Verified Numerical Methods for Ordinary Differential Equations

Ariel E. Kellison[1] and Andrew W. Appel[2]

[1] Cornell University
[2] Princeton University **DRAFT May 10, 2022**

Abstract. Ordinary differential equations (ODEs) are used to model the evolution of the state of a system over time. They are ubiquitous in the physical sciences, and often used in computational models with safety-critical applications. For critical computations, numerical solvers for ODEs that provide useful guarantees of their accuracy and correctness are required, but do not always exist in practice. In this work, we demonstrate how to use the Coq proof assistant to verify that a C program correctly and accurately finds the solution to an ODE initial value problem (IVP). Our verification framework is modular, and concisely disentangles the high-level mathematical properties expected of the system being modeled from the low-level behavior of a particular C program. Our approach relies on the construction of two simple functional models in Coq: a floating-point valued functional model for analyzing the intermediate-level behavior of the program, and a real-valued functional model for analyzing the high-level mathematical properties of the system being modeled by the IVP. Our final result is a proof that the floating-point solution returned by the C program is an accurate solution to the IVP, with a good quantitative bound. Our framework assumes only the operational semantics of C and of IEEE-754 floating point arithmetic.

1 Introduction

Computing accurate solutions to differential equations is a main topic in the field of numerical analysis. A typical problem in ordinary differential equations requires computing the numerical solution to autonomous initial-value problems (IVPs) of the form

$$\frac{dx}{dt} = f(x), \quad x(t_0) = x_0 \tag{1}$$

at some time $t \in [t_0, T]$ to within a user-specified error tolerance. In this paper, our objective is to demonstrate a logical framework for verifying the accuracy and correctness of numerical programs that compute the solution to problems of the form (1). This framework is most suitable for critical applications that require guarantees of numerical accuracy (i.e., the numerical solution does not exceed the user-specified error tolerance) and program correctness (i.e., the particular implementation is bug-free and meets its specification). Our main result is a machine-checked theorem stating that a specific imperative implementation of a numerical method for the solution to an IVP produces a solution within a guaranteed error bound of the true solution, where the error bound accounts for two sources of error: discretization error and round-off error. We obtain this machine-checked theorem using a modular, layered approach to program verification that allows us to treat program correctness and each source of error separately within one logical framework, namely, the Coq proof assistant. For the results presented in this work, we have chosen the simple harmonic oscillator as an elementary but sufficiently illustrative example of an initial-value problem[3]; for the numerical solution to this IVP, we consider a C implementation of the Störmer-Verlet ("leapfrog") method.

In contrast to *validated* numerical methods, which have a long history of deriving guaranteed error bounds for IVPs for ODEs [1–3], the framework for *verified* methods presented here has three distinct advantages for critical applications. First, verified numerical methods introduce no additional computational overhead at run-time, whereas validated methods typically require considerably more computational time than standard methods. Second, the modular approach of the verification framework allows each source of error (e.g, discretization error, round-off error, data error, and implementation bugs) to be treated separately, enabling users to easily identify or emphasize areas of concern in their method or program. Finally, our verification framework connects guaranteed error bounds to low-level properties of an implementation (in C or below), providing assurance beyond the scope of validated methods.

To obtain a correctness-and-accuracy theorem that connects guaranteed error bounds to the low-level correctness of a C implementation of the leapfrog method, we *layer* the verification using several tools and libraries that are fully integrated into the Coq proof assistant. In particular, we prove that the C program refines a functional model using VST [4], and (separately) prove that the functional model has the desired properties using the Coquelicot formalization of real analysis [5], the Coq Interval [6] package, and VCFloat2 [7, 8].

Our Coq development is available at `github.com/VeriNum/VerifiedLeapfrog`

2 Main Result

Our main objective is to verify that a C implementation of leapfrog integration (given in figure 1.1) is correct, and that it accurately solves the system of ordinary differential equations for the simple harmonic oscillator in $\mathbb{R}^2$ to within an accuracy `acc` at time T. In particular, we consider the system of equations

$$\frac{dp}{dt} = -\omega^2 q, \quad \frac{dq}{dt} = p, \tag{2}$$

where ω, p, and q are, respectively, the frequency, momentum, and position of the oscillator. To indicate that two functions $p : \mathbb{R} \to \mathbb{R}$ and $q : \mathbb{R} \to \mathbb{R}$ with

[3] This particular model problem admits an analytical solution and is therefore not expected to be of practical interest on its own; it is chosen for demonstrating and analyzing the performance of our logical framework.

initial conditions $p(t_0) = p_0$ and $q(t_0) = q_0$ constitute the continuous system (2) we use the predicate Harmonic_oscillator_system ω t_0 p_0 q_0 p q:

Definition F (x ω : $\mathbb{R}$) : $\mathbb{R}$:= $-\omega^2 \cdot$ x.
Definition Harmonic_oscillator_system (ω t_0 p_0 q_0 : $\mathbb{R}$) (p q : $\mathbb{R} \to \mathbb{R}$) : Prop :=
$p(t_0) = p_0 \wedge q(t_0) = q_0 \wedge$ smooth_fun p $\wedge$ smooth_fun q
$\wedge\ \forall\ t$, (Derive_n q 1 $t = p$ t $\wedge$ Derive_n p 1 t = F($q(t)$, ω)
$\wedge\ \|(p(t),\ \omega q(t))\|_2 = \|(p(t_0),\ \omega \cdot q(t_0))\|_2$).

where (smooth_fun f) indicates that f is continuously differentiable, i.e.,

Definition smooth_fun (f: R $\to$ R) := $\forall$ (x: R) (n: nat), ex_derive_n f n x.

and Derive_n is the Coquelicot abstraction for the nth derivative of f at x.

An integer-step leapfrog discretization of the continuous system (2) on a time interval $[0, T]$ uniformly partitioned by a fixed time step h with unit frequency $\omega = 1$ updates the position q and momentum p of the oscillator as

$$q_{n+1} = q_n + hp_n - \frac{h^2}{2} q_n \tag{3}$$

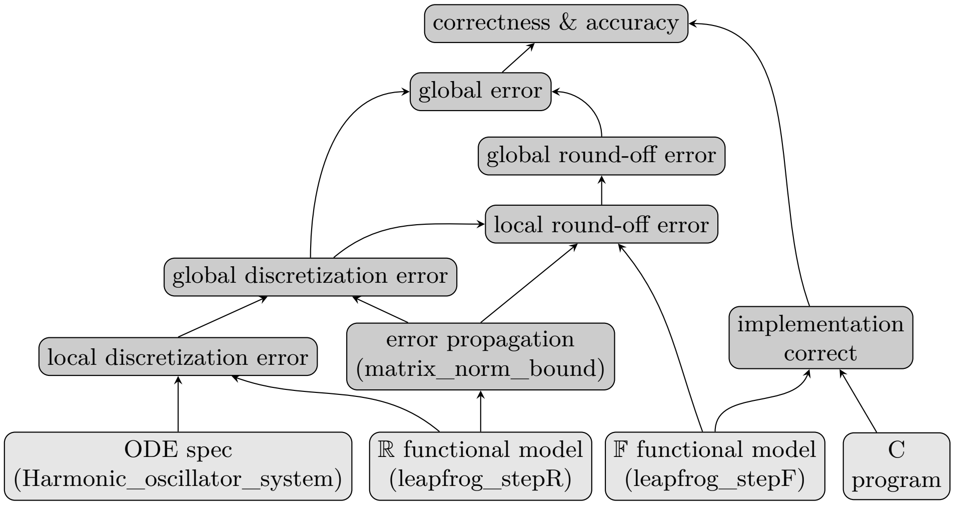
$$p_{n+1} = p_n - \frac{h}{2}(q_n + q_{n+1}). \tag{4}$$

```c
struct state {float p, q;};

float force(float q) { return -q; }

void integrate(struct state *s) {
   int n, N=1000; float t, h = 1.0f / 32.0f;

   s->q = 1.0f; s->p = 0.0f; t = 0.0f;

   for (n = 0; n < N; n++) {
       float a = force(s->q);
       s->q = s->q + h * s->p + (0.5f * (h * h)) * a;
       s->p = s->p + (0.5f * h) * (a + force(s->q));
       t = t + h;
}   }
```

Program 1.1. Leapfrog integration of the harmonic oscillator implemented in C with time step $h = \frac{1}{32}$, frequency $\omega = 1$, initial conditions $(p_0, q_0) = (0, 1)$.

If we define the *global error* at the nth time step $t_n = nh \leq T$ of leapfrog integration as the residual between the ideal solution $(p(t_n), q(t_n))$ and the numerical solution (p_n, q_n), i.e., $E_n = \|(p(t_n), q(t_n)) - (p_n, q_n)\|$, then the C implementation of equations (3 - 4) is accurate if it has global error $E_n \leq$ acc.

We prove the accuracy and correctness of the C implementation by composing several proofs: that the C program correctly implements a floating-point functional model; that in each iteration the floating-point functional model accurately approximates a real-valued functional model; that in each iteration the real-valued model accurately approximates the continuous ODE; that per-iteration errors are uniformly bounded by a propagation factor; and that the global propagation of per-iteration errors is bounded above by the desired accuracy. The main theorem then proves, from the composition of all of these theorems, and assuming only the operational semantics of C and of IEEE-754 floating point arithmetic, that the floating-point solution returned by Program (1.1) is an accurate solution to the ODE, with a good quantitative bound.



correctness & accuracy
global error
global round-off error
local round-off error
global discretization error
local discretization error
error propagation (matrix_norm_bound)
implementation correct
ODE spec (Harmonic_oscillator_system)
ℝ functional model (leapfrog_stepR)
𝔽 functional model (leapfrog_stepF)
C program

Fig. 1. Theorem dependency.

```
Definition F (x : 𝔽): 𝔽 :=−x.
Definition leapfrog_stepF
(h: 𝔽) (ic : 𝔽 × 𝔽) : 𝔽 × 𝔽 :=
  let p := fst ic in let q := snd ic in
  let q′ := (q+ h·p)+(1/2·(h· h))·F(q) in
  let p′ := p+(1/2·h)·(F(q)+F(q′)) in
  (p',  q').

Fixpoint iternF
(h: 𝔽) (ic: 𝔽²) (n: ℕ) : 𝔽² :=
  match n with
  | 0     ⇒ ic
  | S n′ ⇒
    iternF  h (leapfrog_stepF h ic) n′
  end.
```

```
Definition leapfrog_stepR
(h : ℝ) (ic : ℝ × ℝ ) : ℝ × ℝ :=
  let q := snd ic in let p := fst ic in
  let q′ := (1 − h²/2)·q + h·v in
  let p′ := (1 − h²/2)·p − h/2·(2 − h²/2)·q in
  (p',  q').

Fixpoint iternR
(h: ℝ) (ic : ℝ²) (n: ℕ) : ℝ² :=
   match n with
   | 0     ⇒ ic
   | S n′ ⇒
     iternR  h (leapfrog_stepR h ic) n′
   end.
```

We encapsulate the expected intermediate floating-point behavior of the C function integrate from Program 1.1 on input $(p, q) = ic \in \mathbb{F}^2$ with time-step h and frequency $\omega = 1$ using a floating-point valued functional model (leapfrog_stepF h ic). To reason about the behavior of leapfrog integration in exact arithmetic, we define a real-valued functional model (leapfrog_stepR h ic). Iterations of leapfrog_stepF and leapfrog_stepR are defined as iternF and iternR.

Henceforth we assume $\omega = 1$ and we omit it.

We use the predicate (accurate_harmonic_oscillator acc x n) to indicate that the single-precision floating-point valued momentum-position pair x differs by at most acc from the true position and momentum (at time $T = Nh$) of the ideal system defined by equation 2. Then, with x being the result computed by the C program, the C program specification is stated as

```
Definition integrate_spec :=
  DECLARE _integrate
  WITH s: val
  PRE [ tptr t_state ]  PROP()   PARAMS(s)   SEP(data_at_ Tsh t_state s)
  POST [ tvoid ]        EX (x: 𝔽 × 𝔽), PROP(accurate_harmonic_oscillator acc x N)
                                RETURN()   SEP(data_at Tsh t_state x s).
```

The precondition and postcondition are assertions about any C value s that is the address of a struct state. In particular,

PRE: The precondition asserts that the function parameter (of type pointer-to-struct-state) does indeed contain the value s and that the “data at” that location is uninitialized (or is initialized but we don’t care).

POST: The postcondition asserts that a pair x of single-precision floating-point values that are an accurate solution to the ODE are stored at address s.

If the C function satisfies this specification, then it correctly implements an accurate numerical integration of the ODE, which is our desired main result. We denote the C function’s abstract-syntax tree as f_integrate and prove the main theorem body_integrate, which guarantees that f_integrate satisfies the specification integrate_spec:

Theorem body_integrate : semax_body Vprog Gprog f_integrate integrate_spec.

In the remainder of the paper, we present the modular proofs of accuracy and correctness that are composed to derive this main result.

3 Verified Error Bounds

For a given accuracy acc, time step h, initial condition (p_0, q_0), and final time $T = Nh$, our goal is to prove that the solution $(\hat{p}_N, \hat{q}_N) \in \mathbb{F}^2$ obtained by the C implementation of equations (3 - 4) given in Figure 1.1, has global error E_N bounded above by acc:

$$E_N = \|(p(t_N), q(t_N)) - (\hat{p}_N, \hat{q}_N)\| \leq \texttt{acc}. \tag{5}$$

We derive a verified upper bound for E_N by considering separately the global discretization error and global round-off error. If we denote the numerical solution in ideal arithmetic at time t_N as $(\tilde{p}_N, \tilde{q}_N)$, then an upper bound on the global error is

$$\begin{aligned} E_N &= \|(p(t_n), q(t_N)) - (\hat{p}_N, \hat{q}_N)\| \\ &\leq \underbrace{\|(p(t_N), q(t_N)) - (\tilde{p}_N, \tilde{q}_N)\|}_{\text{global discretization error}} + \underbrace{\|(\tilde{p}_N, \tilde{q}_N) - (\hat{p}_N, \hat{q}_N)\|}_{\text{global round-off error}} = D_N + R_N. \end{aligned} \tag{6}$$

We obtain upper bounds for the global discretization error D_N and global round-off error R_N by first estimating the maximum possible *local error* from each source, and then upper-bounding the propagation of the local errors as the iterations advance using the 2-norm of the transition matrix of the leapfrog updates to position and momentum.

The local error associated with a numerical method is the residual between the ideal solution and the numerical solution after a single time step of size h starting from the same initial point. When estimating the local discretization error we take the ideal solution as the continuous system (Harmonic_oscillator_system) and the numerical solution as leapfrog integration carried out in exact arithmetic (leapfrog_stepR); and when estimating the local round-off error we take the ideal solution as leapfrog integration carried out in exact arithmetic (leapfrog_stepR) and the numerical solution as leapfrog integration carried out in single precision floating-point arithmetic (leapfrog_stepF).

To bound the global errors R_N and D_N, we must be able to invoke our local error theorems at any time step $0 \leq n \leq N$. To do this, we conservatively estimate the local error as τ such that $\sup_{n \in [N]} \|(p(t_n), q(t_n)) - (p_n, q_n)\| \leq \tau$. We derive τ_d for local discretization error and τ_r for local round-off error using the fact that the momentum p and position q do not grow too large. For the initial conditions of our model problem, $\|(p_0, q_0)\| = 1$. In the continuous Harmonic_oscillator_system one can prove (even without solving the ODE) that $\|(p(t), q(t))\| = \|(p_0, q_0)\|$ for all t. Unfortunately, that conservation property does not hold *exactly* in leapfrog_stepR or leapfrog_stepF, but we prove bounds on the growth of $\|(p(t), q(t))\|$—see Section 3.2.

The exact conservation of $\|(p(t), q(t))\|$ in our model problem is useful for deriving tight bounds on local errors, but it is not a requirement for our method. The error analysis will apply to IVPs of the form (1) provided that the numerical method used for integration is provably (adequately) *stable* [9], i.e., numerical solutions x_n to (1) satisfy $\|x_{n+1}\| \approx \|x_n\|$ for all n.

3.1 Local Discretization Error

Local discretization error is estimated as the residual difference between the exact solution characterized by Harmonic_oscillator_system and the numerical solution computed in exact arithmetic by (leapfrog_stepR) starting from the point $(p(t_n), q(t_n)) \in \mathbb{R}^2$ after a single time step of size h. We prove that the local discretization error is bounded, for all t, by $\tau_\mathrm{d} = h^3 \|(p(t_0), q(t_0))\|$.

Theorem local_discretization_error
  ∀ (p q : $\mathbb{R} \to \mathbb{R}$) ($t_0$ t_n h : $\mathbb{R}$), 0 < h ≤ 4 →
  let ω := 1 **in** Harmonic_oscillator_system ω t_0 $p(t_0)$ $q(t_0)$ p q →
  let (p_n, q_n) := leapfrog_stepR h $(p(t_n), q(t_n))$ **in**
  $\|(p(t_n+h), q(t_n+h)) - (p_n, q_n)\| \ \leq \ h^3 \, \|(p(t_0), q(t_0))\|$.

Proof. We expand the ideal solution of the harmonic oscillator $(p(t_n+h), q(t_n+h))$ as Taylor expansions around t_n using the Taylor_Lagrange theorem from the Coquelicot library [5] and use the derivative relations for p and q from Harmonic_oscillator_system to derive the differences

$$|p(t_n+h) - p_n| = h^3 \left| \frac{p(\eta_1)}{4!} - \frac{q(t_n)}{12} \right|, \tag{7a}$$

$$|q(t_n+h) - q_n| = \frac{h^3}{3!} \, |p(\eta_2)| \tag{7b}$$

for some $t_n < \eta_1, \eta_2 < t_n + h$. Recall that $\|(p(t), q(t))\| = \|(p(t_0), q(t_0))\|$ is a property of our model problem. Provided that $0 < h \leq 4$, it then follows that

$$\|(p(t_n), q(t_n)) - (p_n, q_n)\| \leq \tau_{\mathrm{d}} = h^3 \ \ \|(p_0, q_0)\| \,. \tag{8}$$

We will see in the next section that the restriction $h \leq 4$ is not overly restrictive.

3.2 Propagation of Errors

To bound the propagation of local errors over n iterations, we use the 2-norm of the transition matrix of leapfrog updates to position and momentum [10–12]. In particular, if we represent the leapfrog method for the evolution of the harmonic oscillator as the transition matrix $M(h) : (p_n, q_n) \mapsto (p_{n+1}, q_{n+1})$

$$M(h) = \begin{pmatrix} 1+\zeta & \frac{\zeta}{h}(2+\zeta) \\ \\ h & 1+\zeta \end{pmatrix}, \text{ with } \quad \zeta = -\frac{h^2}{2}. \tag{9}$$

Then the evolution over n steps is denoted as applications of powers of the transition matrix $M(h)$ to the initial conditions p_0 and q_0:

$$\begin{pmatrix} p_n \\ q_n \end{pmatrix} = (M(h))^n \begin{pmatrix} p_0 \\ q_0 \end{pmatrix}. \tag{10}$$

An upper bound for the global error e_n (where e_n could be either R_n or D_n) at step n can be decomposed into two parts: the local error at step n and the propagation of accumulated errors from previous steps.

$$\begin{aligned} e_n &= \|(p(t_n), q(t_n)) - (p_n, q_n)\| = \|(p(t_n), q(t_n)) - M(h)(p_{n-1}, q_{n-1})\| \qquad (11) \\ &\leq \underbrace{\|(p(t_n), q(t_n)) - M(h)(p(t_{n-1}), q(t_{n-1}))\|}_{\text{local error at step } n} + \\ &\qquad\qquad \underbrace{\|M(h)(p(t_{n-1}), q(t_{n-1})) - M(h)(p_{n-1}, q_{n-1})\|}_{\text{propagation of prior local errors}} \\ &\leq \tau + \|M(h)\| \, \|(p(t_{n-1}), q(t_{n-1})) - (p_{n-1}, q_{n-1})\| \,. \end{aligned}$$

We can therefore estimate an upper bound for e_n from an appropriate estimate for the local error at step n and a reasonable approximation of $\|M(h)\|$. (This is the typical result that *convergence = stability + consistency.*)

To use equation (11) in our verified error analysis, we define $M(h)$ using the Coquelicot matrix library. We define the operator 2-norm for matrices in $\mathbb{R}^{n\times n}$ as

$$\|A\|_2 := \inf\{s : \mathbb{R}_+ \mid \forall v \in \mathbb{R}^n,\ \|Av\|_2 = s\,\|v\|_2\}, \tag{12}$$

and use the predicate two_norm_pred to indicate that the real number σ is the 2-norm of the $n \times n$ matrix A:

Definition two_norm_pred (n: $\mathbb{N}$) (A : matrix $\mathbb{C}$ n n) (σ: $\mathbb{R}$) : Prop :=
$\forall$ (u : vector $\mathbb{C}$ n), $\|Au\| \leq \sigma\,\|u\| \wedge (\neg\exists$ (s : $\mathbb{R}$), $\forall$ (x : vector $\mathbb{C}$ n), $\|Ax\| \leq s\,\|x\| < \sigma\,\|x\|$).

We prove (but do not present here) that this predicate is satisfied for any matrix $A \in \mathbb{R}^{2\times 2}$ by the maximum singular value of A. $\|M(h)\|$ is therefore defined as the positive real number $\sigma(h)$ in (two_norm_pred 2 ($M(h)$) ($\sigma(h)$)) such that $\sigma(h)$ is the square root of the maximum eigenvalue of the matrix $B = \overline{M}(h)^T M(h)$:

Definition σ (h : $\mathbb{R}$) : $\mathbb{R}$:=
  let a:= $\sqrt{h^6 + 64}$ **in**
  let A:= $(h^{10} - h^7 a + 4h^6 + 64h^4 - 4h^3 a - 32ha + 256)(h^2 - 2)^2$ **in**
  $\sqrt{A/(2(h^4 - 4h^2 + 4)((h^3 - 8h + a)^2 + 16(h^2 - 2)^2))}$

It is clear that $\sigma(h)$ is only well defined for time steps h in the interval $0 < h < \sqrt{2}$. For values of h outside of this range, leapfrog integration of the harmonic-oscillator with unit frequency is *unstable*, which means that the growth of the error can no longer be uniformly bounded [13]. Since our time step is fixed by $h = \frac{1}{32}$ in our C program, we derive a verified bound on the solution vector (p_n, q_n) at step n for any initial (p_0, q_0) by proving the following theorem, which follows by induction on the iteration number and unfolding of the definition of the predicate for the 2-norm.

Theorem matrix_bound : $\forall$ (p_0 q_0: $\mathbb{R}$) (n : $\mathbb{N}$), $\|(M(h))^n (p_0, q_0)\| \ \leq\ (\sigma(h))^n\,\|(p_0, q_0)\|$.

A bound on leapfrog_stepR follows as a corollary. In particular, for $h = \frac{1}{32}$, we have $\sigma(h) \leq 1.000003814704543$, and therefore

Corollary method_norm_bound :
  $\forall$ p q: $\mathbb{R}$, $\|$(leapfrog_stepR (p,q) h)$\|$ $\leq$ $1.000003814704543\,\|(p, q)\|$.

Given that the C program (Figure 1.1) runs for $N = 1000$ iterations with the initial condition $(p(t_0), q(t_0)) = (0, 1)$, method_norm_bound guarantees that each component of the numerical solution (position and momentum) will be bounded by 1.00383 in absolute value; we use these bounds on the solution vector when deriving an upper bound on the local round-off error.

3.3 Global Discretization Error

Analyzing the recurrence in the term for the propagation of prior local errors in equation (11) over several iterations leads to the following estimate of an upper bound for the global discretization error.

$$D_N = \|(p(t_n), q(t_n)) - (\text{iternR h } (p(t_0), q(t_0)) \text{ n})\| \leq h^3 \, \|(p(t_0), q(t_0))\| \sum_{k=0}^{n-1} \sigma(h)^k.$$

We prove that this estimate holds by invoking our local error theorem and performing induction on the iteration step in the following theorem.

Theorem global_discretization_error :
∀ $(p\ q : \mathbb{R} \to \mathbb{R})$ $(t_0 : \mathbb{R})$, **let** $\omega := 1$ **in**
Harmonic_oscillator_system $p_0\ q_0\ \omega\ t_0\ p\ q \to$
∀ (n : $\mathbb{N}$), **let** $t_n := t_0 + nh$ **in**
$\|(p(t_n), q(t_n)) -$ (iternR h $(p(t_0), q(t_0))$ n)$\| \leq h^3 \, \|(p(t_0), q(t_0))\| \sum_{k=0}^{n-1} \sigma(h)^k$.

Given that the C program (Figure 1.1) runs for $N = 1000$ iterations with time step $h = \frac{1}{32}$ and initial condition $(p(t_0), q(t_0)) = (0, 1)$, the contribution from discretization error to the global error at $t = Nh$ is guaranteed to be at most $3.06 \cdot 10^{-2}$.

3.4 Local Round-off Error

Local round-off error is the residual difference between the numerical solution computed in exact arithmetic and the numerical solution computed in single-precision floating-point arithmetic after a single time step of size $h = \frac{1}{32}$ on the same input. We derive a bound on the maximum possible local round-off error for leapfrog integration of the harmonic oscillator using VCFloat [7, 8] and the Coq interval package [6]:

Theorem local_roundoff_error:
∀ x : state , boundsmap_denote leapfrog_bmap (leapfrog_vmap x) →
‖FT2R_prod(leapfrog_stepF h x) − leapfrog_stepR h (FT2R_prod x)‖ $\leq 1.399 \cdot 10^{-7}$.

The proof of local_roundoff_error is mostly automatic. We will not show the details here; see Ramanandro *et al.* [8] and Appel and Kellison [7]. The function FT2R_prod : $\mathbb{F} \times \mathbb{F} \to \mathbb{R} \times \mathbb{R}$ in local_roundoff_error injects floating-point pairs to real number pairs. The boundsmap_denote hypothesis enforces bounds on the components of the state vector $x = (p, q) \in \mathbb{F}^2$. In particular, we have constructed leapfrog_bmap to specify that $-1.0041 \leq p, q \leq 1.0041$.

Tighter bounds on p and q will result in a tighter round-off error bound. Initially, $\|(p_0, q_0)\| = 1$, so $-1 \leq p_0, q_0 \leq 1$. But as errors (from discretization and round-off) accumulate, the bounds on p, q must loosen.

In principle the leapfrog_bmap could be a function of n; the nth-iteration bounds on p, q could be used to calculate the nth-iteration round-off error. As discussed at the beginning of Section 3, we prove a local round-off error bound τ just once, in part because the VCFloat library requires the bounds to be constant values. So τ is basically the worst-case bound on p, q after the last iteration.

The looser the bounds, the worse the round-off error, therefore the looser the bounds. Fortunately the round-off error is only weakly dependent on the bounds on p and q, so we can cut this Gordian knot by choosing τ_{r} small enough to prove an adequately tight bound in local_roundoff_error and large enough to be proved from global_roundoff_error.

We derive $\tau_{\mathrm{r}} = 1.0041$ as follows. If there were no round-off error, then (according to theorem method_norm_bound), $\|(p, q)\|$ increases by (at most) a factor of 1.0000039 in each iteration, so over 1000 iterations that is (at most) 1.00383. The machine epsilon for single-precision floating point is $\epsilon = 1.19 \cdot 10^{-7}$. Assuming this error in (each component of) the calculation of $\|(p, q)\|$, then the norm of the floating-point solution for N iterations can be bounded as:

$$\begin{aligned} \|(\text{leapfrog\_stepF x})\| &\leq \|(\text{leapfrog\_stepR x}\| + \epsilon \sum_{k=0}^{N-1} \sigma(h)^k \\ &\leq \sigma(h)^N + \epsilon \sum_{k=0}^{N-1} \sigma(h)^k \leq 1.0041. \end{aligned} \tag{13}$$

Finally, note that the appropriate application of the function FT2R_prod has been elided in equation (13) for succinctness—e.g. $\|\text{leapfrog\_stepR}(x)\|$ should appear as $\|\text{FT2R\_prod}(\text{leapfrog\_stepR}(x))\|$—we will continue to omit this function in the remainder of the paper.

3.5 Global Round-Off Error

We estimate an upper bound for the global round-off error R_N by replicating the analysis for the global discretization error D_N given in Section 3.3.

Theorem global_roundoff_error :
boundsmap_denote leapfrog_bmap (leapfrog_vmap (p_0, q_0)) $\rightarrow$
$\forall$ (n: $\mathbb{N}$), $n \leq N \rightarrow$
boundsmap_denote leapfrog_bmap (leapfrog_vmap (iternF h (p_0, q_0) n))
$\wedge \;\|$ (iternR h (p_0, q_0) n) $-$ (iternF h (p_0, q_0) n) $\| \leq \tau_{\mathrm{r}} \sum_{k=0}^{n-1} \sigma(h)^k$.

Theorem global_roundoff_error provides a verified bound for global round-off error. It states that if the bounds required by the boundsmap_denote predicate (see Section 3.4) hold, then the solution (iternF h (p_0, q_0) n) obtained by single precision leapfrog integration over N iterations satisfies the bounds required by boundsmap_denote, and that the global round-off error for N iterations is upper bounded by the product of the maximum local round-off error and the sum of powers of the global error propagation factor (see Section 3.2).

The upper bound on $\|$ (iternR h (p_0, q_0) n) $-$ (iternF h (p_0, q_0) n) $\|$ follows by induction: if we define the floating-point solution after n steps of integration as

$(\hat{p}, \hat{q}) = (\text{iternF } h\ (p_0, q_0)\ n)$ then

$$
\begin{aligned}
&\|\text{iternR } h\ (p_0,q_0)\ (n+1) - \text{iternF } h\ (p_0,q_0)\ (n+1)\| \\
&\qquad = \|\text{iternR } h\ (p_0,q_0)\ (n+1) - \text{leapfrog\_stepF } (\hat{p},\hat{q})\| \\
&\qquad \le \underbrace{\|\text{iternR } h\ (p_0,q_0)\ (n+1) - \text{leapfrog\_stepR } (\hat{p},\hat{q})\|}_{\text{propagation of prior local errors}} + \\
&\qquad\qquad\qquad \underbrace{\|(\text{leapfrog\_stepR } (\hat{p},\hat{q}) - \text{leapfrog\_stepF } (\hat{p},\hat{q})\|}_{\text{local round-off error at step } (n+1)} \\
&\qquad \le \tau_{\mathrm{r}} \sum_{k=0}^{n} \sigma(h)^k.
\end{aligned}
\tag{14}
$$

From equation 14 it is clear that we must invoke local_roundoff_error in the proof of global_roundoff_error . To do so, we must show that $(\hat{p}, \hat{q})$ satisfy the boundsmap_denote predicate. To this end, we prove lemma itern_implies_bmd, which guarantees that the estimate in equation (13) is sufficient.

Lemma itern_implies_bmd:
$\forall$ (p q: $\mathbb{F}$) (n: $\mathbb{N}$), n + 1 $\le N \to$
boundsmap_denote leapfrog_bmap (leapfrog_vmap (iternF h (p,q) n)) $\to$
$\|$ (iternR h (p,q) $(n$+1)) $-$ (iternF h (p,q) $(n$+1)) $\| \le \tau_{\mathrm{r}} \sum_{k=0}^{n} \sigma(h)^k \to$
$\|$ (iternR h (p,q) $(n$+1))$\| \le \sigma(h)^N \to$
boundsmap_denote leapfrog_bmap (leapfrog_vmap (iternF h (p,q) $(n$+1))).

From global_roundoff_error we conclude that the contribution from round-off error to the global error at $t = Nh$ is guaranteed to be at most $1.4 \cdot 10^{-4}$.

3.6 Total Global Error

Using global_roundoff_error and global_discretization_error from Sections 3.1 and 3.4, we derive a verifed concrete upper bound for the total global error for single precision leapfrog integration of the harmonic oscillator over N time steps as

$$
\begin{aligned}
E_N &\le \underbrace{\|(p(t_N), q(t_N)) - (\tilde{p}_N, \tilde{q}_N)\|}_{\text{global discretization error}} + \underbrace{\|(\tilde{p}_N, \tilde{q}_N) - (\hat{p}_N, \hat{q}_N)\|}_{\text{global round-off error}} \\
&\le (\tau_{\mathrm{d}} + \tau_{\mathrm{r}}) \sum_{k=0}^{N-1} \sigma(h)^k \le 0.0308.
\end{aligned}
\tag{15}
$$

This bound is guaranteed by the following theorem, which uses the closed form expression for the geometric series in equation 15.

Theorem total_error:
$\forall$ (p_t q_t: $\mathbb{R} \to \mathbb{R}$) ($n$: $\mathbb{N}$), $n \le N \to$
let t_0 := 0 **in let** $t_n := t_0 + nh$ **in** $p_t(t_0) = p_0 \quad \to \quad q_t(t_0) = q_0 \to$
let ω := 1 **in** Harmonic_oscillator_system p_0 q_0 ω t_0 p_t q_t $\to$
$\|(p_t(t_n), q_t(t_n)) - (\text{iternF } h\ (p_0, q_0)\ n)\| \ \le\ (\tau_{\mathrm{d}} + \tau_{\mathrm{r}})(\sigma(h)^n - 1)/(\sigma(h) - 1)$.

In the following sections, we describe how the bound provided by total_error is composed with the refinement proof that C program implements the floating-point functional model to prove our main result.

4 Program Verification

The Verified Software Toolchain [14] is a program logic for C, with a soundness proof in Coq with respect to the formal operational semantics of C, and with proof automation tools in Coq for interactive verification.

When verifying programs in VST (or in other program logics), it is common to *layer* the verification: prove that the C program refines a functional model, and (separately) prove that the functional model has the desired properties. In this case, the functional model is our floating-point model defined by the functions leapfrog_stepF and iternF .

We showed a *high-level* specification for the integrate function of the C program (Figure 1.1) in Section 2; namely, that it accurately solves the ODE. Here we start with the *low-level* spec that the C program implements the floating-point functional model:

```
Definition integrate_spec_lowlevel :=
  DECLARE _integrate
  WITH s: val
  PRE [ tptr t_state ]
      PROP(iternF_is_finite) PARAMS (s) SEP(data_at_ Tsh t_state s)
  POST [ tvoid ]
      PROP() RETURN()
      SEP(data_at Tsh t_state (floats_to_vals (iternF h (p_init,q_init) N)) s).
```

This claims that when the function returns, the float values iternF h $(p_{\mathrm{init}}, q_{\mathrm{init}})$ N will be stored at location s—provided that (in the precondition) struct-fields s->p and s->q are accessible, and assuming iterF_is_finite .

The functional model is deliberately designed so that its floating-point operations adhere closely to the operations performed by the C program.[4] So the proof is almost fully automatic, except that the VST user must provide a loop invariant. In this case the loop invariant looks much like the function postcondition, except with the iteration variable n instead of the final value N.

The proofs of these functions are fairly short; see Figure 1. If the program had used nontrivial data structures or shared-memory threads, the functional model might be exactly the same but the C program would be more complex. The relation between the C program and the functional model would be more intricate. VST can handle such intricacy with more user effort.

[4] The only difference is that the C program has $t + h$ where the model has $h + t$. Because the commutative law is provable in floating point, we can accommodate this difference in the proof that the C program implements the functional model. We could not tolerate use of the associative law (which fails in floating-point).

We designed the functional model so that one can prove correctness of the C program without knowing (almost) anything about the properties of floating-point, and (completely) without knowing about the existence of the real numbers. One is simply proving that the C program does these float-ops, in this tree-order, without needing to know why.

C Program	C Lines	Proof Lines[5]	Proof Chars
force	3	2	25
lfstep	6	15	281
integrate	12	30	953

Table 1. VST proof effort

5 Combining the main theorems

We have shown how to prove,

```
Theorem total_error :  (* float  functional  model  integrates  the ODE of
    a harmonic oscillator   to  accuracy  0.0308.  *)

Corollary  yes_accurate_harmonic_oscillator:
    accurate_harmonic_oscillator  (iternF  h (p_init,q_init)  N)  N 0.0308.

Theorem yes_iternF_is_finite:  iternF_is_finite  .
  (* for  0 ≤ n ≤ N, the floating−point values don't overflow *)
```

Using these we prove that the low-level C function spec, integrate_spec_lowlevel , implies the high-level spec, integrate_spec :

```
Lemma subsume_integrate:
    funspec_sub (snd integrate_spec_lowlevel ) (snd integrate_spec).
```

The proof is only a few lines long (since all the hard work is done elsewhere). Then, using VST's subsumption principle [15] we can prove the body_integrate theorem stated in Section 2.

6 Soundness

Underlying our main result are several soundness theorems: soundness of VST [14] with respect to the formal operational semantics of C and the Flocq [6] model of IEEE-754 floating point; soundness of the Interval package and the VCFloat package with respect to models of floating point and the real numbers; proofs of Coquelicot's standard theorems of real analysis.

To test this, we used Coq's **Print Assumptions** command to list the axioms on which our proof depends. We use 6 standard axioms of classical logic (excluded middle, dependent functional extensionality, propositional extensionality) and 74 axioms about primitive floats and primitive 63-bit integers.

Regarding those 74: One could compute in Coq on the binary model of floating-point numbers, with no axioms at all. Our proofs are portable to such

[5] Counting nonblank, noncomment lines of C code or Coq proof. Proofs count text between (but not including) **Proof** and **Qed**.

reasoning, and we could do it with a small configuration change. But it would be slow. Standard installations of the Interval package now use the Coq kernel's support for native 64-bit floating-point and 63-bit modular integers. The algorithms within Interval and VCFloat (written as functional programs in Coq's Gallina language) run much faster with machine floats and machine integers. But then we have to trust that Coq's kernel uses them correctly, as claimed by those 74 axioms.

7 Related Work

A significant difference between the logical verification framework presented in this paper and the majority of existing methods for estimating the error in numerical solvers for differential equations is that our verification framework connects guaranteed error bounds to low-level properties of the solver implementation. An exception is the work by Boldo *et al.* [16], which verifies a C program implementing a second-order finite difference scheme for solving the one-dimensional acoustic wave equation. Although their model problem is a PDE, the framework could be generalized to IVPs for ODEs. The authors derive a total error theorem that composes global round-off and discretization error bounds, and connect this theorem to a proof of correctness of their C program. The authors use a combination of tools to perform their verification, including Coq, Gappa, Frama-C, Why, and various SMT solvers. Unlike VST, Frama-C has no soundness or correctness proof with respect to any formal semantics of C. Furthermore, VST is embedded in Coq and therefore enjoys the expressiveness of Coq's high-order logic; Frama-C lacks this expressivity, and this point was noted by the authors as a challenge in the verification effort.

The leapfrog method used as a solver for the two-dimensional model IVP in this paper is a simple example of one of many different families of solvers for IVPs for ODEs. Another class of methods that have been studied using logical frameworks and their related tools are Runge-Kutta methods. Boldo *et al.* [17] analyze the round-off errors (but not discretization error) of Runge-Kutta methods applied to linear one-dimensional IVPs using Gappa [18, 19], a tool for bounding round-off error in numerical programs that produces proof terms which can be verified in Coq. The authors use Gappa to derive tight local error bounds, similar to our use of VCFloat as described in Section 3.4, but perform their global round-off error analysis outside of a mechanized proof framework. Immler *et al.* formalize IVPs for ODEs in Isabelle/HOL, and prove the existence of unique solutions. They perform an error analysis on the simplest one-dimensional Runge-Kutta method and treat discretization error and round-off error uniformly as perturbations of the same order of magnitude.

Finally, as previously mentioned, validated numerical methods for ODEs have a long history of using interval arithmetic to derive guaranteed estimates for global truncation and round-off error [1, 3, 20–23]; this can be computationally inefficient for practical use. However, even unvalidated methods for estimating global error are inefficient. A common approach entails computing the solution a second time using a smaller time step, and using this second computation as an approximation of the exact solution [24]. An alternative approach implements *a posteriori* global error estimates in existing ODE solvers [25, 26] to dynamically adjust the time-step h, which controls discretization error. Unlike the error bounds derived in validated methods, the global error estimates have no guarantees of correctness.

8 Conclusion and future work

We have presented a framework for developing *end-to-end* proofs verifying the accuracy and correctness of imperative implementations of ODE solvers for IVPs, and have demonstrated the utility of this framework on leapfrog integration of the simple harmonic oscillator. Our framework leverages several libraries and tools embedded in the Coq proof assistant to modularize the verification process. The end-to-end result is a proof that the floating-point solution returned by a C implementation of leapfrog integration of the harmonic oscillator is an accurate solution to the IVP. This proof is composed of two main theorems that clearly disentagle program correctness from numerical accuracy.

Our main theorem regarding the numerical accuracy of the program treats round-off error, discretization error, and global error propagation distinctly, and makes clear how discretization error can be used to derive tight bounds on round-off error. By treating each source of error in this modular way, our framework could be extended to include additional sources of error of concern in the solution to IVPs for ODEs, such as error in the data and uncertainty in the model; we leave this extension to future work.

In its current state, our framework would require substantial user effort in order to be re-used on a different IVP or ODE solver. This obstacle could be overcome by developing proof automation for each component of the error analysis presented in Section 3. In particular, the derivation of local discretization error presented in Section 3.1 is standard: given user supplied input for the order at which to truncate the Taylor series expansion, the specification for the continuous system could be used to derive a maximum local error bound. Furthermore, the error propagation analysis in Section 3.2 could mostly be automated: if a user supplied (or if an unverified tool calculated) a transition matrix $M(h)$ for the ODE solver, the relevant matrices of eigenvalues Λ and eigenvectors V, and a guess for the bounds on the time-step h required for stability, one would only need to discharge a proof that $\overline{M(h)}^T M(h)\ V = V\Lambda$ in order to use the largest singular value of $M(h)$ for the total error theorem. Finally, while the proof of a local round-off error bound is already mostly automatic using VCFloat2, employing the global discretization error bound in the proof of local-round off error is currently done by the user; this process could be completed in an automatic way.

References

1. N.S. Nedialkov, K.R. Jackson, and G.F. Corliss. Validated solutions of initial value problems for ordinary differential equations. *Applied Mathematics and Computation*, 105(1):21–68, 1999.
2. Youdong Lin and Mark A. Stadtherr. Validated solutions of initial value problems for parametric odes. *Applied Numerical Mathematics*, 57(10):1145–1162, 2007.
3. Julien Alexandre dit Sandretto and Alexandre Chapoutot. Validated Explicit and Implicit Runge-Kutta Methods. *Reliable Computing electronic edition*, 22, July 2016.
4. Andrew W. Appel. Verified software toolchain. In Gilles Barthe, editor, *ESOP'11: European Symposium on Programming*, volume 6602 of *LNCS*, pages 1–17. Springer, 2011.
5. Sylvie Boldo, Catherine Lelay, and Guillaume Melquiond. Coquelicot: A user-friendly library of real analysis for coq. *Mathematics in Computer Science*, 9(1):41–62, March 2015.
6. Sylvie Boldo and Guillaume Melquiond. *Computer Arithmetic and Formal Proofs: Verifying Floating-point Algorithms with the Coq System.* Elsevier, 2017.
7. Andrew W. Appel and Ariel E. Kellison. Vcfloat2: Floating-point error analysis in coq. Draft, 2022.
8. Tahina Ramananandro, Paul Mountcastle, Benoît Meister, and Richard Lethin. A unified Coq framework for verifying C programs with floating-point computations. In *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs*, CPP 2016, page 15–26, New York, NY, USA, 2016. Association for Computing Machinery.
9. Arieh Iserles. *Error control.* Cambridge Texts in Applied Mathematics. Cambridge University Press, 2 edition, 2008.
10. Ernst Hairer, Christian Lubich, and Gerhard Wanner. *Geometric numerical integration*, volume 31 of *Springer Series in Computational Mathematics.* Springer-Verlag, Berlin, second edition, 2006. Structure-preserving algorithms for ordinary differential equations.
11. Nawaf Bou-Rabee and Jesús María Sanz-Serna. Geometric integrators and the hamiltonian monte carlo method. *Acta Numerica*, 27:113 – 206, 2018.
12. Sergio Blanes, Fernando Casas, and J. M. Sanz-Serna. Numerical integrators for the hybrid monte carlo method. *SIAM Journal on Scientific Computing*, 36(4):A1556–A1580, 2014.
13. Robert D. Skeel. *Integration Schemes for Molecular Dynamics and Related Applications.* Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.
14. Andrew W. Appel, Robert Dockins, Aquinas Hobor, Lennart Beringer, Josiah Dodds, Gordon Stewart, Sandrine Blazy, and Xavier Leroy. *Program Logics for Certified Compilers.* Cambridge, 2014.
15. Lennart Beringer and Andrew W. Appel. Abstraction and subsumption in modular verification of C programs. *Formal Methods in System Design*, 2021.
16. Sylvie Boldo, François Clément, Jean-Christophe Filliâtre, Micaela Mayero, Guillaume Melquiond, and Pierre Weis. Trusting computations: A mechanized proof from partial differential equations to actual program. *Computers and Mathematics with Applications*, 68(3):325–352, 2014.
17. Sylvie Boldo, Florian Faissole, and Alexandre Chapoutot. Round-off Error Analysis of Explicit One-Step Numerical Integration Methods. In *24th IEEE Symposium on Computer Arithmetic*, London, United Kingdom, Jul 2017.

18. Marc Daumas and Guillaume Melquiond. Certification of bounds on expressions involving rounded operators. *ACM Transactions on Mathematical Software*, 37(1):1–20, 2010.
19. Florent de Dinechin, Christoph Lauter, and Guillaume Melquiond. Certifying the floating-point implementation of an elementary function using gappa. *IEEE Transactions on Computers*, 60(2):242–253, 2011.
20. George F. Corliss. *Guaranteed Error Bounds for Ordinary Differential Equations*. Oxford University Press, 1994.
21. Nedialko S. Nedialkov, Kenneth R. Jackson, and John D. Pryce. An effective high-order interval method for validating existence and uniqueness of the solution of an IVP for an ODE. *Reliable Computing*, 7(6):449–465, 2001.
22. Kenneth R. Jackson and Nedialko S. Nedialkov. Some recent advances in validated methods for ivps for odes. *Applied Numerical Mathematics*, 42(1):269–284, 2002.
23. Robert Rihm. Interval methods for initial value problems in odes. In *Topics in validated computations : proceedings of IMACS-GAMM International Workshop on Validated Computation*, September 1993.
24. L. F. Shampine. Error estimation and control for odes. *J. Sci. Comput.*, 25(1):3–16, oct 2005.
25. Yang Cao and Linda Petzold. A posteriori error estimation and global error control for ordinary differential equations by the adjoint method. *SIAM Journal on Scientific Computing*, 26(2):359–374, 2004.
26. Benjamin Kehlet and Anders Logg. A posteriori error analysis of round-off errors in the numerical solution of ordinary differential equations. *Numer. Algorithms*, 76(1):191–210, sep 2017.