CMSC631: Project Report

Shrey Patel, UID: 120210242

December 2024

Project Overview: Coq for Numerical Analysis

Goal: To study how Coq can be used for formal verification of numerical algorithms for solving differential equations.

Accomplished Goals

- Survey and understanding of real analysis libraries:
  1. **Coq STL:**
     - Set of real numbers `R`
     - Operators `+, *` with identities `R0, R1`
     - Axioms on real numbers: `total_order, completeness`, etc.
     - Lemmas for operators, orders, transcendental functions on reals (e.g. trigonometric, exponential, etc.)
  2. **Coquelicot[2]:**
     - Extension of Coq STL to support differential and integral calculus
     - Subsets defined as predicates on sets: `T -> Prop`
     - Neighbourhoods defined as filters: `(T -> Prop) -> Prop`
     - Limits defined as maps between neighbourhoods:
       `(T -> U) -> ((T -> Prop) -> Prop) -> ((U -> Prop) -> Prop)`
     - Derivatives as maps between functions: `(T -> U) -> ((T -> Prop) -> Prop) -> (T -> U) -> Prop`
     - Lemmas for derivatives, integrals, Taylor Series, etc.
- Defined and proved various helper lemmas on real numbers (in code file `Base.v`):

```
Lemma Rplus_lt_r : forall r r' : R, (0 < r') → (r < r + r').
Lemma Rplus_le_r : forall r r' : R, (0 <= r') → (r <= r + r').
Lemma Rabs_scalar : forall x y : R, (0 < x) → (Rabs (x * y) = x * Rabs y).
```

- Defined an ordinary differential equation for exponential decay (in code file `ODE.v`):
  - **Differential Equation:** $\frac{dy}{dt} = -\lambda y, \;\; y(t_0) = y_0$
  - **Coq definition:** As a relation of type `R -> (R -> R) -> Prop`:

```
Definition is_differentiable (f: R → R): Prop :=
forall (x: R) (n: nat),
ex_derive_n f n x.

Definition exp_ode (lambda : R) (y : R → R) :=
(is_differentiable y) ∧
(forall t : R, Derive_n y 1 t = − (lambda * (y t))).
```

  - Defined and proved a lemma for double derivative:

```
Lemma double_deriv : forall lambda zeta : R, forall y : R → R,
(forall t : R, Derive_n y 1 t = − (lambda * y t)) →
Derive_n y 2 zeta = lambda * lambda * y zeta.
```

- Implemented a numerical algorithm for solving the above differential equation (in code file `NumericalMethod.v`):
  - **Forward Euler:** $y_{n+1} = (1 - \lambda\, \Delta t) y_n$
  - **Coq definition**:

```
Definition euler_step (dt : R) (lambda : R) (yn : R) : R := (yn * (1 - dt * lambda)).

Fixpoint euler (y0 : R) (lambda : R) (dt : R) (n : nat) : R :=
match n with
| 0%nat => y0
| S n'  => euler_step dt lambda (euler y0 lambda dt n')
end.
```

- Proved a theorem on local error bound (in code file `LocalError.v`)
  - **Theorem:** Let the exact solution be y and the numerical solution be $\hat{y}$. Then, assuming that at time t_n, both the exact and the numerical solution agree (i.e. $y(t_n) = \hat{y}_n$), the error introduced by the numerical solution in one time step after t_n is bounded by the factor $\left\|\frac{y_0(\lambda\Delta t)^2}{2}\right\|$, where y_0 is the initial value of y, i.e. $y_0 = y(t_0)$. In short, $|y(t_n + \Delta t) - \hat{y}_{n+1}| \leq \left\|\frac{y_0(\lambda\Delta t)^2}{2}\right\|$
  - **Coq definition:**

```
Theorem local_error_bounded :
forall y : R -> R,
forall lambda t0 tn dt : R,
0 < lambda ->
0 < dt ->
t0 <= tn ->
0 < (lambda * dt) < 1 ->
exp_ode lambda y ->
(Rabs ((y (tn + dt)) - (euler_step dt lambda (y tn))))
<= ((lambda * dt)^2 * (Rabs (y t0))) / INR 2.
```

- Proved a theorem on global error bound (in code file `GlobalError.v`)
  - **Theorem:** Let the exact solution be y and the numerical solution be $\hat{y}$. Then, after starting from the same initial state y_0 at time t_0, the error after n time steps (i.e. at time $t_n = t_0 + n\,\Delta t$) is bounded by the factor $\left\|\frac{n y_0(\lambda\Delta t)^2}{2}\right\|$, meaning that the error grows linearly in worst case for the chosen numerical algorithm. In short, $|y(t_0 + n\,\Delta t) - \hat{y}_n| \leq \left\|\frac{n y_0(\lambda\Delta t)^2}{2}\right\|$
  - **Coq definition:**

```
Theorem global_error_bounded :
forall y : R -> R,
forall lambda t0 dt : R,
0 < lambda ->
0 < dt ->
0 < (lambda * dt) < 1 ->
exp_ode lambda y ->
forall n : nat,
(Rabs ((y (t0 + (INR n) * dt)) - (euler (y t0) lambda dt n)))
<= INR n * (((lambda * dt)^2 * (Rabs (y t0))) / INR 2).
```

Unaccomplished Goal:

The proof for local error uses a lemma (defined in code file `ODE.v`) that any function of type `R -> R` which satisfies the differential equation `exp_ode` defined above for a given value of λ and initial value y_0 must be of the form `exp_ode_exact` defined below:

```
Definition exp_ode_exact (lambda y0 : R) := fun (t : R) ⇒ (y0 ∗ (exp (− (lambda ∗ t)))).

(* Theorem to prove that the exact solution is the only solution of the given ODE *)
Theorem exp_eqv : forall lambda t0 : R, forall y : R → R,
exp_ode lambda y → y = exp_ode_exact lambda (y t0).
```

This lemma is crucial for expressing the local and global error bounds in terms of the initial value of the function y_0. It is analytically provable using integration. Integration in Coquelicot library is implemented using Reimann sums, which I have not been able to understand properly, so I have not been able to complete the proof of this lemma.

References

1. Sylvie Boldo, Catherine Lelay, Guillaume Melquiond. Improving Real Analysis in Coq: A User-Friendly Approach to Integrals and Derivatives. 2012.

2. Sylvie Boldo, Catherine Lelay, Guillaume Melquiond. Coquelicot: A user-friendly library of real analysis for Coq. 2014.

3. Ariel Kellison and Andrew Appel. Verified Numerical Methods for Ordinary Differential Equations. 2022.