

## LE PROGRAMME resol

Les TI-89, 92, 92+ et V200 ne possèdent pas de fonction permettant la résolution d'inéquations. Le programme resol répond à cette absence.

Il permet de résoudre toute inéquation du type  $f(x) > g(x)$  (ou tout autre symbole parmi  $>$ ,  $\leq$ ,  $\geq$ ) dans laquelle  $f(x)$  et  $g(x)$  sont des fractions rationnelles ou des sommes de fractions rationnelles. Le résultat retourné est alors sous forme d'intervalles ou de réunion d'intervalles, et il tient compte des *valeurs interdites*. Dans les mêmes conditions, il permet de résoudre une équation de type  $f(x) = g(x)$  en éliminant les valeurs interdites.

Donnons quelques exemples :

```
Algebra Calc Char PrgmIO Clean Up
introduire l'(in)équation:
3x-1>-2x+4
valeurs interdites:
{}
soit à résoudre:
3·x-1>4-2·x
on résout donc:
5·(x-1)>0
S= ]1, ∞[

x^2-3x+2≤0
valeurs interdites:
{}
soit à résoudre:
x^2-3·x+2≤0
on résout donc:
-(x-2)·(x-1)≥0
S= [1, 2]

introduire l'(in)équation:
(x+1)/(x-3)≥2
valeurs interdites:
{3}
soit à résoudre:
x+1
x-3
≥2
on résout donc:
-(x-7)
x-3
≥0
S= ]3, 7]

introduire l'(in)équation:
(x+1)^2/(x+1)<0
valeurs interdites:
{-1}
soit à résoudre:
x+1<0
on résout donc:
-(x+1)>0
S= ]-∞, -1[
PROGRAMS RAD AUTO FUNC 4/30
```

Premier cas très simple :

$$3x - 1 > -2x + 4$$

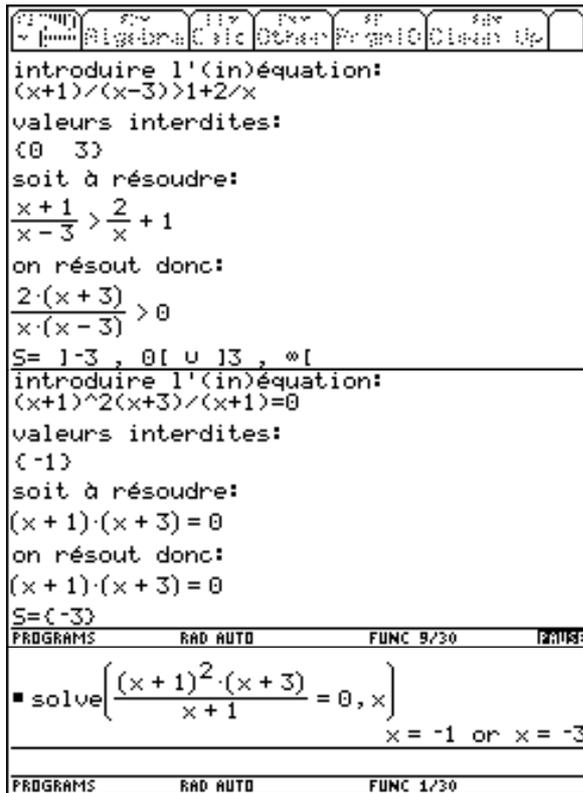
L'inéquation  $x^2 - 3x + 2 \leq 0$ .

Plus compliqué :  $\frac{x+1}{x-3} \geq 2$

L'énoncé est un peu farfelu, on supposera que l'on a aboutit à ce problème :

$$\frac{(x+1)^2}{x+1} < 0.$$

Terminons par deux derniers exemples.



L'inéquation  $\frac{x+1}{x-3} > 1 + \frac{2}{x}$

Et pour finir le cas de l'équation

$$\frac{(x+1)^2(x+3)}{x+1} = 0$$

La calculatrice est moins rigoureuse.

### Les obstacles rencontrés :

C'est au niveau des valeurs interdites que le problème a été le plus difficile à résoudre. C'est d'ailleurs ce qui a motivé l'écriture de ce programme puisque la demande des élèves était de pouvoir « faire confiance » à l'ensemble solution retourné et donc qu'il corresponde exactement aux contraintes imposées par l'ensemble de validité de l'inéquation ou de l'équation.

La calculatrice a été programmée de telle façon qu'un certain nombre de simplifications se font automatiquement ce qui bien sûr la conduit aux erreurs comme celle montrée ci-dessus. Ce constat justifie l'écriture d'un « gros programme » pour corriger ce problème.

J'ai cru dans un premier temps pouvoir utiliser le fait que bien qu'elle simplifie la calculatrice « connaît » les valeurs interdites comme le montre l'exemple suivant :

■ Define $f(x) = \frac{x^2 - 1}{x - 1}$	Done
■ f(x)	x + 1
■ f(-1)	0
■ f(1)	undef
■ Define $f(x) = \frac{x - 1}{x + 1}$	Done
■ f(-1)	undef
■ f(0)	0

La simplification est effectuée mais la valeur 1 reste interdite ; le problème est de savoir où la calculatrice range ce résultat et comment le retrouver

Dans ce second cas, avec un niveau de plus, la calculatrice ne parvient plus à déterminer les interdits (le programme non plus...)

L'idée de base pour empêcher l'évaluation était de rentrer l'(in)équation sous forme de chaîne de caractères. Mais ceci fait comment extraire le dénominateur éventuel avant toute simplification. Travailler sur cette chaîne en tentant de repérer les symboles « / » s'est rapidement révélé utopique tant le nombre de cas à prévoir était grand.

L'astuce va alors consisté à remplacer dans la chaîne de caractères chaque « x » successif par une lettre différente, ce que l'on fera en utilisant le fonction # (indirection). Une fois ce travail réalisé, on passe à l'évaluation et là bien entendu aucune simplification n'est possible. On peut alors sans problème isoler le dénominateur commun et trouver les valeurs interdites.

### Le texte commenté du programme

```

resol()
Prgm

Local signe,i, chb,interdit,valeurs
Local j,k,n,inf,sup,signes,f,solution
Local sgn1,sgn2,dans,symb,tri,sol,modif

ClrIO

Define tri(liste)=Func
Local listel,inf,i
If liste={} Then
    liste
    Return
EndIf
min(liste)→ inf
{inf}→ listel
While dim(liste)>0
    min(liste)→ inf
    If max(listel)<inf
        augment(listel,{inf})→ listel
    0 → i
    Loop
        1+i → i
        If liste[i]=inf
            Exit
        EndLoop
    augment(mid(liste,1,i-1),mid(liste,i+1))
    → liste
EndWhile
listel
EndFunc

Define dans(elmt,listel)=Func
Local t,m,fin
0 → fin
dim(liste)→ m
0 → t

```

La fonction tri, définie localement, généralise la fonction sortA de la TI-92 (tri croissant), d'une part parce que cette dernière instruction se comporte étrangement pour trier une liste comprenant des nombres négatifs avec des racines carrées, d'autre part parce que la fonction tri permet d'éviter les répétitions d'un élément.

**Argument d'appel** liste

**Si** liste est vide **alors retourner** liste

**Tant que** liste n'est pas vide

**Ajouter** le plus petit élément inf de liste à listel.

**Supprimer** inf autant de fois qu'il apparaît dans liste

**Fin de Tant que**

**Retourner** listel

La fonction dans permet de savoir si l'élément **elmt** appartient à la liste **liste**. Elle retourne 1 si c'est exact et 0 sinon.

```

Loop
  t+1 → t
  If t>m
  Exit
  If elmt=liste[t] Then
    1 → fin
    Exit
  EndIf
EndLoop
fin
EndFunc

```

**Argument d'appel** `elmt,liste`  
`m` := nombre d'éléments de liste  
`t` := 0  
**Répéter jusqu'à** `t > m` **ou** `fin = 1`  
`t` := `t + 1`  
**Si** `elmt = le tième élément de liste` **alors** `fin := 1`  
**Fin de Répéter**  
**Retourner** `fin`

```

InputStr "introduire l'(in)équation:",cha
Lire cha (chaîne contenant l'(in)équation)
part(expr(cha),0)→ signe
signe contient l'un des symboles : =, >, <, ≤, ≥

```

```

97 → j
cha → chb
Loop
  char(j)→ inc
  DelVar #inc
  inString(chb,"x")→ i
  If i=0
  Exit
  mid(chb,1,i-1)&inc&mid(chb,i+1)→ chb
  j+1 → j
EndLoop

```

Cette partie correspond au remplacement de la lettre x par les lettres a, b, c...  
 On place successivement "a", "b", ... dans **inc** à l'aide de la fonction char. La fonction # (indirection) permet d'effacer d'éventuels contenus des variables a, b...  
 On détermine la place de « x » (**i**). S'il n'y a plus de « x », c'est fini, sinon on remplace « x ».

```

{}→ interdit
j-97 → j
expr(chb)→ chb
right(chb)-left(chb)→ chb
getDenom(chb)→ chb
For i,1,j
  char(96+i)→ inc
  limit(chb,#inc,x)→ chb
EndFor
zeros(chb,x)→ interdit
Disp "valeurs interdites:"
Pause interdit

```

Recherche des valeurs interdites :  
 Transformer la chaîne **chb** en expression mathématique.  
 Faire « tout passer du même côté ».  
 Extraire le dénominateur (commun).  
 Remplacer chacune des lettres de ce dénominateur par x.  
 Chercher les zéros de l'expression obtenue.  
 Les ranger dans **interdit**  
**Ecrire interdit.**

*Transformation et affichage de l'(in)équation à résoudre.  
 Pour simplifier en ce qui concerne les inéquations, on se ramène à deux cas seulement.*

```

Disp "soit à résoudre:"
expr(cha) → cha
Pause cha
left(cha)-right(cha)→ cha
If signe="<" or signe="≤"
  (-)cha → cha
  cha → chb
Disp "on résout donc:"
If signe="<"
  ">" → signe
If signe="≤"

```

**cha** contient l'expression initiale évaluée et donc simplifiée. **Ecrire cha.**  
**Si** `signe = "<"` **ou** `"≤"` **alors**  
`cha ← -cha`  
`chb ← cha`  
**Si** `signe = "<"` **alors** `signe ← ">"`  
**Si** `signe = "≤"` **alors** `signe ← "≥"`  
 Construire l'expression `cha > 0` **ou**



valeurs[2]-1 → inf  
 valeurs[n-1]+1 → sup  
 {inf} → signes

inf ← deuxième élément de valeurs - 1  
 sup ← l'avant-dernier élément de valeurs + 1

*Si n=3, inf = sup et la liste signes est complète.*

```
If n ≥ 4 Then
  For i, 2, n-2
    (valeurs[i]+valeurs[i+1])/2 → signes[i]
  EndFor
EndIf
```

**Si n ≥ 4 alors**  
**Pour i** allant de 2 à n-2  
 signes[i] ←  $\frac{\text{valeurs}[i] + \text{valeurs}[i+1]}{2}$   
**Fin Pour**  
**Fin Si**

sup → signes[n-1]

Placer sup au bout de la liste signes

*On calcule les valeurs que prend chb pour chaque élément de signes, puis on range les signes de ces valeurs dans la liste signes.*

```
chb|x=signes → signes
sign(signes) → signes
```

*La fonction sign rend 1 ou -1 suivant que le nombre est positif ou négatif.*

*On obtient une relation entre valeurs et signes explicitée par le tableau ci-dessous.*

valeurs	valeurs[i]	valeurs[i+1]
signes	signes[i]	

```
" " → solution
" " → symb
```

*Initialisation des variables solution et symb. solution contient la chaîne donnant l'ensemble solution, symb contient soit " " soit "∪"*

*Début de la résolution proprement dite. On examine la situation selon un premier filtre celui du signe > ou ≥. Les variables sgn1 et sgn2 contiennent les crochets bornant les intervalles solutions. On va repérer les changements de signes dans la liste signes.*

*La variable modif est initialisée à 0 ; elle passe à 1 dans le cas particulier où l'on a ≥ pour signe et où les intervalles de part et d'autres d'un certain élément de valeurs conviennent sans que cette valeur soit interdite : dans ce cas, on supprime cette valeur et l'on recommence.*

*La variable sol est initialisée à 0 ; elle prend la valeur 1 si l'on a repéré un intervalle solution et la valeur 2 si l'on a repéré un réel isolé comme solution.*

*En reprenant le tableau précédent, on peut résumer les choses de la façon suivante :*

**Cas de signe = ">"**

*Dans ce cas peu importe que valeurs[i] et valeurs[i+1] soient des valeurs interdites ou des zéros de l'expression.*

valeurs[i]	valeurs[i+1]
signes [i] = -1	

sol ← 0  
 modif ← 0

valeurs[i]	valeurs[i+1]
signes[i] = 1	

sol ← 1  
 modif ← 0  
 solution ← solution ∪ ]valeurs[i],valeurs[i+1[

exemple :  $\frac{3x-1}{x+1} > 0$  pour valeurs[1] = -1 ou pour valeurs[2] =  $\frac{1}{3}$

Cas de **signe** = "≥"

La situation est plus compliquée

a) Le cas le plus simple :

il y a changement de signes et l'intervalle [valeurs[i], valeurs[i+1]] (ouvert ou fermé) ne fait pas partie de l'ensemble solution.

valeurs[i]	valeurs[i+1]	valeurs[i+2]
signes [i] = -1		signes[i+1] = 1

**sol** ← 0

**modif** ← 0

exemple :  $\frac{(x-1)(x+2)}{x-3} \geq 0$  pour valeurs[2] = -2

Une variante

valeurs[n-1]	∞
signes [n-1] = -1	

**sol** ← 0

**modif** ← 0

exemple :  $(x+1)(-x+3) \geq 0$   
pour valeurs[3] = 3

b) Un petit peu plus compliqué

valeurs[i]	valeurs[i+1]	valeurs[i+2]
signes [i] = 1		signes[i+1] = -1

**sol** ← 1

**modif** ← 0

**solution** ← **solution** ∪ (valeurs[i], valeurs[i+1]) (les parenthèses correspondant aux crochets ] ou [ suivant que valeurs[i] et valeurs[i+1] sont des valeurs interdites ou non

exemple :  $\frac{(x-1)(x+2)}{x-3} \geq 0$  pour valeurs[3] = 1

Une petite variante :

valeurs[n-1]	∞
signes [n-1] = 1	

**sol** ← 1

**modif** ← 0

**solution** ← **solution** ∪ (valeurs[i], ∞ [

exemple :  $2x + 1 \geq 0$  pour  $\text{valeurs}[2] = -\frac{1}{2}$

c) On se complique vraiment (avec bien sûr  $i < n - 1$ )

Pour le moment, ça va encore.

valeurs[i]	valeurs[i+1]	valeurs[i+2]
signes [i] = 1		signes[i+1] = 1

sol ← 1

modif ← 0

solution ← solution  $\cup$  (valeurs[i],valeurs[i+1])

exemple :  $\text{Error!} \geq 0$  pour  $\text{valeurs}[3] = -1$

Maintenant, on obtient un cas particulier important :

valeurs[i]	valeurs[i+1]	valeurs[i+2]
signes [i] = 1	0	signes[i+1] = 1

sol ← 0

modif ← 1

L'intervalle à considérer sera bien sûr (valeurs[i],valeurs[i+2]). Il convient donc d'éliminer valeurs[i+1] de la liste valeurs et signes[i+1] de la liste signes. Puis de faire :  $n \leftarrow n - 1$ .

Ensuite on recommence avec les nouvelles listes.

C'est ce qu'indique modif = 1

exemple :  $(x + 2)(x + 1)^2 \geq 0$  pour  $\text{valeurs}[3] = -1$

d) Le dernier cas, enfin (avec toujours  $i < n - 1$ )

Cette fois encore, il n'y a pas changement de signe mais aucun des deux intervalles n'est solution.

valeurs[i]	valeurs[i+1]	valeurs[i+2]
signes [i] = -1		signes[i+1] = -1

sol ← 0

modif ← 0

Ici pas de problème : valeurs[i+1] est une valeur interdite

exemple :  $\text{Error!} \geq 0$  pour  $\text{valeurs}[2] = -2$

valeurs[i]	valeurs[i+1]	valeurs[i+2]
signes [i] = 1	0	signes[i+1] = 1

sol ← 2

modif ← 0

valeurs[i+1] fait partie de l'ensemble des solutions (solution isolée) : c'est ce qu'indique sol = 2.

**solution**  $\leftarrow$  **solution**  $\cup$  {valeurs[i+1]}

exemple :  $(x + 1)(x + 2)^2 \geq 0$  pour valeurs[2] = -2

On peut passer à la programmation :

```
Lbl calcul
0  $\rightarrow$  modif
0  $\rightarrow$  i
Loop
i+1  $\rightarrow$  i
If i=n
Exit
0  $\rightarrow$  sol
If signe=">" Then
  "]"  $\rightarrow$  sgn1
  "["  $\rightarrow$  sgn2
  If signes[i]=1
    1  $\rightarrow$  sol
Else
  "["  $\rightarrow$  sgn1
  "]"  $\rightarrow$  sgn2
  If dans(valeurs[i],interdit)=1
    "]"  $\rightarrow$  sgn1
  If signes[i]=1 Then
    1  $\rightarrow$  sol
    If i<n-1 Then
      If signes[i+1]=(-)1 or sgn1="]" Then
        If dans(valeurs[i+1],interdit)=1
          "["  $\rightarrow$  sgn2
        Else
          0  $\rightarrow$  sol
      augment(mid(valeurs,1,i),mid(valeurs,i+2))
       $\rightarrow$  valeurs
      augment(mid(signes,1,i-1),mid(signes,i+1))
       $\rightarrow$  signes
      n-1  $\rightarrow$  n
      1  $\rightarrow$  modif
    Exit
      EndIf
      EndIf
    Else
      If i<n-1 Then
        If signes[i+1]=(-)1 Then
          If dans(valeurs[i+1],interdit)=0
            2  $\rightarrow$  sol
          EndIf
        EndIf
      EndIf
    EndIf
  EndIf
  EndIf
  EndIf
```

$i \leftarrow 0$

Répéter jusqu'à  $i = n$

$i \leftarrow i + 1$

**Si** signe = ">" **alors** prendre des crochets « ouverts »

**Si** signes[i] = 1 **alors** l'intervalle ]valeurs[i], valeurs[i+1][ fait partie de l'ensemble solution.

**Fin Si**

**Sinon** ( donc signe = "≥" )

prendre des crochets « fermés »

Si la borne inférieure est une valeur interdite, prendre un crochet ouvert.

**Si** signes[i] = 1 **alors**

**Si** signes[i + 1] = -1 (il y a changement de signe) **ou** sgn1 = "]" ( la valeur est interdite) **alors** il y a un intervalle solution : 1  $\leftarrow$  sol

Pour une valeur interdite, on change le crochet

**Sinon**( donc il n'y a pas changement de signes et ce n'est pas une valeur interdite), on supprime cette valeur dans valeurs et ce qui lui correspond dans signes. On fait  $n \leftarrow n - 1$

**Fin Si**

**Fin Si**

**Sinon** (c'est-à-dire signes[i] = -1)

**Si** il y a un autre intervalle qui suit ( $i < n-1$ ),

**alors si** cet intervalle n'est pas solution (signes[i + 1]= -1 et que la valeurs[i] n'est pas interdite, **alors** c'est une solution isolée 2  $\leftarrow$  sol

**Fin Si**

**Fin Si**

**Fin Si**

Construction de la chaîne **solution**

```
If sol=1 Then
If i=1
```

**Si** sol = 1 (il y a un intervalle solution)

```

"]" → sgn1
If i=n-1
 "[" → sgn2
solution&symb&sgn1&string(valeurs[i])&" ,
"&string(valeurs[i+1])&sgn2 → solution
" ∪ " → symb
EndIf
If sol=2 Then
solution&symb&string({valeurs[i+1]})→
solution
" ∪ " → symb
EndIf
EndLoop
If modif=1
Goto calcul

```

```

If solution=""
"{" → solution
Pause "S="&solution
EndPrgm

```

**alors si** c'est le premier ou le dernier  
il y a toujours des crochets ouverts  
(pour  $-\infty$  et  $\infty$ )

**solution** ← **solution** ∪ cet intervalle  
**symb** ← " ∪ "

**Fin Si**

**Si sol = 2** (*il y a une valeur isolée  
solution*)

**solution** ← **solution** ∪ {valeur isolée}  
**symb** ← " ∪ "

**Fin Si**

**Fin de la boucle Répéter**

Si **modif** = 1 on recommence tout le  
calcul.

**Si solution = ""** **alors solution** ← {}

**Ecrire solution**

**Fin du programme**

---