iter

Implémentation des fonctions *iterate* et *iterates* de *Derive*

Calculatrices : 89 92 92+ V200
Niveaux : Tale Tale S Sup
Descriptif : Groupe de fonctions – Implémentation des deux fonctions *iterate* et *iterates* du logiciel de calcul formel *Derive*
Auteur : Alain Pomirol
Mots-clefs : iterate, iterates
Date de dernière révision : Septembre 2002

Présentation :

***iterates*(exp,var,init,nb) :**

Cette fonction répète l'instruction *exp→var* où *var* est initialisée au début de l'itération par la valeur *init* et retourne la liste des résultats en commençant la liste par la valeur *init*. L'itération s'arrête si deux résultats consécutifs de la variable *var* sont égaux.

***iterate*(exp,var,init,nb) :**

Cette fonction répète l'instruction *exp→var* où *var* est initialisée au début de l'itération par la valeur *init* et retourne le dernier résultat. L'itération s'arrête si deux résultats consécutifs de la variable *var* sont égaux.

Mode d'emploi :

L'utilisation des deux fonctions est décrite dans la presentation ci-dessus.

Notons simplement que si *nb* est positif, le nombre de boucles est limité à *nb*. La liste a donc au plus *nb* + 1 éléments, mais peut-être moins si la méthode converge. Si *nb* est nul, la liste retournée est {*init*}.

En voici quelques exemples :

***iterates* :**

- `iterates(floor(x/2),x,1000,-1)` donne {1000, 500, 250, 125, 62, 31, 7, 3, 1, 0, 0}
- `iterates(floor(x/2),x,1000,3)` donne {1000, 500, 250, 125}
- `iterates(floor(x/2),x,1000,100)` donne {1000, 500, 250, 125, 62, 31, 7, 3, 1, 0, 0}
- `iterates(floor(x/2),x,1000,0)` donne {1000}
- Méthode de Newton : `iterates(x-u/d(u,x),x,x0,n)→newton(u,x,x0,n)`

`newton(x²-3,x,2,5)` donne {2, 1.75, 1.732143, 1.732051, 1.732051, 1.732051} en approximation, mode float7.

***iterate* :**

- `iterate(floor(x/2),x,1000,-1)` donne 0
- `iterate(floor(x/2),x,1000,3)` donne 125
- `iterate(floor(x/2),x,1000,0)` donne 1000
- Fonction puissance : `iterate(a*x,a,1,n)→pow(x,n)`

`pow(2,8)` donne 256

- Suite de Fibonacci : `iterate({v[2],v[1]+v[2]},v,{0,1},n)[1]→fib(n)`

`fib(100)` donne 354224848179261915075 en moins de 10 secondes !

- Composées d'une fonction : `iterate(f(y),y,x,n)→ff(x,n)`

`ff(2,5)` donne $f(f(f(f(f(2)))))$
Si on a défini $f(x) = x$^2, `ff(t,5)` donne t^{32} et `ff(2,5)` donne 4294967296

Restrictions :
- *var* doit être une lettre comprise entre *a* et *z*. Cela interdit les lettres grecques et les noms de variables à plusieurs caractères.
- L'expression *exp* ne supporte pas de paramètres non affectés autres que la variable *var*.

Remarque :
On peut parfois passer outre cette dernière restriction en utilisant les listes. Par exemple pour calculer les polynômes de Chebychev de première espèce, définis par la récurrence :
pour n=0, T(n,x) = 1
pour n=1, T(n,x) = x
pour n>1, T(n,x) = 2x T(n-1,x) − T(n-2,x)

Polynôme de Chebychev :
`iterate({v[2],2*v[2]*v[3]-v[1],v[3]},v,{1,x,x},n)[1]→cheb(n,x)`
`cheb(3,x)` donne $x\,(4\,x^2 - 3)$ et `tCollect(cheb(3,cos(x)))` donne cos(3x)
Quelques secondes suffisent pour calculer par exemple *tCollect*(*cheb*(15,cos(x))) !

Sources :

```
Iterate (θe,θv,θd,θf)
Func
Local
θi,θe_,θv_,θw_,a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,u,v,w,x,y,z,t
string(θe)→θe_
string(θv)→θv_
θd→#θv_
While  θf≠0
 #θv_→θw_
 expr(θe_)→#θv_
```

```
 θf-1→θf
 If  when(#θv_=θw_,true,false,false):Exit
EndWhile
#θv_
EndFunc

Iterates (θe,θv,θd,θf)
Func
Local
θe_,θv_,θw_,θl,a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z
string(θe)→θe_
string(θv)→θv_
{θd}→θl
θd→#θv_
While  θf≠Ø
 #θv_→θw_
 expr(θe_)→#θv_
 θf-1→θf
 augment(θl,{#θv_})→θl
 If  when(#θv_=θw_,true,false,false):Exit
EndWhile
θl
EndFunc
```