The Ocaml Mathematical Framework

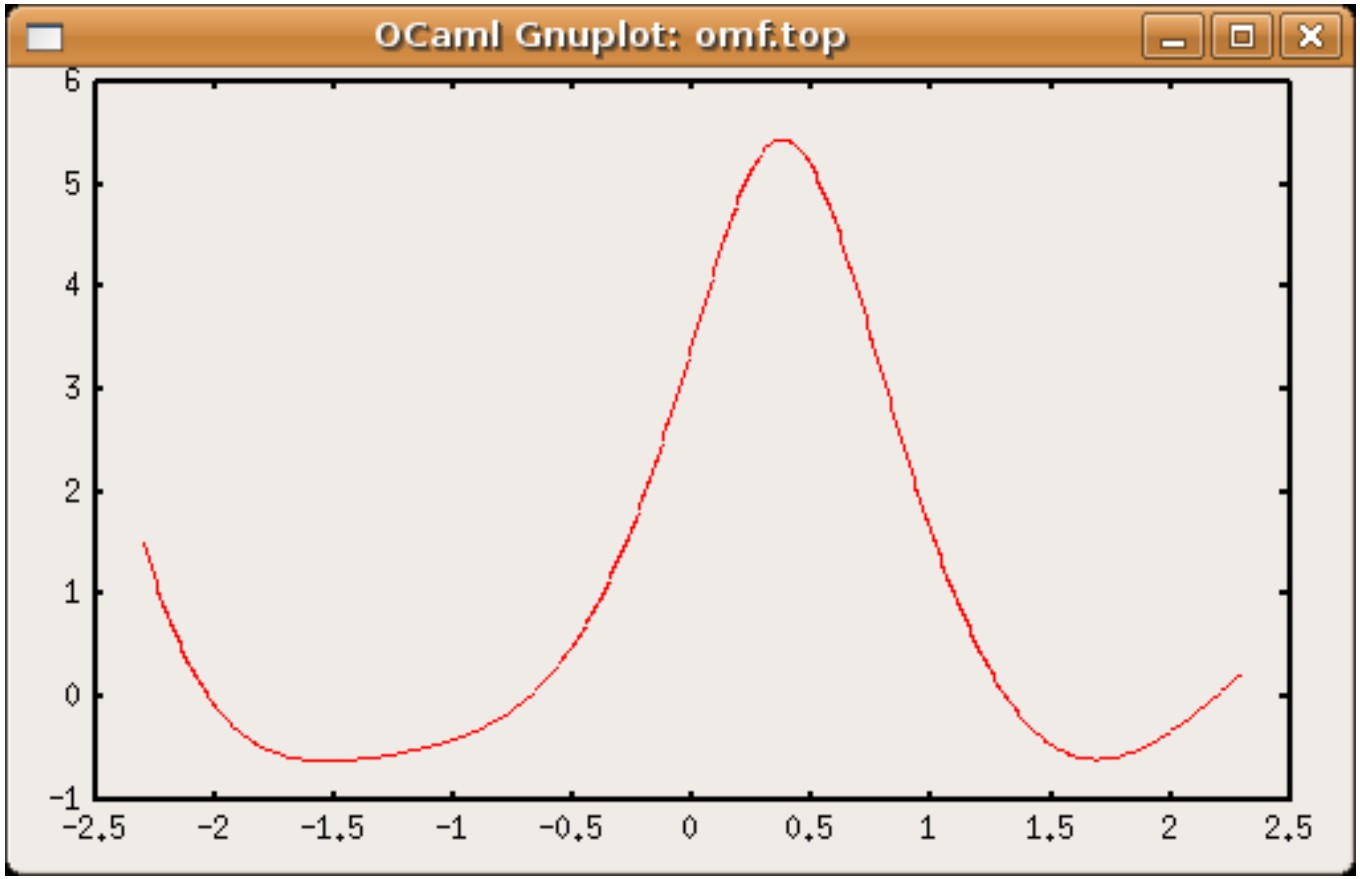
Olivier BOUISSOU and Alexandre CHAPOUTOT

Ocaml Summer Project Meeting - August 2008

Motivation : why OMF ?

Find roots of

$$E = (2x^2 - 3x - 3) * (sin(x + 1) + log(3x^2 - 2x + 1) - 2)$$



Motivation : why OMF ?

Find roots of

$$E = (2x^2 - 3x - 3) * (sin(x + 1) + log(3x^2 - 2x + 1) - 2) \quad (1.1)$$

Formal methods

- Notice E is a product.
- Roots of E are roots of :
- Special rules for polynomials
- *Home made* recipes for other terms
- *Example : MAPLE*

Numerical methods

- Consider E as a function of x
- Bisection method :
- Problem : only find *one* root and often need to give the initial bracketting $[a, b]$
- *Example : GSL*

Motivation : why OMF ?

Find roots of

$$E = (2x^2 - 3x - 3) * (sin(x + 1) + log(3x^2 - 2x + 1) - 2) \quad (1.1)$$

Formal methods

- Notice E is a product.
- Roots of E are roots of :
  - $2x^2 - 3x - 3$
  - $sin(x+1)+log(3x^2-2x+1)-2$
- Special rules for polynomials
- *Home made* recipes for other terms
- *Example : MAPLE*

Numerical methods

- Consider E as a function of x
- Bisection method :
- Problem : only find *one* root and often need to give the initial bracketting $[a, b]$
- *Example : GSL*

Motivation : why OMF ?

Find roots of

$$E = (2x^2 - 3x - 3) * (sin(x + 1) + log(3x^2 - 2x + 1) - 2) \quad (1.1)$$

Formal methods

- Notice E is a product.
- Roots of E are roots of :
- Special rules for polynomials
- *Home made* recipes for other terms
- *Example : MAPLE*

Numerical methods

- Consider E as a function of x
- Bisection method :
- Problem : only find *one* root and often need to give the initial bracketting $[a, b]$
- *Example : GSL*

Motivation : why OMF ?

Find roots of

$$E = (2x^2 - 3x - 3) * (sin(x + 1) + log(3x^2 - 2x + 1) - 2) \quad (1.1)$$

Formal methods

- Notice E is a product.
- Roots of E are roots of :
- Special rules for polynomials
- *Home made* recipes for other terms
- *Example : MAPLE*

Numerical methods

- Consider E as a function of x
- Bisection method :
  - find a, b s.t. $E(a) < 0 < E(b)$
  - divide $[a, b]$ into halves until it is small enough
- Problem : only find *one* root and often need to give the initial bracketting $[a, b]$
- *Example : GSL*

Motivation : why OMF ?

Find roots of

$$E = (2x^2 - 3x - 3) * (sin(x + 1) + log(3x^2 - 2x + 1) - 2) \quad (1.1)$$

Formal methods

- Notice E is a product.
- Roots of E are roots of :
- Special rules for polynomials
- *Home made* recipes for other terms
- *Example : MAPLE*

Numerical methods

- Consider E as a function of x
- Bisection method :
- Problem : only find *one* root and often need to give the initial bracketting $[a, b]$
- *Example : GSL*

Motivation : why OMF ?

Find roots of

$$E = (2x^2 - 3x - 3) * (sin(x + 1) + log(3x^2 - 2x + 1) - 2)$$

Can't we have

- a tool that has both advantages ?
- a tool that is easy to use and free ?
- a tool that we can use within our favorite language ?

YES

It is the goal of OMF

Motivation : why OMF ?

Find roots of

$$E = (2x^2 - 3x - 3) * (sin(x + 1) + log(3x^2 - 2x + 1) - 2)$$

Can't we have

- a tool that has both advantages ?
- a tool that is easy to use and free ?
- a tool that we can use within our favorite language ?

YES

It is the goal of OMF

A mix of formal and numerical method is necessary

Taylor method for root finding

- iterative method that requires derivative evaluation
- in GSL, the user needs to compute it by hand and give it as a C function

However...

- knowing the expression implies knowing its derivative
- derivation is painless for a computer, but may be hard for the user

Thus

For many algorithms, the formal representation of the expression saves the user from unecessary computations.

Goal of OMF

OMF is designed to be a mathematical library that allows :

- formal computations (derivatives, primitives,...)
- numerical problems solving (root finding, ODE solving, ...)

Requirements :

- Easy to use while well integrated into OCAML
- Good precision of the results rather than efficiency

Easy to use

- We want to write expressions as in MAPLE.
- We want to re-use already defined terms in new expressions.
- Solution : CAMLP4. We defined a syntax extensions that
  - parses a mathematical expression and builds the corresponding tree ;
  - keeps trace of the already defined terms.
- Example :

```
let symb p = 3**x**2-2*x-3;;
let symb e = p*(sin(x+1)+log(x**2-2*x+2)-2);;
```

- CAMLP4 transforms this into :

```
let p = E.expr_of_poly (P.add (P.poly_of_monom 3 "x" 2) ...
let e = E.mul p (E.add (E.expr_of_func "sin" (...)));;
```

Precision

- OMF can use GMP and MPFR as base types for (formal or numerical) constants.
- Thus, we can use an arbitrary number of bytes for the floating point representation.
- OMF can also use `Bigint` and `double` numbers when efficiency is more important.
- Selection between both is user defined when compiling the library.

Implemented methods

Formal methods :

- Polynomial computation
  - Karatsuba's multiplication
  - Univariate GCD and euclidian division
  - Square free factorization
- Rational functions
- Derivatives computation
- Primitive computation for polynomials and rational functions
- Naive formal root finder algorithm

Numerical methods :

- Roots finding
- ODE solving
- Expression minimization

Implemented methods

Formal methods :

- Polynomial computation
- Rational functions
  - Standard arithmetic
  - Simplifications in the univariate case
- Derivatives computation
- Primitive computation for polynomials and rational functions
- Naive formal root finder algorithm

Numerical methods :

- Roots finding
- ODE solving
- Expression minimization

Implemented methods

Formal methods :

- Polynomial computation
- Rational functions
- Derivatives computation
- Primitive computation for polynomials and rational functions
- Naive formal root finder algorithm

Numerical methods :

- Roots finding
  - Bijection method
  - Taylor method
- ODE solving
- Expression minimization

Implemented methods

Formal methods :

- Polynomial computation
- Rational functions
- Derivatives computation
- Primitive computation for polynomials and rational functions
- Naive formal root finder algorithm

Numerical methods :

- Roots finding
- ODE solving
  - Euler and RK4 method (fixed step size)
  - RKCK method (variable step size)
  - Taylor series method (variable step size)
- Expression minimization

Implemented methods

Formal methods :

- Polynomial computation
- Rational functions
- Derivatives computation
- Primitive computation for polynomials and rational functions
- Naive formal root finder algorithm

Numerical methods :

- Roots finding
- ODE solving
- Expression minimization
  - Brent's method with and without derivatives

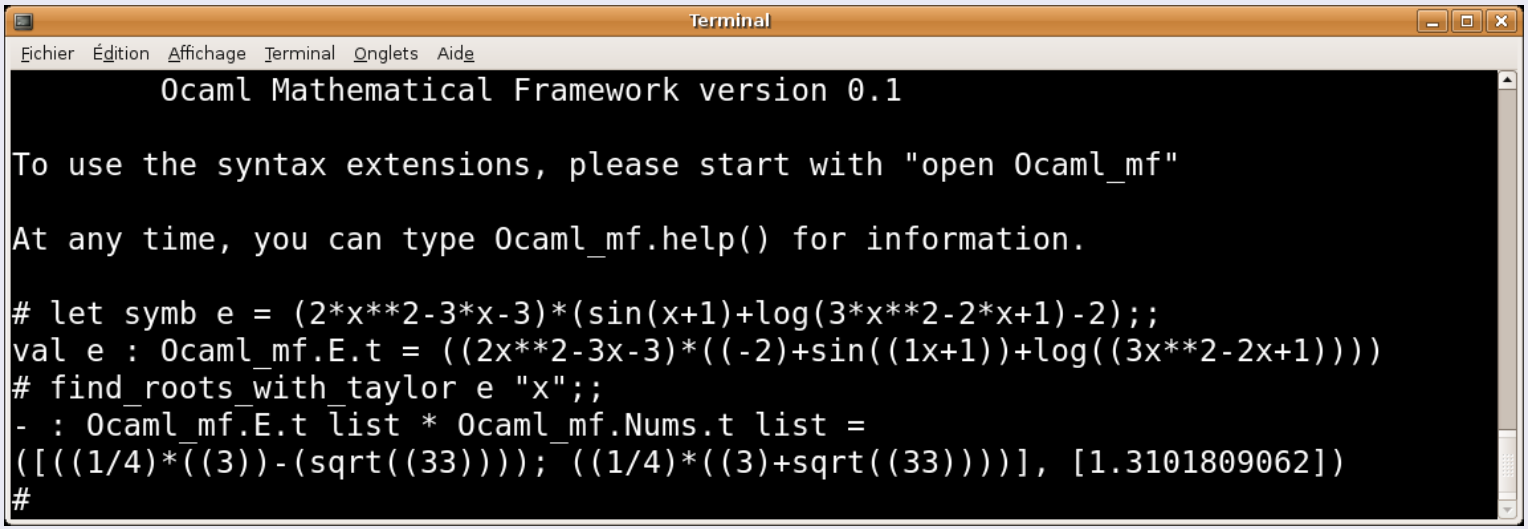
Using OMF

As a library

- OMF provides a byte code and a native code library.
- OMF syntax extensions are for OCaml 3.10 only.

As a top level

- OMF also comes with a top level with included help system.
- The top level allows for fast testing of the tool.



Conclusion

- OMF is usable for midsize problems.
- Mixing formal and numerical computations allows for the painless use of high order algorithms.
- Using MPFR/GMP makes the accuracy of the results almost as small as we want.

Future work :

- better expressions evaluation system,
- improve the formal kernel,
- make it public so that people can use it and make it better.