SAND2022-10930C

Towards Verified Rounding-Error Analysis for Stationary Iterative Methods

Ariel Kellison[§]
Department of Computer Science
Cornell University
Ithaca, NY, USA
ak2485@cornell.edu

Mohit Tekriwal[§]
Department of Aerospace engineering
University of Michigan
Ann Arbor, MI, USA
tmohit@umich.edu

Jean-Baptiste Jeannin
Department of Aerospace engineering
University of Michigan
Ann Arbor, MI, USA
jeannin@umich.edu

Geoffrey Hulette
Digital Foundations and Mathematics
Sandia National Laboratories
Livermore, CA, USA
ghulett@sandia.gov

***Abstract*—Iterative methods for solving linear systems serve as a basic building block for computational science. The computational cost of these methods can be significantly influenced by the round-off errors that accumulate as a result of their implementation in finite precision. In the extreme case, round-off errors that occur in practice can completely prevent an implementation from satisfying the accuracy and convergence behavior prescribed by its underlying algorithm. In the exascale era, where cost is paramount, a thorough and rigorous analysis of the delay of convergence due to round-off should not be ignored. In this paper, we use a small model problem and the Jacobi iterative method to demonstrate how the Coq proof assistant can be used to formally specify the floating-point behavior of iterative methods, and to rigorously prove the accuracy of these methods.**

***Index Terms*—Iterative convergence error, round-off error**

I. Introduction

Solving sparse linear systems is often the most time-consuming computation in large-scale scientific and engineering problems [1]. A major challenge in computational science is to therefore design methods for solving these systems that can be efficiently implemented at scale. This task is particularly challenging for widely used iterative methods, whose convergence behavior and attainable accuracy can be hard to determine a priori. Iterative methods [1] solve a system of linear equations by constructing a sequence of solution vectors, which approximate the exact solution of the linear system. A critical but often neglected consideration in the design of scalable iterative methods is a thorough analysis of the effect of rounding errors and the potential for their amplification [2]. Even when a thorough and interpretable rounding error analysis does exist, developing and executing comprehensive tests at scale to check that the analysis holds for a particular implementation is time consuming and computationally intensive [3], [4]. Furthermore, it is often hard to determine if inaccurate results are due to the floating-point behavior of the implementation or other sources of program error. The design of scalable and accurate iterative methods for solving linear systems is therefore inextricably linked to other notions of program correctness.

In this paper, we introduce our work towards verifying the accuracy and correctness of stationary iterative methods and their implementations using the Coq proof assistant [5]. The Coq proof assistant is an interactive theorem proving environment that has been used to great success in the development of *formal proofs* of the functional correctness of programs [6], [7]. The theoretical guarantee of a formal proof of program correctness is that the program will behave as expected on all possible inputs. For numerical programs such as stationary iterative methods, a thorough proof of functional correctness requires performing round-off error analysis in Coq; we refer to this error analysis as *verified round-off error analysis*.

Our verified round-off error analysis is informed by the standard round-off error analysis of iterative methods given by Higham and co-authors [8], [9], but provides concrete error bounds in place of big-O estimates, and uses a slightly different rounding error model that accounts for subnormal numbers. Our work is facilitated by advancements in automatic and interactive theorem proving [10]–[13] and other recent formalizations of numerical methods [14]–[25]. Our verification approach leverages several pre-existing Coq packages and libraries for reasoning about mathematical abstractions in linear algebra and real-analysis, and for reasoning about floating-point arithmetic. Overall the work outlined in this

Ariel Kellison is supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Department of Energy Computational Science Graduate Fellowship under Award Number DE-SC0021110.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

[§]Equal contribution

paper makes the following contributions, which we believe are relevant to both the interactive theorem proving community and to the developers and maintainers of numerical software:

- We illustrate how two previously unconnected Coq libraries – VCFloat [6], [26] and MathComp [27] – can be interfaced in order to perform verified round-off error analysis of algorithms from numerical linear algebra;
- We demonstrate how to develop formal models of stationary iterative methods in both exact arithmetic and floating-point arithmetic in Coq;
- We show how formal models of numerical algorithms can be used to prove concrete bounds on the total round-off error for the Jacobi method [1], using a simple model problem consisting of a 3×3 linear system;
- We extend the Coq linear algebra library MathComp [27] with vector and matrix infinity norm definitions that are sufficient for round-off error analysis.

This paper is structured as follows. Our model problem is introduced in Section II. In Section III, we provide an overview of the Coq MathComp mathematical component library and VCFloat package that were used in our formalization. The functional models for the Jacobi iterations in floating-point and exact arithmetic are described in Section IV. Our main theorem on the accuracy of floating-point Jacobi iterations carried out in single-precision arithmetic is given in Section V, for the Jacobi iteration algorithm applied to a simple model problem. Finally, our formalization required the addition of matrix and vector infinity norms to the core of the MathComp library, and we discuss the definitions and properties of these norms in Section VI. In Section VII, we discuss some key takeaways from our work and end with future directions.

Our full formalization is available at: https://github.com/ak-2485/iterative_error.git.

II. Problem Formulation

Stationary iterative methods are among the oldest and simplest methods for solving linear systems of the form

$$Ax = b, \quad A = M + N \in \mathbb{R}^{n \times n}, \quad b \in \mathbb{R}^n. \tag{1}$$

The nonsingular and usually non-Hermitian matrix A and vector b in such systems typically appear, for example, in the solution of a partial differential equation. The stationary iterations for the solution vector x have the form

$$Mx_m + Nx_{m-1} = b, \tag{2}$$

where the vector x_{m-1} is an approximation to the solution vector x obtained after $m-1$ iterations, and is known at the m^{th} step. The unknown x_m is therefore given by

$$x_m = -(M^{-1}N)x_{m-1} + M^{-1}b \tag{3}$$

In this paper, we demonstrate our work towards verifying the accuracy and correctness of stationary iterative methods by considering the Jacobi method, where $M = diag(A)$, on a simple model problem. In this case the model problem is representative of solving a linear boundary value problem with a second order central difference scheme; this simple model problem serves as a sufficient "stress test" for our proposed verification method, indicating how the Coq libraries and packages we utilize will need to be developed in order to handle larger problems and provide more general proofs (see Section VII). In particular, we consider the tri-diagonal matrix system $Ax = b$ where A is a coefficient matrix of size 3×3, x is the unknown solution vectors, and b is a known data vector:

$$A = \frac{1}{h^2}\begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}; \quad b = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \tag{4}$$

Although most of our theorems are parameterized by the discretization parameter h, we set $h = 1$ globally in our analysis for simplicity. Ultimately, we are interested in a formal proof of the accuracy of an iterative solution to the system (4) obtained in floating-point arithmetic by a particular implementation in an imperative language. Fortunately, there is a well-established road map for obtaining such a proof. In particular, the following steps for proving the accuracy and correctness of floating-point programs has been described before by Appel and Bertot [28] for a Newton's-method square root function, and Kellison and Appel [29] for Verlet integration of the simple harmonic oscillator. For our model problem, the steps are as follows.

1) Write a C program that solves the system (4) by Jacobi iterations of the form (3).
2) Write a *floating-point functional model* in Coq – a recursive functional program that operates on floating-point values – that solves the system (4) by Jacobi iterations of the form (3) in the precision of the C program from Step 1.
3) Prove that the program written in Step 1 implements the floating-point functional model of Step 2 using a program logic for C.
4) Write a *real functional model* in Coq – a recursive functional program that operates on Coq's axiomatic real numbers – that solves the system (4) by Jacobi iterations of the form (3) using exact arithmetic.
5) Prove a tight upper bound on the accuracy by which the floating-point functional model approximates the real functional model.

In this work, we focus on the proof of accuracy of Jacobi iterations, which involves steps 2, 4, and 5. In the following section, we briefly describe the tools we have used for writing the functional models in steps 2 and 4. We describe the proof of accuracy in Section V.

III. Background

We define functional models as purely functional programs written in Coq that implement the Jacobi iterates in equation (3). The real functional model is written using the mathematical components (MathComp) library, and the floating-point functional model is written using the VCFloat library. For the interactive theorem proving community, a highlight of this work is a demonstration of the interaction between

the VCFloat and MathComp libraries for proving an upper bound on the accuracy to which the floating-point specification approximates the exact arithmetic specification. We briefly review relevant background on the MathComp and VCFloat libraries in the following sections.

A. The VCFloat Coq Library

The VCFloat [26], [30] Coq library performs semi-automatic floating-point round-off error analysis on floating-point expressions. VCFloat utilizes the Flocq [31] formalization of IEEE-754 binary floating-point formats, which is an inductive data-type parameterized by the precision $prec \in \mathbb{N}$ and the exponent $emax \in \mathbb{Z}$. For the round-to-nearest rounding mode, VCFloat models rounding error as

$$rnd(x) = x \times (1 + \delta) + \epsilon \tag{5}$$

where $\delta \leq prec$ gives the maximum relative error for normal numbers and $\epsilon \leq (3 - emax - prec - 1)$ gives the maximum absolute error for subnormal numbers.

VCFloat provides wrappers for the ordinary Flocq floating-point arithmetic operators that enable users to write floating-point expressions in Coq's "native" logic – which we refer to as shallow-embedded expressions – using infix notation, along with tactics for automatically translating these shallow-embedded expressions into deep-embedded expressions, which are expression trees over floating-point types. VCFloat's core theorem operates on these deep-embedded expressions by soundly applying the rounding error model of equation (5) to generate a correctly rounded deep-embedded expressions over the reals; that is, an expression tree with the correct insertion of epsilons (ϵ) and deltas (δ). These rounded expression trees are only generated if intermediate proofs of the absence of overflow and underflow are discharged, which guarantees their soundness with respect to the Flocq formalization and the rounding error model [1].

Finally, additional VCFloat tactics automatically transform the correctly rounded expression tree back into a shallow-embedded correctly rounded real expression. An absolute forward error bound is then obtained automatically by applying the Coq interval library [32] to the absolute difference between the correctly rounded shallow-embedded expression and its corresponding shallow-embedded expression in the absence of rounding error. In particular, if we represent the correctly rounded shallow-embedded expression as $\tilde{r}$ and its corresponding shallow-embedded expression in the absence of rounding error as r, then the Coq Interval library automatically proves theorems with goals of the form $|\tilde{r} - r| \leq const.$ supposing that the shared variables of r and $\tilde{r}$ are sufficiently bounded in the hypothesis of the theorem.

The VCFloat predicate used for stating Coq theorems bounding the absolute local round-off error of a reified expression tree $expr$ over floating-point types by the real value bnd is (`prove_round-off_bound` $map1$ $map2$ $expr$ bnd), where $map2$ maps the positive identifiers used to construct the reified expression tree to floating-point valued variables, and $map1$ maps these floating-point valued variables to their real-valued bounds; the real-valued bounds on variables are provided by the user, and are necessary both for proving the absence of underflow and overflow of the expression, and for generating tight error bounds. A full demonstration of VCFloat's functionality is provided by Appel and Kellison [30].

B. The Mathcomp Coq Library

The MathComp mathematical components [27] library formalizes theories of sequences, matrices, and vectors, and provides an abstraction over algebraic structures like rings and fields. These algebraic structures can be instantiated with Coq's axiomatic reals, which allows users to perform real analysis using Coq's standard library. The MathComp theories for matrices and sequences were utilized for this work.

In Mathcomp, a matrix is defined as a function from an ordinal type to an appropriate ring type.

```
Variant matrix : predArgType :=
   Matrix of {ffun 'I_m × 'I_n →R}.
```

where `'I_n` and `'I_m` denote an ordinal type, i.e., set of naturals from $0 \ldots n-1$ and $0 \ldots m-1$ respectively. For instance, a 2×2 real valued matrix, $A = [1, 2; 3, 4]$ can be defined [2] as

```
Definition A := \matrix_(i< 2, j < 2)
   (if (i = = 0%N :> nat) then
     (if (j = = 0%N :> nat) then 1%Re else 2%Re) else
     (if (j = = 0%N :> nat) then 3%Re else 4%Re)).
```

The matrix theory in Coq defines generic properties like transpose, conjugates, matrix space theory, eigenspace theory etc. We leverage this formalization for our work, while filling some existing gaps in the theory relating to matrix and vector norms.

The `seq` library in MathComp [27] allows us to define a finite sequence. In our formalization, we use sequences to reason about matrix and vector infinity norms. Therefore, it is worth going through some relevant operations from the sequence library. The following notation defines a map for each element `x` in the sequence `s`.

```
[seq E | x ← s] := map (fun x ⇒ E) s.
```

To extract an n^{th} element in the sequence, we use the following notation from MathComp

```
nth x0 s i
```

The MathComp [27] library in Coq provides an infrastructure to define iterated operations. The notation

```
Notation "\big [ op / idx ]_ ( i ←
r \ P ) F" :=
  (bigop idx r (fun i ⇒ BigBody i op P%B F)) :
  big_scope.
```

[1] In fact, there are additional properties that must hold, see Appel and Kellison [30] and Ramananandro and co-authors [26]

[2] The form **Definition** $name$ $(arguments)$: $type$:= term in Coq binds name to the value of the term of type *type*.

allows us to define iterated sums and products by instantiating the `op` operator and the appropriate identity `idx`. Here, `F` is a function of `i` chosen from a finite sequence `r` when the predicate `P` holds true. The matrix operations like matrix-vector multiplication, dot products, traces etc. are defined in terms of these big operations.

We will next discuss our formalization of the error analysis in Coq.

IV. Functional Models

We first define *functional models* for the iterative algorithm (3). This is implemented using the `Fixpoint` operator in Coq which defines a recursive function. In this case, the iterative solution after m steps, x_m is defined recursively.

The real-valued functional model is defined in Coq as

```
Fixpoint X_m_real (m n:nat) (x0 b: 'cV[R]_n) (h:R) :
'cV[R]_n:=
  match m with
  | O ⇒ x0
  | p.+1 ⇒ S_mat n h *m (X_m_real p x0 b h) +
          inv_A1 n h *m b
  end.
```

`X_m_real` takes as inputs: `m`, the iteration number; `n`, the matrix and vector dimension; `x0`, the real valued initial guess column vector of size n; `b`, the real valued data column vector of size n, and the parameter h, which represents the discretization step. `X_m_real` returns a real valued column vector of size n represented by the type `'cV[R]_n`. If the iteration step is zero, `X_m_real` returns the initial guess vector. If the iteration step is non-zero, `X_m_real` returns the iterative solution corresponding to the formula (3). Here, `S_mat` is the iteration matrix, i.e., $S_{mat} \triangleq -M^{-1}N$, and `inv_A1` is the inverse of M.

The floating point functional model is defined in Coq as

```
Fixpoint X_m (m : nat) x0 b h : list (ftype Tsingle) :=
  match m with
  | O ⇒ x0
  | p.+1 ⇒ vec_add (S_mat_mul (X_m p x0 b h))
          (A1_inv_mul_b b h)
  end
.
```

where `S_mat_mul` is defined as $fl((-M^{-1}N)x_{m-1})$ and `A1_inv_mul_b` is defined as $fl(M^{-1}b)$. `vec_add` is a routine that adds elements in a vector recursively.

Here, we used the `CompCert` [33] lists to define a vector of floats. In this work, we consider floats of single precision. Therefore, the return type of `x_m` is `list(ftype Tsingle)`. This choice is governed by the ease with which we can switch between the lists and MathComp vectors. We define this relationship between `CompCert` lists and MathComp column vectors using the following definition

```
Definition listR_to_vecR {n:nat} (l : list R) :=
\col_(i < n)
  match (nat_of_ord i) with
    | S n ⇒ List.nth (S n) l 0
    | 0 ⇒ List.nth 0 l 0
  end.
```

The definition `listR_to_vecR` takes a list of reals of length n, and returns a column vector of length n. The column vector is constructed using pattern matching on the row index i which varies from $0 \ldots (n-1)$. If i is zero, we extract an element from the list at index 0. If i is non-zero, or a successor of some natural number, we extract an element from the list at that index. An important point to note here is that the natural numbers in Coq are either 0 or successor of some other natural number.

V. A Formal Accuracy Proof

The global iterative error defined after $k+1$ iterations is defined as

$$e_{k+1} = ||\tilde{x}_{k+1} - x|| \tag{6}$$

where x is the solution obtained by solving the linear system $Ax = b$ exactly, i.e. $x = A^{-1}b$, and $\tilde{x}_{k+1}$ is the iterative solution after $k+1$ steps computed in floating-point arithmetic. We can further split the global iterative error into the *global round-off error* and the *exact iterative error*:

$$\begin{aligned} e_{k+1} &= ||\tilde{x}_{k+1} - x|| \\ &\leq \underbrace{||\tilde{x}_{k+1} - x_{k+1}||}_{\text{global round-off error}} + \underbrace{||x_{k+1} - x||}_{\text{exact iterative error}} . \end{aligned} \tag{7}$$

The exact iterative error is the difference between the solution obtained by solving the linear system exactly and the solution obtained by solving the linear system using an iterative method in exact arithmetic. A formal proof of convergence in the presence of iterative error in exact arithmetic is given by Tekriwal and co-authors [34]. In this work, we focus particularly on the global round-off error, which is the difference between the iterative solutions obtained in exact and floating-point arithmetic. In particular, we represent the floating-point solution to iterative system in equation (2) as

$$\tilde{x}_{k+1} = -M^{-1}N\tilde{x}_k + M^{-1}b + f_{k+1} \tag{8}$$

where f_{k+1} is the local absolute round-off error from computing $(-M^{-1}N\tilde{x}_k + M^{-1}b)$ at step $k+1$. If we denote the error between the iterative solution obtained in ideal arithmetic from the iterative solution obtained in floating point arithmetic as e_k, then the following recurrence relation holds.

$$e_{k+1} = ||x_{k+1} - \tilde{x}_{k+1}||_\infty \leq ||(M^{-1}N)e_k||_\infty + ||f_{k+1}||_\infty.$$

From examining the error terms at successive steps it is clear that the norm-wise error terms satisfy

$$e_{k+1} \leq \max_{n \leq k}(f_n) \sum_{i=0}^{k} ||M^{-1}N||_\infty^i. \tag{9}$$

where $\max(f_n)$ is the maximum local error over all k iterations.

In order to obtain a concrete maximum floating-point error vector $\max_{n \leq k}(f_n)$, we must first make an initial guess for a component-wise bound on the absolute value of the floating-point solution vector $\tilde{x}$ at any iteration k. This user provided bound must not induce overflow or underflow of the solution

as discussed in Section III-A. Furthermore, we must prove in our global accuracy theorem that the floating-point error accumulated over k iterations does not cause the components of the computed solution to exceed this bound.

We represent the injection of the i-th component of the floating-point solution vector at iteration k into the reals as $\hat{x}_k^i$ and provide a loose bound of $|\hat{x}_k^i| \leq 100$ for our model problem. This bound is encoded into a data-structure, which we denote as *bmap*, which maps the positive identifiers used to construct the reified expression tree for each component of the solution vector to floating-point valued variables. If (*varmp* s) is the map data structure that maps the floating-point valued variables in the tuple s to their real-valued bounds then the predicate (prove_round-off_bound *bmap* (*varmp* s) *expr bnd*) is used to state that the absolute forward error on the component *expr* of the floating-point solution is less than *bnd*. A concrete numerical value for *bnd* is derived automatically using the Coq interval package [35]while constructing the proof. If $\tilde{x}^1$, $\tilde{x}^2$, and $\tilde{x}^3$ are deep-embedded expression trees constructed from the shallow-embedded expression for a single iteration of the floating-point function (i.e., for $k = 1$ in (X_m k x_0 $\tilde{b}$ h)), then the Coq theorems for the component-wise local floating point error of the solution vector $\tilde{x}$ are then stated as follows.

```
Theorem prove_round-off_bound_x1_aux:
  forall s: state,
  prove_round-off_bound bmap (varmap s) x̃¹ (9.04e-06).

Theorem prove_round-off_bound_x2_aux:
  forall s: state,
  prove_round-off_bound bmap (varmap s) x̃² (1.5e-05).

Theorem prove_round-off_bound_x3_aux:
  forall s: state,
  prove_round-off_bound bmap (varmap s) x̃³ (9.01e-06).
```

Using these theorems, we can then construct the vector $\max_{n \leq k}(f_n)$ of component-wise round-off errors as

$$||\max_{n \leq k}(f_n)||_\infty = ||f_{max}||_\infty = (1.5e-05).$$

A core component of the definition of the predicate (prove_round-off_bound *map*1 *map*2 *expr bnd*) is the predicate (boundsmap_denote *map*1 (*map*2 *args*)). If (boundsmap_denote *map*1 (*map*2 *args*) = true), then the floating-point valued variables in *args* are bounded by the user supplied bounds used to construct *map*1.

We make use of the (boundsmap_denote *map*1 (*map*2 *args*)) predicate in the following theorem, which proves an error bound on the infinity norm of the shallow-embedded expressions for the functional models by invoking each of the prior lemmas for the component-wise error on the deep-embedded expressions. For the sake of clarity for the reader, in the following theorems we omit the Coq functions that inject single-precision floating-point data structures into their real counterparts, as well as those functions that map Coq lists to mathcomp vectors. We instead represent the result of such an injection on the floating-point data $\tilde{y}$ as $\hat{y}$. Recall that the discretization parameter is assigned globally to $h = 1$.

```
Theorem step_round_off_error:
  ∀ s : state,
  boundsmap_denote bmap (varmap s) →
  let k:= 1 in
  || X_m_real(k,ŝ,b̂,h) − X_m (k,s,b̃,h) ||∞ ≤ ||f_k||∞.
```

Our main accuracy theorem is stated as

```
Theorem iterative_round_off_error:
  ∀ (x̃₀ : list 𝔽), (k : ℕ),
  (boundsmap_denote bmap (varmap x̃) ∧
  ||x̂₀||∞ ≤ 48 ∧ ||b̂||∞ ≤ 1 ∧ k ≤ 100) →
      let x̃_k = X_m (k, x̃₀, b̃, h) in
      let x_k = X_m_real (k, x̂₀, b̂, h) in
      ||x_k − x̂_k||∞ ≤ ||f_max||∞ Σ_{m=0}^{k} ||M⁻¹N||∞^m
      ∧ boundsmap_denote bmap (varmap x̃_k).
```

We are considering the infinity norm of a vector and a matrix which is defined mathematically as

$$||v||_\infty = \max_i |v_i|; \quad ||A||_\infty = \max_i \sum_{j=1}^{n} |A_{ij}| \tag{10}$$

We define the vector and matrix infinity norms in Coq as

```
Definition vec_inf_norm {n:nat} (v : 'cV[R]_n) :=
 bigmaxr 0%Re [seq (Rabs (v i 0)) | i ← enum 'I_n].

Definition matrix_inf_norm {n:nat} (A: 'M[R]_n) :=
 bigmaxr 0%Re [seq (row_sum A i) | i ← enum 'I_n].
```

where bigmaxr is defined in the MathComp library in Coq, and provides an infrastructure to define the maximum of elements in a given sequence. The definition vec_inf_norm takes a real column vector of size n denoted by 'cV[R]_n, and returns a maximum of the sequence of absolute values of each of its components, denoted by Rabs (v i 0), where i is taken list of ordinal numbers {0 .. (n-1)}. Similarly, the definition matrix_inf_norm takes a real values square matrix A denoted by 'M[R]_n and returns a maximum of the sequence of the row sum of the components of A. We define the row sum in Coq as follows

```
Definition row_sum {n:nat} (A: 'M[R]_n) (i: 'I_n) :=
  \big[+%R/0]_(j<n) Rabs (A i j).
```

row_sum takes a square matrix A and a row index i and returns a sum of the absolute values of the components of A in that row. The big operator, defined in MathComp, returns an iterated sum of finite components in the row i of the matrix A in this case.

The proof of the theorem iterative_round_off_error follows by induction. The base case follows trivially: no error is introduced between the input starting vectors $\tilde{x}_0$ and its injection $\hat{x}_0$ to the reals. For the induction case, we first prove the left conjunct $||x_{k+1} - \tilde{x}_{k+1}||_\infty \leq ||f_{k+1}||_\infty \sum_{m=0}^{k+1} ||M^{-1}N||_\infty^m$. Decomposing $||x_{k+1} - \hat{x}_{k+1}||_\infty$ as single iterations over the inputs x_k and $\tilde{x}_k$ yields

$$||x_{k+1} - \hat{x}_{k+1}||_\infty =$$
$$||\text{X\_m\_real } (1, x_k, \hat{b}, h) \; - \; \text{X\_m } (1, \hat{x}_k, \tilde{b}, h)||_\infty,$$

which can further be decomposed into a local error term and an accumulation of error term:

$$||\texttt{X\_m\_real}\ (1, x_k, \hat{b}, h)\ -\ \texttt{X\_m}\ (1, \tilde{x}_k, \tilde{b}, h)||_\infty \leq$$
$$||\underbrace{\texttt{X\_m\_real}\ (1, x_k, \hat{b}, h)\ -\ \texttt{X\_m\_real}\ (1, \hat{x}_k, \hat{b}, h)}_{\text{global accumulation of error}}||_\infty +$$
$$||\underbrace{\texttt{X\_m\_real}\ (1, \hat{x}_k, \hat{b}, h)\ -\ \texttt{X\_m}\ (1, \tilde{x}_k, \tilde{b}, h)}_{\text{local round-off error}}||_\infty =$$
$$\underbrace{||M^{-1}N||_\infty||x_k - \hat{x}_k||_\infty}_{\text{global accumulation of error}} +$$
$$\underbrace{||\texttt{X\_m\_real}\ (1, \hat{x}_k, \hat{b}, h)\ -\ \texttt{X\_m}\ (1, \tilde{x}_k, \tilde{b}, h)||_\infty}_{\text{local round-off error}}.$$

The desired conclusion

$$||x_{k+1} - \hat{x}_{k+1}||_\infty \leq ||f_k||_\infty \sum_{m=0}^{k+1} ||M^{-1}N||_\infty^m$$

then follows in two steps. To bound the global accumulation of error term we need only invoke the inductive hypothesis which bounds $||x_k - \hat{x}_k||_\infty$. To bound the local round-off error term, we must have evidence that each component of the floating-point solution vector $\tilde{x}_k$ has not exceeded the user specified bound that was used to derive the local-floating point error bound $||f_{max}||_\infty$ that serves as the maximum local error; observe that this follows from the inductive hypothesis which includes the predicate (`boundsmap_denote bmap (varmap` $\tilde{x_k}$`)`). This predicate is used to satisfy the premise of the theorem `step_round_off_error`, which is invoked to bound the local round-off error term and concludes the proof of the left conjunct of the conclusion.

Finally, in order to prove the right conjunct of the conclusion, `boundsmap_denote bmap (varmap` $\tilde{x}_{k+1}$`)`, we must show that each component i of the floating-point solution vector at step $k+1$ is bounded by the user supplied bounds: $|\hat{x}_k^i| \leq 100$. To do this, we decompose the error bound at step $k+1$ that we have just proved in order to bound the floating-point solution:

$$||\hat{x}_{k+1}||_\infty \leq ||f_k||_\infty \sum_{m=0}^{k+1} ||M^{-1}N||_\infty^m + ||x_{k+1}||_\infty. \quad (11)$$

We obtain a bound on the exact arithmetic solution vector $||x_{k+1}||_\infty$ that satisfies $||\hat{x}_{k+1}||_\infty \leq 100$ under the conditions $||x_0||_\infty \leq 48$, $||b||_\infty \leq 1$, and $k \leq 100$:

```
Lemma sol_up_bound_exists:
  ∀ (x_0 b : lists ℝ) (k: ℕ),
      (||x_0||_∞ ≤ 48 ∧ ||b||_∞ ≤ 1 ∧ k ≤ 100) →
      || X_m_real(k + 1, x_0, b, h)||_∞ ≤ 99.
```

Invoking this lemma concludes the proof.

Note that from the definition of iterative system (3), we arrive at the following bound for the real solution vector x_{k+1}

$$||x_{k+1}||_\infty \leq ||(M^{-1}N)||_\infty^{k+1}||x_o||_\infty +$$
$$||M^{-1}||_\infty||b||_\infty \sum_{j=0}^{m} ||M^{-1}N||_\infty^j$$

For our model problem, we proved that the norm of the iteration matrix is exactly 1, i.e., $||M^{-1}N||_\infty = 1$. Therefore, the geometric sum of the norm of the iteration matrix depends on the iteration count, i.e., $\sum_{j=0}^{m} ||M^{-1}N||_\infty^j = k+1$. We also proved that $||M^{-1}||_\infty \leq \frac{1}{2}$. Hence,

$$||x_{k+1}||_\infty \leq ||x_o||_\infty + \frac{1}{2}||b||_\infty(k+1)$$

Thus, to prove that $||x_{k+1} \leq 99$, we need to invoke the preconditions, $||x_o||_\infty \leq 48$, $k \leq 100$, and $||b||_\infty \leq 1$.

VI. Matrix and vector infinity norm formalization

A by-product of this work is the formalization of infinity norms of matrix and vectors. This is missing in the current formalization of linear algebra in MathComp. We therefore contribute a formalization of the properties of infinity norms.

Table I illustrates properties of vector infinity norm, and Table II illustrates the properties of matrix norm, that we formalized in Coq.

TABLE I
Formalization of properties of vector infinity norm

Properties	Coq formalization
$\|\|0\|\|_\infty = 0$	`Lemma vec_inf_norm_0_is_0 {n:nat}: @vec_inf_norm n.+1 0 = 0%Re.`
$\|\|a+b\|\|_\infty \leq \|\|a\|\|_\infty + \|\|b\|\|_\infty$	`Lemma triang_ineq {n:nat} : forall a b: 'cV[R]_n.+1, vec_inf_norm(a + b) <= vec_inf_norm a + vec_inf_norm b.`
$0 <= \|\|v\|\|_\infty$	`Lemma vec_norm_pd {n:nat} (v : 'cV[R]_n.+1): 0 <= vec_inf_norm v.`
$\|\| - v\|\|_\infty = \|\|v\|\|_\infty$	`Lemma vec_inf_norm_opp {n:nat}: forall v: 'cV[R]_n, vec_inf_norm v = vec_inf_norm (-v).`

VII. Conclusion and future work

We argue that tools that connect guarantees of program correctness to guarantees of floating-point accuracy can assist in the design of scalable, accurate, and correct iterative methods for solving linear systems by providing a priori guarantees on worst case convergence behavior and attainable accuracy. In this work, we demonstrated how the Coq proof assistant and its associated packages and libraries can be used to provide guarantees of the floating-point accuracy of a small model

TABLE II
FORMALIZATION OF PROPERTIES OF MATRIX INFINITY NORM

Properties	Coq formalization
$\|\|Av\|\|_\infty \leq \|\|A\|\|_i \|\|v\|\|_\infty$	`Lemma submult_prop {n:nat} (A: 'M[R]_n.+1) (v : 'cV[R]_n.+1): vec_inf_norm (A *m v) <= matrix_inf_norm A * vec_inf_norm v.`
$0 <= \|\|A\|\|_i$	`Lemma matrix_norm_pd {n:nat} (A : 'M[R]_n.+1): 0 <= matrix_inf_norm A.`
$\|\|AB\|\|_i \leq \|\|A\|\|_i \|\|B\|\|_i$	`Lemma matrix_norm_le {n:nat}: forall (A B : 'M[R]_n.+1), matrix_inf_norm (A *m B) <= matrix_inf_norm A * matrix_inf_norm B.`
$\|\|A+B\|\|_i \leq \|\|A\|\|_i + \|\|B\|\|_i$	`Lemma matrix_norm_add {n:nat}: forall (A B : 'M[R]_n.+1), matrix_inf_norm (A + B) <= matrix_inf_norm A + matrix_inf_norm B.`
$\|\|1\|\|_i = 1$	`Lemma matrix_inf_norm_1 {n:nat}: @matrix_inf_norm n.+1 1 = 1%Re.`

$||A||_i$ denotes induced infinity matrix norm.

problem whose solution was found using Jacobi iterates. As future work, we have three goals. First, we plan to generalize this analysis to a generic $n \times n$ matrix and a generic iteration algorithm, i.e., parametric in A, M and N. Second, we plan to connect our accuracy proof to previous work [34] that has formalized sufficient and necessary conditions for asymptotic convergence of the iterative solution obtained in exact arithmetic to the solution obtained by solving $Ax = b$ directly. Combining these works would provide a proof of accuracy that soundly composes the effects of rounding errors to the effects of iterative errors. Finally, we plan to connect our accuracy proof to a proof of program correctness in order to guarantee that a binary compiled from a C implementation of an iterative method will always exhibit numerical error within the proven bounds. We intend to carry out the proof of program correctness using the Verified Software Toolchain (VST) [36], which is proven sound with respect to the formal operational semantics of C.

REFERENCES

[1] Y. Saad, *Iterative methods for sparse linear systems*, 2nd ed. Philadelphia, Pa.: Society for Industrial and Applied Mathematics (SIAM), 2003.

[2] E. Carson and Z. Strakoš, "On the cost of iterative computations," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 378, p. 20190050, 03 2020.

[3] J. Dongarra, J. Hittinger, J. Bell, L. Chacon, R. Falgout, M. Heroux, P. Hovland, E. Ng, C. Webster, and S. Wild, "Applied mathematics research for exascale computing," 2 2014. [Online]. Available: https://www.osti.gov/biblio/1149042

[4] G. Gopalakrishnan, P. D. Hovland, C. Iancu, S. Krishnamoorthy, I. Laguna, R. A. Lethin, K. Sen, S. F. Siegel, and A. Solar-Lezama, "Report of the hpc correctness summit, january 25-26, 2017, washington, dc," 10 2017. [Online]. Available: https://www.osti.gov/biblio/1470989

[5] "Introduction and contents — coq 8.15.2 documentation," https://coq.inria.fr/distrib/current/refman/, (Accessed on 08/04/2022).

[6] A. W. Appel, "Coq's vibrant ecosystem for verification engineering," in *CPP'22: Proceedings of the 11th ACM SIGPLAN International Conference on Certified Programs and Proofs*, 2022, pp. 2–11.

[7] T. Ringer, K. Palmskog, I. Sergey, M. Gligoric, and Z. Tatlock, "QED at large: A survey of engineering of formally verified software," *CoRR*, vol. abs/2003.06458, 2020. [Online]. Available: https://arxiv.org/abs/2003.06458

[8] N. J. Higham and P. A. Knight, "Componentwise error analysis for stationary iterative methods," in *Linear Algebra, Markov Chains, and Queueing Models*, C. D. Meyer and R. J. Plemmons, Eds. New York, NY: Springer New York, 1993, pp. 29–46.

[9] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed. USA: Society for Industrial and Applied Mathematics, 2002.

[10] R. O'Connor, "Certified exact transcendental real number computation in coq," in *International Conference on Theorem Proving in Higher Order Logics*. Springer, 2008, pp. 246–261.

[11] S. Boldo, C. Lelay, and G. Melquiond, "Coquelicot: A user-friendly library of real analysis for Coq," *Mathematics in Computer Science*, vol. 9, no. 1, pp. 41–62, 2015.

[12] F. Garillot, G. Gonthier, A. Mahboubi, and L. Rideau, "Packaging mathematical structures," in *International Conference on Theorem Proving in Higher Order Logics*. Springer, 2009, pp. 327–342.

[13] É. Martin-Dorel, L. Rideau, L. Théry, M. Mayero, and I. Pasca, "Certified, efficient and sharp univariate taylor models in coq," in *2013 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*. IEEE, 2013, pp. 193–200.

[14] I. Pasca, "Formal verification for numerical methods," Ph.D. dissertation, Université Nice Sophia Antipolis, 2010.

[15] G. Cano and M. Dénès, "Matrices à blocs et en forme canonique," in *JFLA - Journées francophones des langages applicatifs*, D. Pous and C. Tasson, Eds. Aussois, France: Damien Pous and Christine Tasson, Feb. 2013. [Online]. Available: https://hal.inria.fr/hal-00779376

[16] R. Thiemann, "A perron–frobenius theorem for deciding matrix growth," *Journal of Logical and Algebraic Methods in Programming*, p. 100699, 2021.

[17] S. Boldo, F. Clément, J.-C. Filliâtre, M. Mayero, G. Melquiond, and P. Weis, "Formal proof of a wave equation resolution scheme: the method error," in *International Conference on Interactive Theorem Proving*. Springer, 2010, pp. 147–162.

[18] ——, "Trusting computations: a mechanized proof from partial differential equations to actual program," *Computers & Mathematics with Applications*, vol. 68, no. 3, pp. 325–352, 2014.

[19] ——, "Wave equation numerical resolution: a comprehensive mechanized proof of a c program," *Journal of Automated Reasoning*, vol. 50, no. 4, pp. 423–456, 2013.

[20] M. Tekriwal, K. Duraisamy, and J.-B. Jeannin, "A formal proof of the lax equivalence theorem for finite difference schemes," in *NASA Formal Methods*, A. Dutle, M. M. Moscato, L. Titolo, C. A. Muñoz, and I. Perez, Eds. Cham: Springer International Publishing, 2021, pp. 322–339.

[21] F. Immler and J. Hölzl, "Numerical analysis of ordinary differential equations in isabelle/hol," in *International Conference on Interactive Theorem Proving*. Springer, 2012, pp. 377–392.

[22] F. Immler, "A verified ode solver and smale's 14th problem," Dissertation, Technische Universität München, München, 2018.

[23] F. Immler and C. Traut, "The flow of odes," in *International Conference on Interactive Theorem Proving*. Springer, 2016, pp. 184–199.

[24] ——, "The flow of odes: Formalization of variational equation and poincaré map," *Journal of Automated Reasoning*, vol. 62, no. 2, pp. 215–236, 2019.

[25] F. Immler, "Formally verified computation of enclosures of solutions of ordinary differential equations," in *NASA Formal Methods*, J. M. Badger and K. Y. Rozier, Eds. Cham: Springer International Publishing, 2014, pp. 113–127.

[26] T. Ramananandro, P. Mountcastle, B. Meister, and R. 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*, ser. CPP 2016. New York, NY,

USA: Association for Computing Machinery, 2016, p. 15–26. [Online]. Available: https://doi.org/10.1145/2854065.2854066
[27] A. Mahboubi and E. Tassi, "Mathematical components," 2017.
[28] A. W. Appel and Y. Bertot, "C-language floating-point proofs layered with VST and Flocq," *Journal of Formalized Reasoning*, vol. 13, no. 1, pp. 1–16, Dec. 2020. [Online]. Available: https://hal.inria.fr/hal-03130704
[29] A. E. Kellison and A. W. Appel, "Verified numerical methods for ordinary differential equations," in *15th International Workshop on Numerical Software Verification*, 2022.
[30] A. W. Appel and A. E. Kellison, "VCFloat2: Floating-point Error Analysis in Coq," https://github.com/VeriNum/vcfloat/raw/master/doc/vcfloat2.pdf, April 2022.
[31] S. Boldo and G. Melquiond, "Flocq: A unified library for proving floating-point algorithms in coq," in *2011 IEEE 20th Symposium on Computer Arithmetic*. IEEE, 2011, pp. 243–252.
[32] G. Melquiond, "Floating-point arithmetic in the coq system," *Information and Computation*, vol. 216, pp. 14–23, 2012.
[33] X. Leroy, S. Blazy, D. Kästner, B. Schommer, M. Pister, and C. Ferdinand, "Compcert-a formally verified optimizing compiler," in *ERTS 2016: Embedded Real Time Software and Systems, 8th European Congress*, 2016.
[34] M. Tekriwal, J. Miller, and J.-B. Jeannin, "Formal verification of iterative convergence of numerical algorithms," 2022. [Online]. Available: https://arxiv.org/abs/2202.05587
[35] G. Melquiond, Érik Martin-Dorel, M. Mayero, I. Pasca, L. Rideau, and L. Théry, "Interval Coq Library," http://coq-interval.gforge.inria.fr/, (Accessed on 9/20/2020).
[36] A. W. Appel, "Verified software toolchain," in *Programming Languages and Systems*, G. Barthe, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 1–17.