Numerical integration in Coq

Bas Spitters

Radboud University Nijmegen

20 Oct 2009

Outline

- Formath project
- Numerical integration
- Formalization (work in progress)
- Reconstruction of corn
- Type classes for algebraic hierarchy, efficient data structures, bigops

This talk

What are our plans? Request for input.

Formath project

EU STREP project on the formalization of constructive/effective algebra, analysis, algebraic topology.
Nijmegen (Geuvers/Spitters): computational analysis with applications to hybrid systems.

Formath project

EU STREP project on the formalization of constructive/effective algebra, analysis, algebraic topology.

Nijmegen (Geuvers/Spitters): computational analysis with applications to hybrid systems.

Exact Verified Analysis

Bishop's program to use constructive analysis for exact numerical analysis.

Vision: formalize a numerical analysis textbook.

Vacancy for a PhD-student

Bishop's program

Motivation:

- The need for formal verification is clearly recognized by the interval community.
- We need computations in our proofs.

How:
Use ssreflect library for discrete structures, completion monad for continuous ones.
Start with a correct implementation and speed up.

Bishop's program

What?

- Speeding up reals
- Numerical integration (higher type computations)
- ODEs
- Needed for Hybrid systems: e.g. Ariadne, using Taylor models.

Numerical integration

Riemann very slow, but general and verified! (DEMO)

Numerical integration

Riemann very slow, but general and verified! (DEMO)
Newton-Cotes. Approximate a function by a polynomial and integrate this.
First theory, then remarks about implementation

Lagrange polys

Definition

If $x_1, \ldots, x_n$ are n distinct numbers and $f : \mathbb{R} \to \mathbb{R}$, then a unique polynomial $P_n(x)$ of degree at most $n-1$ exists with $f(x_k) = P(x_k)$, for each $k = 1, \ldots, n$. This polynomial is called the *Lagrange polynomial*. Explicitly, $P_n(x) := \sum_j f(x_j) \prod_{i, j \neq i} \frac{x - x_i}{x_j - x_i}$.

Theorem (Lagrange error formula)

Let f be n times differentiable. Then for all x, $|f(x) - P_n(x)| \leqslant \frac{\prod (x - x_k)}{n!} \sup |f^{(n)}|$.

Proof uses generalized Rolle.

Generalized Rolle

Theorem (Classical Rolle's theorem)

Let f be differentiable and have two zeroes in an interval $[a, b]$. Then f' has a zero in (a, b).

Theorem (Classical generalized Rolle's theorem)

Let f be n times differentiable and have $n + 1$ zeroes in an interval $[a, b]$. Then $f^{(n)}$ has a zero in $[a, b]$.

Is not constructive.

Generalized Rolle

Three solutions:

- Approximate (ϵ) version
- Generic zeroes using sheaf models
- **Divided differences** (Thanks Henri)

Hermite-Genocchi formula

Let R be a field and $f : R \to R$. The interpolation polynomial in the Newton form is a linear combination of Newton basis polynomials

$$N(x) := \sum_{j=0}^{k} a_j n_j(x)$$

with the Newton basis polynomials defined as

$$n_j(x) := \prod_{i=0}^{j-1} (x - x_i)$$

and the coefficients defined as $a_j := f[x_0, \ldots, x_j]$, where $f[x_0, \ldots, x_j]$ is the notation for divided differences:

Hermite-Genocchi formula

divided differences defined recursively by:

$$f[a] = f(a)$$

$$f[a, b] = f(a) - f(b)/a - b$$

$$f[a, b, c] = f[a, c] - f[b, c]/a - b$$

and in general, $f[a : b : l] := f[a : l] - f[b : l]/a - b$.
Thus the Newton polynomial can be written as

$$N(x) := f[x_0]+f[x_0, x_1](x-x_0)+\cdots+f[x_0, ..., x_k](x-x_0)(x-x_1)\cdots(x-x_{k-1})$$

Hermite-Genocchi formula

The Newton polynomial coincides with the Lagrange polynomial.
The divided difference $f[a_1, \ldots, a_n]$ is the coefficient of x^n in the (Newton) polynomial that interpolates f at $a_1, \ldots, a_n$.

Hermite-Genocchi formula

The Newton polynomial coincides with the Lagrange polynomial. The divided difference $f[a_1, \ldots, a_n]$ is the coefficient of x^n in the (Newton) polynomial that interpolates f at $a_1, \ldots, a_n$.

$$f[a, b] = \int_0^1 f'(a + (b - a)t)\,\mathrm{dt}\,.$$

Generally,

$$f[a_1, ..., a_n] = \iint_{n-1} f^{(n-1)}(u_1 a_1 + ... + u_n a_n) du_1 \cdots du_{n-1}$$

with $u_1 + \cdots + u_n = 1$ and $0 \leqslant u_i \leqslant 1$.

Hermite-Genocchi formula

The Newton polynomial coincides with the Lagrange polynomial. The divided difference $f[a_1, \ldots, a_n]$ is the coefficient of x^n in the (Newton) polynomial that interpolates f at $a_1, \ldots, a_n$.

$$f[a, b] = \int_0^1 f'(a + (b - a)t)\,\mathrm{d}t\,.$$

Generally,

$$f[a_1, ..., a_n] = \iint_{n-1} f^{(n-1)}(u_1 a_1 + ... + u_n a_n) du_1 \cdots du_{n-1}$$

with $u_1 + \cdots + u_n = 1$ and $0 \leqslant u_i \leqslant 1$.Corollary,

$$f(x) - P_n f(x) = \prod_{i=1}^{n}(x - x_i) \iint_{n-1} f^{(n-1)}(u_1 a_1 + ... + u_n a_n) du_1 \cdots du_{n-1}$$

Hermite-Genocchi formula

The Newton polynomial coincides with the Lagrange polynomial. The divided difference $f[a_1, \dots, a_n]$ is the coefficient of x^n in the (Newton) polynomial that interpolates f at $a_1, \dots, a_n$.

$$f[a, b] = \int_0^1 f'(a + (b - a)t)\,\mathrm{dt}\,.$$

Generally,

$$f[a_1, ..., a_n] = \iint_{n-1} f^{(n-1)}(u_1 a_1 + ... + u_n a_n) du_1 \cdots du_{n-1}$$

with $u_1 + \cdots + u_n = 1$ and $0 \leqslant u_i \leqslant 1$.Corollary,

$$f(x) - P_n f(x) = \prod_{i=1}^{n}(x - x_i) \iint_{n-1} f^{(n-1)}(u_1 a_1 + ... + u_n a_n) du_1 \cdots du_{n-1}$$

Replace differentiation by integration.

Simpson's rule

Corollary (Simpson's rule)

If $|f^{(4)}| \leqslant M$, *then*

$$|\int_a^b f(x)\,dx - \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right]| \leqslant \frac{(b-a)^5}{2880}M.$$

The right hand side is the integral of the Lagrange polynomial P_3 at $a, \frac{a+b}{2}, b$. For the error we adopt the classical proof, but replace the use of Rolle's theorem and the Mean Value Theorem by the Hermite-Genocchi formula.

Simpson's rule

Corollary (Simpson's rule)

If $|f^{(4)}| \leqslant M$, *then*

$$|\int_a^b f(x)\,dx - \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right]| \leqslant \frac{(b-a)^5}{2880}M.$$

The right hand side is the integral of the Lagrange polynomial P_3 at $a, \frac{a+b}{2}, b$. For the error we adopt the classical proof, but replace the use of Rolle's theorem and the Mean Value Theorem by the Hermite-Genocchi formula.Define $F(t) := f(\frac{a+b}{2} + \frac{b-a}{2}t)$. This reduces the problem to showing that $|\int_{-1}^{1} F(\tau)d\tau - \frac{1}{3}(F(-1) + 4F(0) + F(1)]| \leqslant N/90$, where $|F^{(4)}| \leqslant N$

Simpson's rule

Define

$$G(t) = \int_{-t}^{t} F(\tau)d\tau - \frac{t}{3}(F(-t) + 4F(0) + F(t)]$$

We need to prove that $90G(1) \leqslant \|F^{(4)}\|$.

Simpson's rule

Define

$$G(t) = \int_{-t}^{t} F(\tau)d\tau - \frac{t}{3}(F(-t) + 4F(0) + F(t)]$$

We need to prove that $90G(1) \leqslant \|F^{(4)}\|$.To do so, define $H(t) := G(t) - t^5 G(1)$. Then

$$H(0) = H(1) = H'(0) = H''(0) = 0.$$

Simpson's rule

Define

$$G(t) = \int_{-t}^{t} F(\tau)d\tau - \frac{t}{3}(F(-t) + 4F(0) + F(t)]$$

We need to prove that $90G(1) \leqslant \|F^{(4)}\|$.To do so, define $H(t) := G(t) - t^5 G(1)$. Then

$$H(0) = H(1) = H'(0) = H''(0) = 0.$$

Hence, $H[0,0,0,1] = -(H[0,0,0] - H[0,0,1]) = 0 + (-H[0,0] + H[0,1]) = 0$.
Moreover, $H^{(3)}(t) = -\frac{t}{3}(F^{(3)}(t) - F^{(3)}(-t)) - 60t^2 G(1) = -\frac{t}{3}(\int_{-t}^{t} F^{(4)}) - 60t^2 G(1)$.

Simpson's rule

This shows that

$$\begin{aligned}
0 = H[0,0,0,1] &= \int_0^1 H^{(3)} \\
&= \int_0^1 -\frac{t}{3}\left(\int_{-t}^{t} F^{(4)}\right) - 60t^2 G(1) \\
&\geqslant \int_0^1 -\frac{t}{3} 2\,\mathrm{tN} - 60t^2 G(1) \\
&= -\frac{2}{3}(N + 90G(1)) \int_0^1 t^2 \\
&= -\frac{2}{3}(N + 90G(1))\frac{1}{3}.
\end{aligned}$$

Simpson's rule

This shows that

$$\begin{aligned}
0 = H[0,0,0,1] &= \int_0^1 H^{(3)} \\
&= \int_0^1 -\frac{t}{3}\left(\int_{-t}^{t} F^{(4)}\right) - 60t^2 G(1) \\
&\geqslant \int_0^1 -\frac{t}{3} 2\,\mathrm{tN} - 60t^2 G(1) \\
&= -\frac{2}{3}(N + 90G(1)) \int_0^1 t^2 \\
&= -\frac{2}{3}(N + 90G(1)) \frac{1}{3}.
\end{aligned}$$

Hence, $N \geqslant -90G(1)$. Similarly, $0 \leqslant -\frac{2}{9}(-N + 90G(1))$.
Consequently, $90G(1) \leqslant N$. We conclude that $|90G(1)| \leqslant N$.

Differentiation over general fields [Bertrand, Glöckner, Neeb]

> *The proofs are 'algebraic' in nature and in this way become often simpler and more transparent even than the usual proofs in $\mathbb{R}^n$ because we avoid the repeated use of the Mean Value Theorem (or of the Fundamental Theorem) which are no longer needed once they are incorporated in [the definition of the derivative by a difference quotient].*

Formalization

In progress

Some issues

- Reconstruction of corn
- Type classes for algebraic hierarchy, efficient data structures, bigops
- induction recursion

Reconstruction of corn

with Eelis van der Weegen
Corn: a library for constructive/computational analysis.
Plans:

- type classes for algebraic hierarchy, setoid rewrite, ring, faster data structures, ...
- ssreflect/intro patterns: robustness, discrete structures, ...
- machine integers, floats
- remove apartness?
- removing old stuff, old tactics
- ...

Removing apartness?

Current library looks strange, upside down.

```
Definition fun_strext := forall x y, f x # f y -> x # y.
```

We have added decidable setoids.

Equational algebraic theories should not need apartness, or decidability (?)

Removing apartness?

Current library looks strange, upside down.

```
Definition fun_strext := forall x y, f x # f y -> x # y.
```

We have added decidable setoids.
Equational algebraic theories should not need apartness, or decidability (?)
fact: **rational** was only used (but $2000\times$) to solve ring equations

Already done/in progress

Addition of ssreflect (moving target: coq trunk, ssreflect)
Removal of parts
rational removed in favor of ring
(However, **ring** does not work for type classes)

Already done/in progress

Addition of ssreflect (moving target: coq trunk, ssreflect)
Removal of parts
rational removed in favor of ring
(However, **ring** does not work for type classes)
Improved compilation: better dependency analysis
coq indenter
Unification of ring implementations
Organizing notations in a notation scope, better: canonical names using type classes

Program

program technology separates proofs and programs.
However, separation is very strict between programs (Type) and proofs (Prop).
This makes it difficult to write programs depending on proofs.
Currently we use a trick:

```
Inductive sq (A : Type) : Prop :=
  insq : A -> (sq A).
Axiom unsq : forall A : Type, (sq A) -> A.
```

Program

program technology separates proofs and programs.
However, separation is very strict between programs (Type) and proofs (Prop).
This makes it difficult to write programs depending on proofs.
Currently we use a trick:

```
Inductive sq (A : Type) : Prop :=
  insq : A -> (sq A).
Axiom unsq : forall A : Type, (sq A) -> A.
```

Coq encourages to use explicit as opposed to constructive mathematics.

Algebraic hierarchy using type classes

Challenge of the algebraic hierarchy (monoids, groups, rings, ...).
Ring is a monoid in two ways. How do we inherit?
Standard solution: record types

Algebraic hierarchy using type classes

Challenge of the algebraic hierarchy (monoids, groups, rings, ...).
Ring is a monoid in two ways. How do we inherit?
Standard solution: record types
is claimed to break down in the large
New solution: packed classes (ssreflect team) supports

- multiple inheritance
- maximal sharing of notations and theories
- automated structure inference

Algebraic hierarchy using type classes

Our solution: use type classes and records with strict separation between operations and properties.
Type classes allow to define a class of types, e.g. semigroups.
(experimental in Coq, by Sozeau)

Algebraic hierarchy using type classes

Our solution: use type classes and records with strict separation between operations and properties.
Type classes allow to define a class of types, e.g. semigroups.
(experimental in Coq, by Sozeau)
Properties are packed:

```
Class SemiGroup A {e: Equiv A} {op: SemiGroupOp A} :=
  { sg_eq:> Equivalence e
  ; sg_ass:> Associative sg_op
  ; sg_mor:> Proper (e ==> e ==> e)%signature sg_op }.
```

Small terms instead of small contexts. No big problems yet.

Algebraic hierarchy using type classes

```
Class SemiRing A {e: Equiv A}{plus: RingPlus A}
  {mult: RingMult A}{zero: RingZero A}{one:RingOne A}:Prop:=
  { semiring_mult_monoid:> Monoid A (op:=mult)(unit:=one)
  ; semiring_plus_monoid:> Monoid A (op:=plus)(unit:=zero)
  ; semiring_plus_comm:> Commutative plus
  ; semiring_mult_comm:> Commutative mult
  ; semiring_distr:> Distribute mult plus
  ; mult_0_l: forall x, 0 * x == 0 }.
```

efficient data types

N as initial semiring, uses our formalization of universal algebra.
Z as initial ring.
We encourage more efficient implementations: the standard implementation has low priority:

```
Instance: Params (@sr_precedes) 7.
```

We will provide machine integers as model.

bigops

bigops (Σ, Π) have been implemented (by ssreflect team).
We are working on another implementation using monoid morphisms from the list monad (the free monoid)

Induction recursion

Similtanuous induction-recursion:

```
DList : Type
fresh : DList -> X -> Prop

dnil: DList
dcons : (x:X)(xs:DList)(p:fresh xs x)DList

fresh dnil y = True
fresh (dcons x xs p) y = ((x # y) /\ (fresh xs y)
```

Allowed in agda, but not in Coq. Use Nodup lists instead.

Future: Implementing Picard iteration

Consider an infinitely often differentiable function, say $\lambda x.sin(sinx)$
To compute the integral we need an upper bound on the derivative.
We can use Cruz-Filipe's tactic to obtain the derivative and a proof that it is the derivative.
Then we can the sup function (O'Connor/S) to obtain a bound.
Finally, we apply Simpson's rule.
Interesting combination of proof techniques.