ISN1 - PROGRAMMATION ÉVÉNEMENTIELLE

Auteur : Marie-Laurence Brivezac **TI-Nspire™ CAS**

Mots-clés : programmation, LUA, spécialité ISN.

Fichiers associés : signe_expr_d2.tns ; signe_expr_d2.lua ; signe_expr_d2.pdf

1. Objectifs

Mise en œuvre de la programmation avec LUA pour TI-Nspire et débuter un projet de programmation événementielle.

2. Énoncé

Une expression du second degré d'une variable réelle étant donnée, afficher son tableau de signe.

3. Commentaires

Les bases de la programmation de seconde et première sont supposées acquises ainsi que les principales manipulations de la calculatrice TI-Nspire CAS.

4. Conduite de l'activité

L'activité est conduite sur ordinateur du fait de l'utilisation du langage LUA en complément des fonctionnalités de la calculatrice. Les écrans du logiciel TI-Nspire CAS sont néanmoins de type calculateur pour une mise en page finale correcte sur la calculatrice qui reste la cible finale de l'affichage.
Ce point de vue pourrait naturellement être reconsidéré.

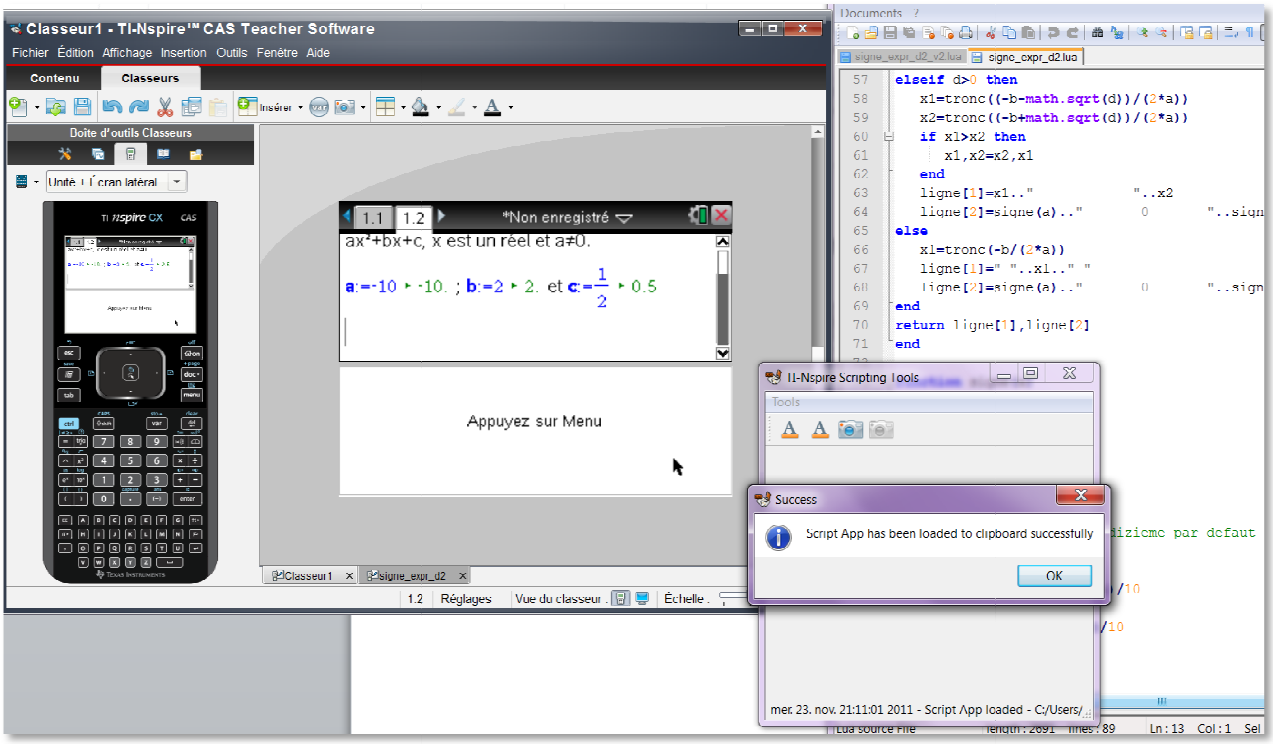
a. Mise en œuvre de l'environnent de développement

Avant tout télécharger l'application « scripting tool » pour l'interprétation du code LUA :
http://education.ti.com/educationportal/sites/US/nonProductSingle/nspire-scripting.html
Le dossier « télécharger » comporte notamment :
- l'outil d'interprétation : TI-Nspire Scripting Tool,
- la documentation des fonctions : TI-Nspire scripting Interface.

Quelques tutoriels :
http://www.inspired-lua.org/?lang=fr

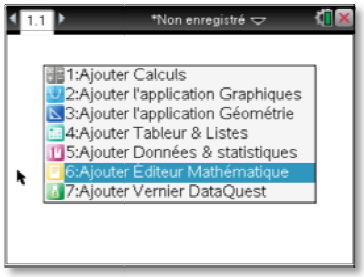
Le site LUA :
http://www.inspired-lua.org/?lang=fr
Pour commencer le développement, le code est écrit à l'aide d'un éditeur de son choix, par exemple **NotePad++**, où l'on prendra soin de préciser que l'on écrit du LUA pour une présentation automatique du texte édité.
L'écran de travail pourra être constitué de l'éditeur, de l'interpréteur et du logiciel TI-Nspire CAS pour le test comme présenté ci-après.

Ecrire le code LUA dans l'éditeur.
Sauvegarder dans le fichier de son choix avec l'extension « .lua ».
Charger le fichier dans l'interpréteur (Scripting Tool). Le code interprété est placé dans le presse-papier.
Coller (Ctrl V) le code interprété dans le classeur TI-Nspire pour le test.
Recommencer en cas d'erreur à corriger.



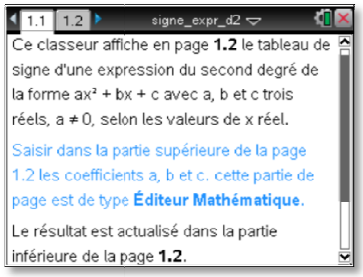
b. Développement de l'application côté TI-Nspire CAS

Les coefficients de l'expression étudiée peuvent être saisis dans une page **Calculs** ou **Éditeur Mathématique**.
Pour des raisons de présentation, on choisit une page **Éditeur Mathématique**.
Ouvrir deux pages de ce type.

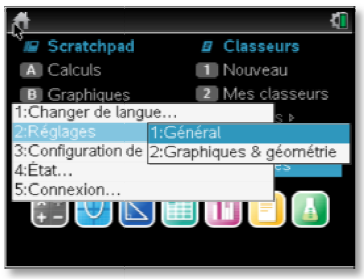


La page **1.1** présente l'application avec ce texte :

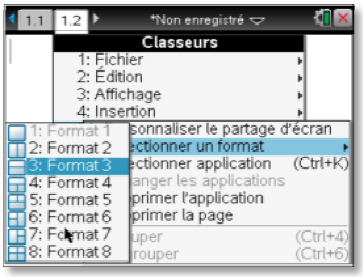
> Ce classeur affiche en page **1.2** le tableau de signe d'une expression du second degré de la forme $ax^2 + bx + c$ avec a, b et c trois réels, $a \neq 0$, selon les valeurs de x réel.
> Saisir dans la partie supérieure de la page **1.2** les coefficients a, b et c. Cette partie de page est de type **Éditeur Mathématique**.
> Le résultat est actualisé dans la partie inférieure de la page 1.2.
> Les racines sont données au dixième par défaut.
>
> Le classeur est réglé pour donner des valeurs approchées.

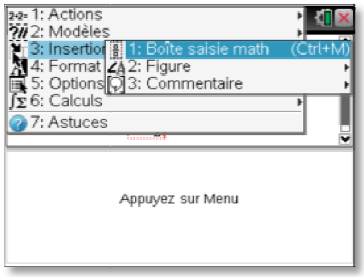


Remarque :
Le réglage en « valeurs approchées » permet de faciliter les affichages du tableau de signe.



La page **1.2** de type **Éditeur Mathématique** est partagée en deux dans le sens horizontal : doc ▾ 5 2 3.
La partie supérieure est complétée pour définir les coefficients *a*, *b* et *c*.

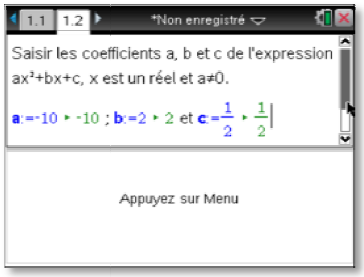




Écrire tout d'abord le texte de présentation :

> Saisir les coefficients a, b et c de l'expression $ax^2 + bx + c$, x est un réel et $a \neq 0$.

Suivi de trois boites mathématiques pour définir les variables a, b et c :
menu 3 1 ou ctrl M.

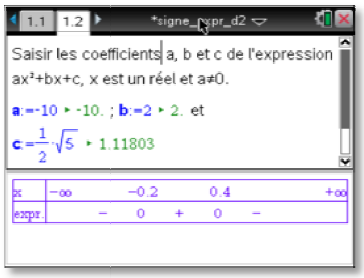


Remarque :
Après la saisie de la valeur du coefficient dans une boite mathématique, valider par enter.
Cela provoque l'affichage en vert de la valeur mémorisée.

A ce stade, la calculatrice est prête à recevoir le code LUA pour tracer le tableau de signe dans la partie inférieure de la page.

En considérant le code LUA écrit (partie suivante),
- ouvrir le TI-Nspire scripting toll,
- charger le fichier signe_expr_d2.lua avec le bouton A le plus à droite ; cela provoque l'interprétation du code LUA qui est placé dans le presse-papier,
- se placer dans la page **1.2** partie supérieure et coller le code avec ctrl V ;

la partie inférieure de la page est utilisée pour les affichages du code LUA.



c. Développement de l'application coté LUA

L'objectif n'est pas de décrire en détail le langage LUA.
Ce programme utilise uniquement la notion événementielle du langage et rien d'autre.
L'orientation objet très intéressante n'est pas utilisée ni la programmation récursive.
Ces notions relatives à la programmation en général doivent faire l'objet d'autres fiches pour ne pas alourdir le propos.

En informatique, une **programmation événementielle** se dit d'un type de programmation fondé sur les événements. Elle s'oppose à la programmation séquentielle. Le programme sera principalement défini par ses réactions aux différents événements qui peuvent se produire. La programmation événementielle peut être définie comme une technique d'architecture, elle est basée sur un événement externe ou interne à l'application. Modifier une variable, utiliser le clavier sont des événements par exemple.

Le code LUA sera dédié à l'aspect graphique de l'application sur réception d'événements.

Consulter le code dans le fichier **signe_expr_d2.lua**, qui peut être ouvert avec un éditeur comme notepad++ (libre) ou dans sa version imprimée **signe_expr_d2.pdf** (voir aussi l'annexe, page 5).

Description du code LUA :

Remarque importante : Les commentaires du programme (les lignes vertes qui commencent par --) ne doivent contenir aucune accentuation.

Le programme contient trois parties.

Partie 1 : Surveillance des événements relatifs au problème, c'est-à-dire les variables a, b et c. La modification de ces variables entraîne la réactualisation de l'écran.

Partie 2 : Gestion de la fenêtre à afficher. Sa dimension est demandée pour permettre de positionner le tableau de signe.
Les éléments fixes du tableau sont d'abord tracés.
Les éléments variables sont calculés par les fonctions de la partie 3 qui retournent deux chaînes de caractères à afficher.

Partie 3: Les fonctions de calcul.
La fonction **etude()** étudie les différents types d'affichage selon le discriminant du polynôme.
La fonction **signe()** attribue le symbole de signe à afficher.
La fonction **arrondi()** détermine l'arrondi, au dixième par défaut, des racines du polynôme. Cet arrondi est indispensable pour faciliter la gestion des affichages sur une seule ligne.

ANNEXE

```
-- Attention: pas d'accentuation dans les commentaires
-------------------------------------------------------------------------------
-- Surveiller les evenements
-------------------------------------------------------------------------------
-- les variables de l'application
var.monitor("a")
var.monitor("b")
var.monitor("c")
-- Reactualiser la fenetre a l'arrivee de nouvelles donnees
function on.varChange(varlist)
  platform.window:invalidate()
end

-------------------------------------------------------------------------------
-- Gestion de l'affichage sur demande
-------------------------------------------------------------------------------
function on.paint(gc)
  local ww,x,a,b,c
  local result={"",""}
-- nombre de pixels de la fenetre
-- ce nombre permettra de gerer la position des affichages
  ww=platform.window:width()

-- Tracer du tableau de signe
  gc:setColorRGB(100, 0, 255) -- un melange de rouge et bleu
  gc:setFont("serif" , "r", 10)
  gc:drawRect( 5, 5, ww-10, 20)
  gc:drawRect( 5, 25, ww-10, 20)
  gc:drawLine(35, 5, 35, 45)
  x=gc:drawString("x", 6, 6,"top")
  x=gc:drawString("-o", x+26, 6,"top")
  x=gc:drawString("o", x-2, 6,"top")
  x=gc:drawString("+o", ww-25, 6,"top")
  gc:drawString("o", x-2, 6,"top")
  gc:drawString("expr.", 6, 26,"top")

-- Lecture des donnees dans la table des variables
  a=var.recall("a")
  b=var.recall("b")
  c=var.recall("c")
-- Determiner le signe et affichage
  result={etude(a,b,c)}
  gc:drawString(result[1], (ww-gc:getStringWidth(result[1]))/2, 6,"top")
  gc:drawString(result[2], (ww-gc:getStringWidth(result[2]))/2, 26,"top")
end
-------------------------------------------------------------------------------
-- FONCTIONS PROPRES A CE SCRIPT
-------------------------------------------------------------------------------
-- calcul des racines et mise en page de l'affichage
-- les resultats sont donnes en deux lignes de texte
function etude(a,b,c)
local d=b^2-4*a*c
local x1,x2=0,0
local ligne={"",""}
```

```
if d<0 then
   ligne[2]=signe(a)
elseif d>0 then
   x1=tronc((-b-math.sqrt(d))/(2*a))
   x2=tronc((-b+math.sqrt(d))/(2*a))
   if x1>x2 then
      x1,x2=x2,x1
   end
   ligne[1]=x1.."          "..x2
   ligne[2]=signe(a).."      0      "..signe(-a).."      0      "..signe(a)
else
   x1=tronc(-b/(2*a))
   ligne[1]=" "..x1.." "
   ligne[2]=signe(a).."      0      "..signe(a)
end
return ligne[1],ligne[2]
end

function signe(a)
 if a>0 then
  return("+")
  else
  return("-")
  end
end

-- arrondi des racines au dixieme par defaut
function tronc(x)
if x<0 then
  return -math.floor(10*-x)/10
else
  return math.floor(10*x)/10
end
end
```