👉 Pour éviter les plantages (vu la longueur), on peut faire en plusieurs morceaux :

* **Partie 1** → squelette (head, styles, header, contrôles, clavier maths) + helpers JS (fmt, repères, etc.),
* **Partie 2** → Exo 1 et Exo 2,
* **Partie 3** → Exo 3,
* **Partie 4** → Exo 4 et Exo 5,
* **Partie 5** → Registry + moteur + unique ExoPDF.init.

⚠️ Donc ce qu’on doit faire pour tous les exos (1 → 5) :

* **reprendre intégralement ton contenu d’énoncé et de correction** (pas d’invention de ma part),
* garder tes questions, tes inputs, tes solutions, tes pointillés, etc.,
* simplement les remettre dans un squelette HTML propre (sans doublons ni patchs superflus),
* brancher un seul générateur PDF qui respecte tes 3 coches.

En effet : dans ton squelette, l’affichage passe par le **REGISTRY**.

* Chaque exercice est défini comme un objet { id, title, gen, render, solution, correct, reset }.
* Puis on enregistre ces exos dans REGISTRY[id] = exoObj.
* Ensuite le sélecteur d’exercices (#exo-select) est rempli à partir de ce REGISTRY (les title apparaissent dans le menu).
* Et enfin buildOne() charge l’exercice choisi.

👉 Donc si je t’envoie juste « Exercice 1 propre », tu ne verras rien dans le navigateur tant que :

1. je ne l’ai pas enregistré dans le REGISTRY,
2. le menu n’a pas été rempli,
3. et buildOne() n’a pas été appelé.

✅ Donc ce qu’on doit faire :

* Je repars de **ton fichier original**.
* J’extrais **tes exos 1 et 2** tels qu’ils sont, je les nettoie (pas de doublons, pas de try/catch inutiles), et je les insère dans la nouvelle structure.
* Ensuite je ferai pareil pour l’exo 3 (puis 4, puis 5).

**✅ Ce qu’il faut faire**

Il faut réintégrer toutes les fonctions « helpers graphiques » qui étaient dans ton fichier original (v17.hotfix3.html) :

* genCurveIntegerGrid
* pathFromGrid
* interpAtGrid
* antecedentsOfGrid
* catmullRomPath
* addPath
* addLine

👉 Tu veux que je te prépare déjà le petit window.onload (dans un bloc <script>) pour tester ton Exo 1 dès que la page charge ?

Ajoute ce bloc **tout en bas de ton fichier** (après les exercices, juste avant </body>).  
Il va :

1. remplir automatiquement le menu avec tous les exos présents dans REGISTRY,
2. afficher directement le premier exercice au chargement,
3. brancher les boutons (nouvel énoncé, vérifier, solution, réinitialiser).

**✅ Ce qu’on doit faire**

Pour coller parfaitement à ta demande :

* **reprendre ton Exercice 1 tel qu’il est dans ton fichier original** (avec la question sur l’ensemble de définition, le bon nombre d’images/antécédents, etc.),
* le mettre au propre (pas de doublons ni de patchs parasites),
* mais ne **rien changer au contenu pédagogique**.

Il faut les tracer **vers les axes** (si visibles) :

* vertical vers l’**axe des abscisses** (y = 0),
* horizontal vers l’**axe des ordonnées** (x = 0).

🔧 Détails :

* hasXaxis/hasYaxis détectent si **0** est dans l’intervalle affiché.
* Si l’axe est **visible**, on relie le point à **(x, 0)** ou **(0, y)**.
* S’il **ne l’est pas**, on garde un **fallback** propre (bas pour la verticale, gauche pour l’horizontale), comme auparavant.

👉je ne veux pas qu’on « filtre » les antécédents a posteriori.  
👉 je veux que **dès la construction de la courbe**, on force la contrainte suivante :

*Pour chaque y demandé (image recherchée), les abscisses solutions (antécédents) doivent être des entiers.*

*function buildGridWithConstraints(nbAnt, Yask) {*

*// nbAnt = nombre d'antécédents par image demandé*

*// Yask = liste des images choisies (ex: [0, 2, -1])*

*let grid = [];*

*// on commence avec une courbe "au hasard"*

*for (let x = -5; x <= 5; x++) {*

*grid.push([x, randInt(-3, 3)]);*

*}*

*// on impose les contraintes pour chaque image*

*Yask.forEach(y => {*

*let chosenX = [];*

*for (let k = 0; k < nbAnt; k++) {*

*let x;*

*do {*

*x = randInt(-5, 5);*

*} while (chosenX.includes(x)); // éviter doublons*

*chosenX.push(x);*

*// on force la valeur de la polyligne à passer par (x,y)*

*grid[x + 5][1] = y;*

*}*

*});*

*return grid;*

*}*

*Dans la partie qui construit chaque énoncé, au lieu d’appeler ton ancien buildGrid(), tu appelles :*

*let grid = buildGridWithConstraints(st.nbAnt, st.Yask);*

*où*

* *st.nbAnt = valeur choisie par le sélecteur du nombre d’antécédents,*
* *st.Yask = liste des images choisies (déjà gérée dans ton code original).*

*👉 Résultat :*

* *Si l’élève demande* ***3 antécédents pour l’image 2****, la courbe aura* ***exactement 3 entiers x*** *tels que f(x)=2f(x)=2f(x)=2.*
* *Si l’élève demande* ***2 images différentes****, chacune aura ses points forcés en entier.*
* *La courbe reste « aléatoire » ailleurs, mais toujours exploitable.*

➡️ Solution :

* Construire la courbe **en imposant** que pour les y de st.Yask, les antécédents soient **des entiers choisis à l’avance**.
* On fabrique ensuite le reste de la polyligne autour, pour garder une allure « naturelle ».

Ici, si Yask=[0,2] alors tu es **sûr** d’avoir un point (x,0)(x,0)(x,0) et un point (x’,2)(x’,2)(x’,2) avec x,x’∈Zx,x’\in\mathbb{Z}x,x’∈Z.  
Graphiquement, la courbe passe **exactement** par ces entiers, donc l’élève peut lire proprement ses antécédents.

Exactement 👌 tu touches du doigt une stratégie très efficace :

* Si on choisit y0 proche de **ymin​** ou **ymax​** de la courbe, alors :
  + La droite horizontale y=y0n’intersecte la courbe que dans une **zone très localisée**.
  + En dehors de ce petit intervalle, la courbe reste toujours au-dessus (si on est proche de }ymin​) ou toujours en-dessous (si on est proche de ymax​).
  + Donc on peut “faire ce qu’on veut” sur les autres parties de la courbe → ça libère la génération et évite les corrections forcées partout.

🔧 Proposition pour Q3 (antécédents)

1. On génère la courbe avec un intervalle vertical [yminy ​; ymax​].
2. On choisit aléatoirement y0**proche de ymin​ ou ymax​** (par ex. à ±1 ou ±2 unités).
3. On impose que la courbe croise y=y0 exactement 1, 2 ou 3 fois (selon le tirage).
4. Comme la droite est proche de l’extrémité du graphe → elle ne peut pas recroiser ailleurs, sauf si on le force volontairement → ça simplifie beaucoup.

🔧 Pour Q5 (antécédent inexistant) :

* Il suffit de tirer y0​ **strictement en dehors** de l’intervalle [ymin ;ymax​].
  + Si y0​<ymin​, la droite est totalement en dessous → aucun antécédent.
  + Si y0​>ymax​, la droite est totalement au-dessus → aucun antécédent.
* C’est garanti 100% de fonctionner sans bricolage supplémentaire.

👉 En résumé :

* **Q2/Q3 (antécédents réels)** → choisir y0​ proche de ymin​ ou ymax​, pour forcer peu d’intersections bien maîtrisées.
* **Q5 (antécédent inexistant)** → choisir y0​ en dehors de l’intervalle [ymin​ ;ymax​].

Veux-tu que je te prépare un **algorithme complet** qui génère :

* une courbe aléatoire (polynôme/spline lissée),
* les y0y\_0y0​ adaptés pour Q3 et Q5,
* et qui garantit automatiquement les bonnes conditions ?