Génération de code en Maple

Hanane Naciri

INRIA
Sophia Antipolis

Email: hanane.naciri@sophia.inria.fr

Mai 1999

PLAN

- Introduction
- Générer du code en Maple
- Comment fonctionne MACROFORT90?
- Comment utiliser MACROFORT90?
- Conclusion

CONTENU

☞ • Introduction
  – Du symbolique au numérique
• Générer du code en Maple
• Comment fonctionne MACROFORT90?
• Comment utiliser MACROFORT90?
• Conclusion

Du symbolique au numérique

(Modélisation puis simulation)

1. Utiliser seulement le système de calcul formel

   -> *Problèmes de performances*

2. Utiliser seulement les langages numériques

   -> *Problèmes de travail et risques d'erreurs*

3. Utiliser le système de calcul formel pour obtenir le code numérique

   - À la main

     -> *Problèmes de travail et risques d'erreurs*

   - Liens avec MATLAB,SCILAB

     -> *Solution coûteuse*

   - Outils de génération de code

Outils de génération de code

- Traduction
  - Bibliothèque Codegen
    - Fonction fortran
    - Fonction C
- Génération
  - Bibliothèque Macrofort (C.Gomez)
  - Bibliothèque MacroC (P.Capolsini)

Fonction fortran

> $>$ expr $:= 1 - 2 * x + 3 * x^2 - 2 * x^3 + x^4;$
> $>$ fortran(expr);
> t0 = 1-2∗x+3∗x∗∗2-2∗x∗∗3+x∗∗4

> $>$ f $:= x$ $->Pi * ln(x^2) - sqrt(2) * ln(x^2)^2;$
> $>$ fortran(f);

```
c The options were : operatorarrow
      real function f(x)
      real x
        f = 0.3141593E1*alog(x**2)-sqrt(2.E0)*alog(x**2)**2
        return
      end
```

Fonction C

> $\text{expr} := 1 - x/2 + 3 * x^2 - x^3 + x^4$
>
> C(expr);

```
t0 = 1.0-x/2.0+3.0*x*x-x*x*x+x*x*x*x;
```

> $\text{f} := x \text{ ->} 1 - 3 * x^2 - 2 * x^3 + x^4$;
>
> C(f);

```
/* The options were : operatorarrow */
    double f(x)
    double x
    {
      {
        return(1.0-3.0*x*x-2.0*x*x*x+x*x*x*x);
      }
    }
```

INRIA

Bibliothèque Macrofort

> f := x ->$Pi * ln(x^2) - sqrt(2) * ln(x^2)^2$;
>
> genfor([subroutinem, foo, [],
>   [[commonf,toto,[a]],
>   [if_then_m,(a>0),[equalf,a,f(a)]]]]);

```
c
c       SUBROUTINE foo
c
   subroutine foo()
      common/toto/a
      if (0.lt.a) then
      a = 0.3141593E1*log(a**2)-sqrt(2.E0)*alog(a**2)**2
      endif
   end
```

Bibliothèque MacroC

> f:= x ->$Pi * ln(x^2) - sqrt(2) * ln(x^2)^2$;

```
> genC([[includeC, '<math.h>'],
        [includeC, '<stdio.h>'],
        [commentC, ''],
        [functionm, double, foo,[[double, [a]]],
        [[if_thenm, (a >0) ,[returnC, f(a)]]]]]);

#include <math.h>
#include <stdio.h>

double foo(a)
double a;
{
if (0 <a)
   return (0.3141593E1*ln(a*a)-sqrt(2)*pow(ln(a*a),2.0));
}
```

Bibliothèque Macrofort90

- Bibliothèque Maple
- Génération du code Fortran90
- Fortran90: Dernière évolution du standard fortran
  - Calcul vectoriel
  - Modularité

Fortran90

Exemple

```
function gauss( a, b, n )
   ...
   real, intent( inout ) :: b(:)
   real, intent( in ) :: a(size(b), size(b))
   ...
   real, dimension( : , : ) , allocatable :: t
   Allocate( t( n+1, n+2 ))
   t(:,:n)=a; t(:,n+1)=b

   !do i = 1, n
   !  do j = 1, n
   !    t(i, j) = a( i, j)
   !    t(i, n+1) = b(i)
   !  end do
   ! end do
   ...
   b(i)=t(i,n+1)-dot_product(t(i,i+1:n),b(i+1:))
   ...
   deallocate( t )
end function gauss
```

CONTENU

- ✓ • Introduction
- ✓ • Générer du code en Maple
- ☞ • Comment fonctionne MACROFORT90?
  - Architecture de MACROFORT90
  - Traduction des expressions
  - Optimisation du code
- Comment utiliser MACROFORT90?
- Conclusion

Architecture de Macrofort90

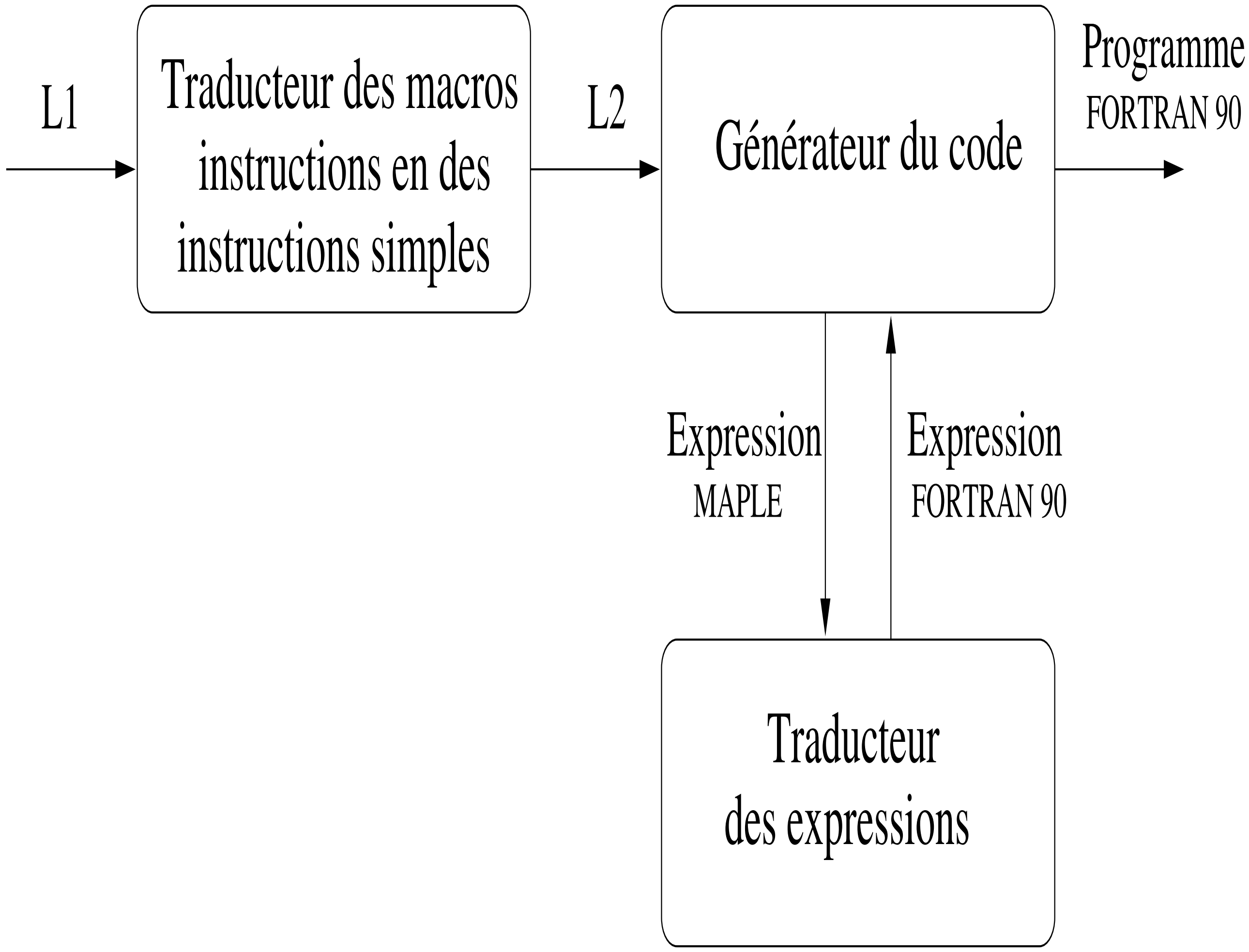
- **Les instructions simples**

[program_F, nom] –> PROGRAM nom

- **Les macros instructions**

[if_then_MF,condition,then_liste] –> if (condition) then
then_liste
end if

Architecture de MACROFORT90 *(suite)*



Architecture de MACROFORT90 *(suite)*

Exemple

```
L1:=[program_MF,analyse,
     [[declare_F,REAL, [A, B, `C(10,10)`]],
     [if_then_MF, if1,(CVAR &eq ``RESET``), [[call_F, find]]]]];
```

```
L2:=[[program_F, analyse],
     [[declare_F, REAL, [A, B, C(10,10)]],
     [if_then_F, if1, CVAR &eq 'RESET'],
     [[call_F, find]],
     [end_F, IF if1]],
     [end_F, PROGRAM analyse]];
```

Programme Fortran90 généré

```
PROGRAM analyse
  REAL  A, B, C(10,10)
  if1 : IF ( CVAR .eq. 'RESET' ) THEN
    CALL  find
  END IF if1
END PROGRAM analyse
```

Traduction des expressions

- Expressions algébriques

```
> m90_fortran(-a+d*e+b^c);
-a+d*e+b**c
```

- Expressions de chaînes de caractères

```
> m90_fortran(b &// c &// 'abc');
b // c // abc
```

- Expressions logiques

```
> m90_fortran(a = b);
a ==  b
> m90_fortran((a and b) &eqv c));
a .and. b .eqv. c
> m90_fortran(a and b &eqv c);
a .and. (b .eqv. c)
```

Optimisation du code

Exemple

> $>$ f:= x ->$Pi * ln(x^2) - sqrt(2) * ln(x^2)^2$;
> $>$ l:=[subroutine_MF,foo,
>   [[common_F,toto,[a]],
>   [if_then_MF,(a>0),[equal_F,a,f(a)]]]];
>
> $>$ **genf90(l);**

```
SUBROUTINE foo()
   COMMON /toto /a
   IF (0 <a) THEN
   a =0.3141593E1*log(a**2)-sqrt(2.E0)*alog(a**2)**2
   END IF
END SUBROUTINE foo
```

Optimisation du code *(suite)*

```
> genf90(l, optimized);
  SUBROUTINE foo()
     COMMON /toto /a
     IF (0 <a) THEN
     t1 = a**2
     t2 = log(t1)
     t4 = sqrt(2.0)
     t5 = t2**2
     a = 0.3141593E1*t2-t4*t5
     END IF
  END SUBROUTINE foo
```

CONTENU

Fonctionnement de Macrofort90

- Chargement de la bibliothèque

```
>with(share);
>with(macrof90);
```

- Initialisation

```
>initf90();
```

- Construction d'une liste des instructions Macrofort90

- Appel de la procédure genf90.

  - `filename=foo`
  - `precision`
  - `optimized`
  - `autodeclare=nom_type`

Algorithme de Newton

- **But:** Approcher la solution du système non linéaire $F(X) = 0$.

$$F(X) = \begin{pmatrix} f_1(x_1,\ldots,x_n) \\ . \\ . \\ . \\ f_n(x_1,\ldots,x_n) \end{pmatrix}$$

- **Algoritme:**

Etape 1 :	Poser $X = X_0 = (x_1,\ldots,x_n)$ (approximation initiale)
Etape 2 :	Calculer $F(X)$ et $F'(X)$ $F'(X) = \left(\frac{\partial f_i}{\partial x_j}\right)_{n,n}$
Etape 3 :	Résoudre le système linéaire $n \times n$ $F'(X)Y = F(X)$ (méthode de Gauss)
Etape 4 :	Poser $X = X - Y$
Etape 5 :	Si $\|\|Y\|\|_\infty \leq \varepsilon$ Alors retourner X . Sinon Etape 2

Programme Macrofort90 pour l'algorithme de Newton

```
newton:=proc (ff, x , n)
local  jac,ii,jj, do_program,l1,program;

#Jacobian matrix
jac:=array(1..n,1..n);
for ii from 1 to n do
 for jj from 1 to n do
     jac[ii,jj]:=diff(ff[ii], x[jj])
 od
od;

l1:=[equal_F, f(0), ff[1]];
for ii from n-1 to n do
  l1:=l1,[equal_F, f(ii-1), ff[ii]];
od;
```

18 Comment l'utiliser?

```
do_program:=[do_MF,
[[if_F, `eps>  norm(f)`, EXIT],
[matrix_MF, zj, jac],
[equal_F,`zjj(:`.n.`, :`.n.`)`, zj],
[equal_F,`b(1:`.n.`)`, `f(0:`.n.`-1)`],
[equal_F, i, gauss(zjj, b, n)],
[print_F, `*` ,[`b(1:`.n.`)`]],
[equal_F, `f(0:`.n.`-1)`,`b(1:`.n.`)` ],
[equal_F, x,`x-f(0:`.n.`-1)` ],
l1]
];


program:=[program_MF, newton,
[[use_F, RootNewton],
[implicit_F, none],
[declare_F, real, [`eps = 1.0e-2` ]],
[declare_F, real,
`dimension (1:`.(n+1).`,1:`.(n+1).`)`,[zjj]],
[declare_F, real,
`dimension (1:`.n.` , 1:`.n.`)`,[zj]],
[declare_F, real,`dimension (0:`.(n-1).`)`,[f]],
[declare_F, real,`dimension (1:`.(n+1).`)`,[b]],
[declare_F, real,`dimension (1:`.n.`)`,[x]],
[declare_F, integer, [i, j]],
[read_F, 10, [x] ],
[format_F, 10, [`E10.3`] ],
l1,
do_program,
[print_F, `*` ,[x]]

genf90([program], file=`newton.f90`);
end:
```

Programme Fortran90 généré

```
PROGRAM newton
  USE  RootNewton
  IMPLICIT none
  real  :: eps = 1.0e-2
  real ,   dimension ( 1:4 , 1:4) :: zjj
  real ,   dimension ( 1:3 , 1:3) :: zj
  real ,   dimension ( 0:2) :: f
  real ,   dimension ( 1:4) :: b
  real ,   dimension ( 1:3) :: x
  integer  :: i, j
  READ 10 ,x
  10 FORMAT (E10.3)
  f(0) = 3*x(1)-cos(x(2)*x(3))-0.5E0
  f(1) = x(1)**2-81*(x(2)+0.1E0)**2+sin(x(3))
         +0.106E1
  f(2) = exp(-x(1)*x(2))+20*x(3)
         +10.0/3.0*0.3141593E1-1
```

```
  DO
    IF ( eps>  norm(f) ) EXIT
     zj(2,1) = 2*x(1)
     zj(1,1) = 3
     zj(2,2) = -162*x(2)-0.162E2
     zj(2,3) = cos(x(3))
     zj(3,2) = -x(1)*exp(-x(1)*x(2))
     zj(3,1) = -x(2)*exp(-x(1)*x(2))
     zj(1,3) = sin(x(2)*x(3))*x(2)
     zj(1,2) = sin(x(2)*x(3))*x(3)
     zj(3,3) = 20
    zjj(:3, :3) = zj
    b(1:3) = f(0:3-1)
    i = gauss(zjj,b,3)
    PRINT  * , b(1:3)
    f(0:3-1) = b(1:3)
    x = x-f(0:3-1)
    f(0) = 3*x(1)-cos(x(2)*x(3))-0.5E0
    f(1) = x(1)**2-81*(x(2)+0.1E0)**2+sin(x(3))
           +0.106E1
    f(2) = exp(-x(1)*x(2))+20*x(3)
           +10.0/3.0*0.3141593E-1
  END DO
  PRINT  * , x
END PROGRAM newton
```

CONTENU

Conclusion

- Distribution de MACROFORT90 dans le share library
- Publication du rapport technique INRIA
- Connexion au logiciel TRANSFOR