Journées de l'APMEP

Programmation et géométrie dynamique avec Scratch, CaRMetal et DGPad

Présentation des concepts

On appelle ici « programmation dynamique » une programmation qui intègre des éléments dynamiques et maintient des liaisons dynamiques (autrement dit dans le même sens que dans l'expression géométrie dynamique).
La programmation dynamique est un enjeu important de la programmation dans le cadre d'un espace de géométrie dynamique : très souvent on souhaite maintenir des liaisons dynamiques dans les objets construits par script (et le logiciel le permet).

Intérêts

La programmation dans le cadre d'un espace de géométrie dynamique permet d'automatiser des taches de géométrie dynamique et d'obtenir des constructions complexes : on exploite le potentiel de la programmation dans le domaine de la géométrie dynamique.
Inversement (et simultanément) on exploite le potentiel de la géométrie dynamique pour développer des compétences de programmation : on travaille dans un espace familier de géométrie avec un retour visuel qui peut valider le programme.

Outils

Quand on programme dans le cadre d'un espace de géométrie dynamique on utilise un langage de programmation (typiquement Javascript ou Python) enrichi d'instructions de l'espace de géométrie dynamique utilisé.
Les instructions de l'espace de géométrie dynamique peuvent être utilisées de façon autonome (à la manière de ce que l'on fait/faisait dans TracenPoche). Mais l'intérêt principal consiste à cumuler les deux approches.

1) Exemple : programmation dynamique des racines d'une équation de degré 2

- avec CaRMetal ;
- avec DGPad (surcouche Blockly).

On considère l'algorithme qui permet de calculer les racines d'une équation de degré 2.

Si on travaille de géométrie dynamique, on peut utiliser trois curseurs a, b, c.

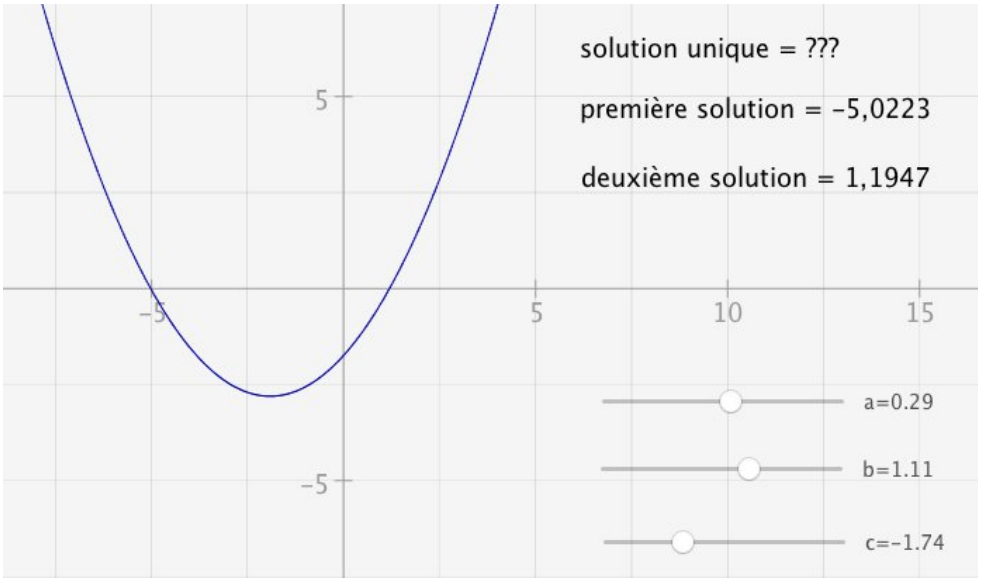
On peut créer les curseurs via l'interface. Puis le script suivant permet de calculer et d'afficher les solutions de l'équation.

```
pseudo-code
a ← PrendreValeurExpression("a")
b ← PrendreValeurExpression("b")
c ← PrendreValeurExpression("c")

delta ← puissance(b,2)- 4×a×c
si delta<0
   Afficher "Pas de solution."
sinon
   si delta =0
      x ← -b/(2×a)
      Afficher "1 solution:" + x
   sinon
      x1 ← (-b-racine(delta))/(2×a)
      x2 ← (-b+racine(delta))/(2×a)
      Afficher "première solution: " + x1
      Afficher "deuxième solution: " + x2
```

On peut aussi afficher les solutions dans la construction via des expressions sol, sol1 et sol2 créées préalablement.

```
pseudo-code
a ← PrendreValeurExpression("a")
b ← PrendreValeurExpression("b")
c ← PrendreValeurExpression("c")

delta ← puissance(b,2)- 4×a×c
si delta<0
   Afficher "Pas de solution."
sinon
   si delta =0
      x ← -b/(2×a)
      MettreValeurExpression("sol",x)
   sinon
      x1 ← (-b-racine(delta))/(2×a)
      x2 ← (-b+racine(delta))/(2×a)
      MettreValeurExpression("sol1",x1)
      MettreValeurExpression("sol2",x2)
```

On peut aussi ajouter l'affichage de la courbe. Et on obtient cette figure :



Les expressions sol, sol1 et sol2 ne sont pas dynamiques par rapport à a, b, c. On n'a pas fait de la programmation dynamique.

On voudrait que les expressions sol, sol1 et sol2 soient dynamiques par rapport à a, b et c. On peut le faire via l'interface en donnant :

à sol la valeur if(b^2-4*a*c==0,-b/(2*a),1/0)

à sol1 la valeur (-b-sqrt(b^2-4*a*c))/(2*a)

à sol2 la valeur (-b+sqrt(b^2-4*a*c))/(2*a)

En programmation dynamique, on va obtenir le même résultat par script.

Le script peut créer les expressions avec les bonnes valeurs. Voici un script de programmation dynamique :

```
pseudo-code
Expression("sol","if(b^2-4*a*c==0,-b/(2*a),1/0)",5,5)
Expression("sol1","(-b-sqrt(b^2-4*a*c))/(2*a)",5,3)
Expression("sol2","(-b+sqrt(b^2-4*a*c))/(2*a)",5,1)
```

On voudrait maintenant que le script affiche aussi la somme et le produit des racines.
On pourrait créer une expression de valeur "sol1 + sol2" et on obtiendrait le résultat.

Mais on va se donner une contrainte supplémentaire.
Dans le script précédent, on a nommé nous-même les expressions "sol " "sol1 " et "sol2 ".
Mais c'est souvent une mauvaise pratique (ou une pratique impossible). Car il pourrait exister dans la figure un autre objet nommé "sol", auquel cas le script créerait une expression nommée "sol*" et les références que l'on ferait ensuite à l'objet "sol " seraient fausses.

C'est pourquoi dans le script il vaut mieux laisser le logiciel nommer les expressions et récupérer ce nom.

On peut reformuler le script précédent ainsi :

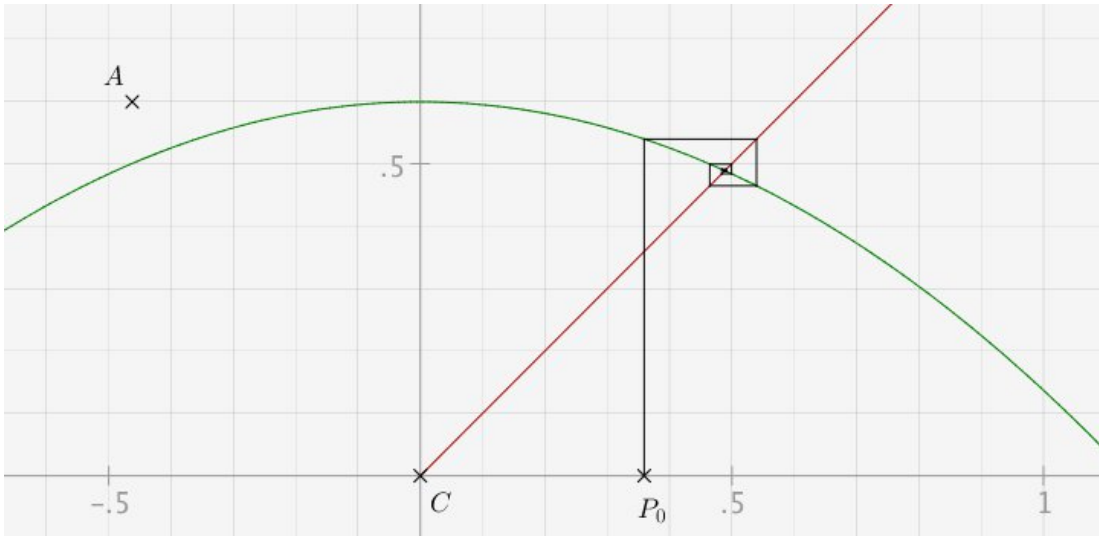
```
pseudo-code
jsSol ← Expression("if(b^2-4*a*c==0,-b/(2*a),1/0)",5,5)
jsSol1 ← Expression("(-b-sqrt(b^2-4*a*c))/(2*a)",5,3)
jsSol2 ← Expression("(-b+sqrt(b^2-4*a*c))/(2*a)",5,1)
```

Et on cherche à afficher la somme et le produit des racines (dynamiques par rapport à a,b,c) (pour illustrer la propriété que c'est égal à -b/a et c/a).

TP

2) Programmation dynamique de la suite définie par récurrence : $u_{n+1} = p.\ u_n^2 + q$, p et q des paramètres réels

Voici la figure que l'on veut obtenir :



Pour les paramètres p et q, on peut utiliser les coordonnées d'un point libre A.

TP CaRMetal et/ou DGPad

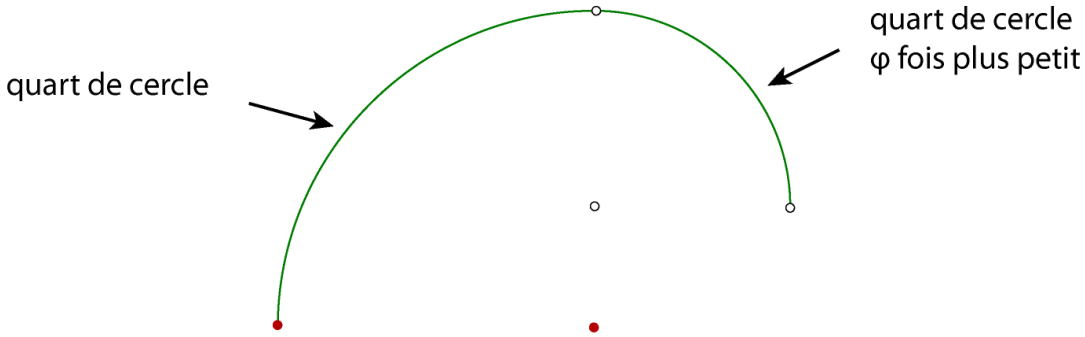
3) Spirale d'or

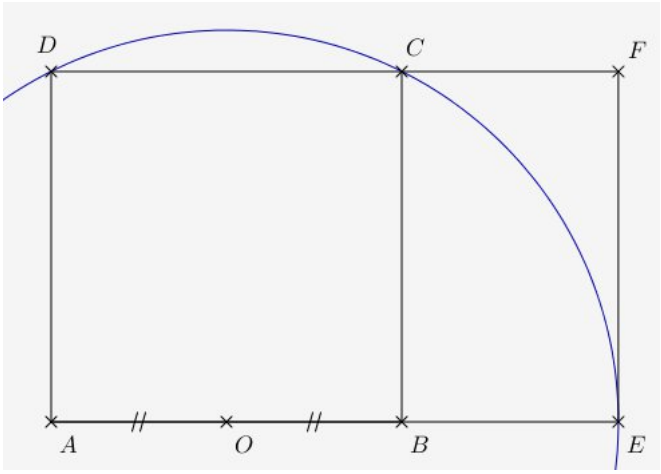
TP

Programmation dynamique avec une macro (CaRMetal)

Programmation non dynamique avec la tortue (Scratch)

Programmation dynamique avec la tortue dynamique (DGPad)





ABCD est un carré, AEFD est un rectangle d'or. $\varphi = AE/AD$.