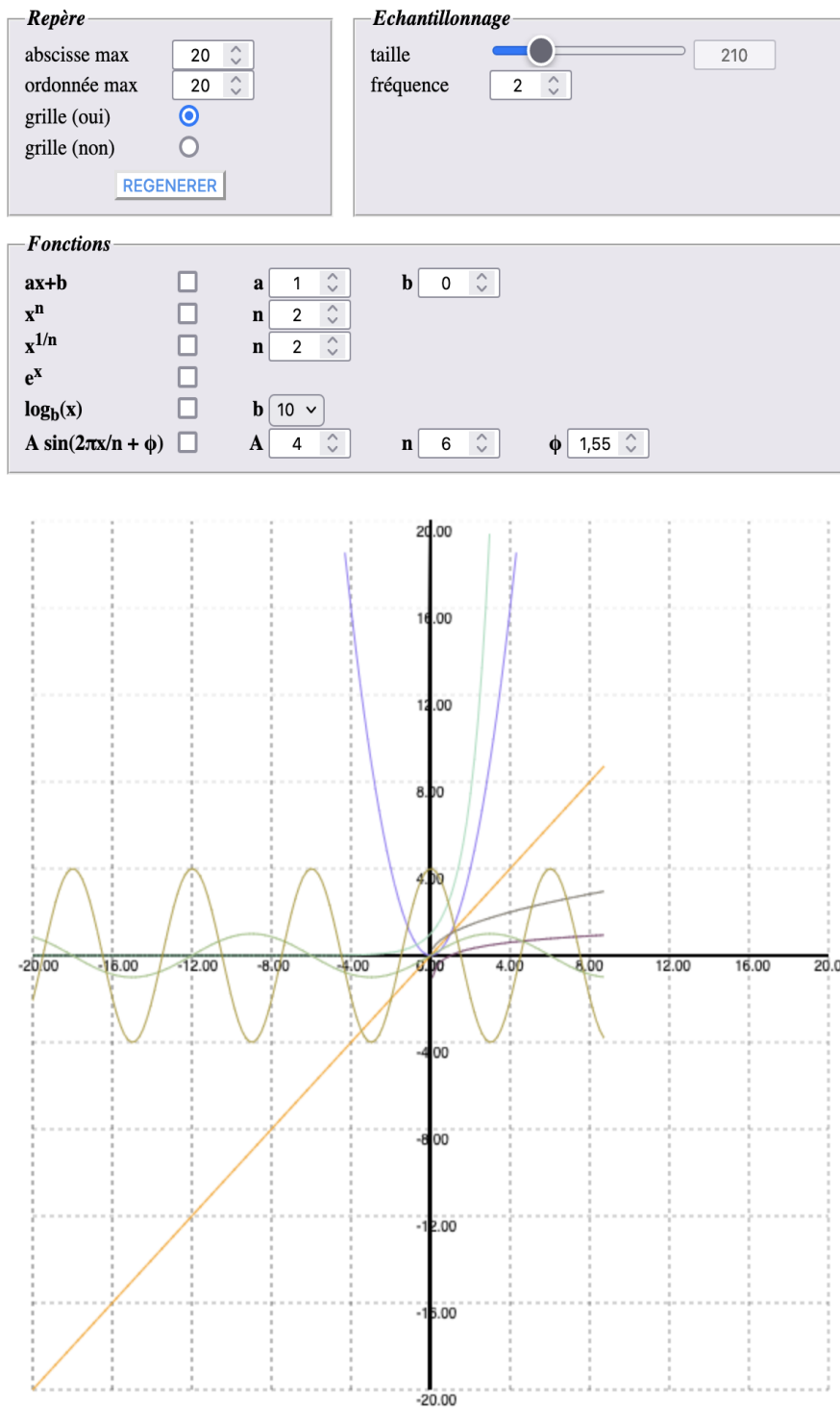


FIGURE 2 – Instantané d'un tracé dynamique de courbes