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, iterative methods.**

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 iterative methods, whose convergence behavior and attainable accuracy can be hard to determine a priori. Iterative methods [2] solve a system of linear equations by constructing a sequence of solution vectors that approximate the exact solution to 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 [3]. Even when a thorough 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 [1], [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 given by a formal proof of program correctness is that the program will behave as expected on all possible inputs. This is a much stronger guarantee than what is provided by traditional software testing. For numerical programs such as stationary iterative methods, a thorough proof of functional correctness requires performing round-off error analysis – that is, analyzing the difference between the floating-point solution obtained by the program and the solution obtained by the ideal algorithm whose behavior is specified using exact arithmetic. We refer to formal proofs of round-off error obtained in an interactive theorem proving environment as *verified round-off error analysis*.

Our verified round-off error analysis for iterative methods is informed by the standard round-off error analysis given by Higham and co-authors [8], [9], but provides concrete error bounds in place of big-O estimates, and uses a slightly

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.

Mohit Tekriwal and Jean-Baptiste Jeannin are partially supported by the National Science Foundation under grant NSF FMitF 2219997.

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

different rounding error model that accounts for subnormal floating-point 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 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 [26], [27] and Mathcomp [28] – can be interfaced in order to perform verified round-off error analysis of algorithms from numerical linear algebra;
- We demonstrate how to develop formal functional models of stationary iterative methods in both exact arithmetic and floating-point arithmetic in Coq;
- We show how functional models of numerical algorithms can be used to prove concrete bounds on the total round-off error for the Jacobi method [2] using a simple model problem consisting of a 3×3 linear system;
- We extend the Coq mathematical components library (Mathcomp) [28] 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 Mathcomp and VCFloat Coq libraries 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 on a simple model problem is given in Section V. We discuss some key takeaways from our work and end with future directions in Section VI. The definitions and properties of the matrix and vector infinity norms that were developed for this work are discussed in Appendix A.

Our full formalization is available at https://github.com/VeriNum/iterative_methods.

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 non-singular and usually non-Hermitian matrix A and vector b in such systems typically appear, for example, in the solution of a partial differential equation. M is chosen such that it is easily invertible. Rather than solving the system $Ax = b$ exactly, one can approximate the solution vector x using stationary iterations of 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 the potential challenges of using existing Coq libraries and packages on larger problems. 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:

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

The matrices M and N of the Jacobi method for this problem are

$$M = \frac{1}{h^2}\begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}; \quad N = \frac{1}{h^2}\begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & -1 \\ 0 & -1 & 0 \end{bmatrix}.$$

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 [29] for a Newton's-method square root function, and Kellison and Appel [30] 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.

6) Prove a bound on the iterative convergence error – the difference between the solution obtained by solving the linear system directly and the solution obtained by solving the linear system using an iterative method.
7) Prove a *total error bound* by composing the proofs of iterative convergence error and floating-point round-off error.

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 Mathcomp [28] and the floating-point functional model is written using VCFloat [26], [27]. For the interactive theorem proving community, a highlight of this work is a demonstration of the interaction between the VCFloat and Mathcomp libraries.

We chose Coq for our development because we intend to compose the effect of rounding error with iterative convergence error formalized in Coq as described by Tekriwal and co-authors [31]. Other interactive theorem provers like HOL Light [32], HOL4 [33], and PVS [34] can be used to formalize properties about floating-point rounding errors and matrices. There have been works on formalization of floating-point error analysis [35]–[38] and matrix theory [39], [40] in HOL. The IEEE-754 floating-point standard has also been formalized in PVS [41] and has been used in various applications [42]–[44].

We briefly review relevant background on Mathcomp library and VCFloat package in the following sections.

A. The VCFloat Coq Package

VCFloat performs automated floating-point round-off error analysis on floating-point expressions in Coq. VCFloat utilizes the Flocq [45] 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 a *functional modeling language* over the Flocq formalization of IEEE-754 binary floating-point formats that enables users to write floating-point expressions – which we refer to as shallow-embedded expressions – using infix notation, along with tactics (algorithms) for automatically translating these shallow-embedded expressions into deep-embedded expressions, which are expression trees over floating-point types. VCFloat's core theorem effectively operates on a deep-embedded expression e by applying the rounding error model of equation (5) to generate a shallow-embedded expressions $\tilde{r}$ over the reals containing epsilons (ϵ) and deltas (δ) such that $rnd(e) = \tilde{r}$. The *soundness* of VCFloat's core theorem follows from the fact that a shallow-embedded expression $\tilde{r}$ is only generated if certain validity conditions are met (e.g., that operations in e do not overflow in the working precision – see [26, §4, Theorem 3]).

Additional VCFloat tactics used in conjunction with the Coq interval library [46] assist users in automatically deriving verified absolute forward error bounds; that is, on the absolute difference between the correctly rounded shallow-embedded expression $\tilde{r}$ and its corresponding shallow-embedded expression r in the absence of rounding error. In particular, VCFloat automatically generates a constant $const$ such that $|\tilde{r} - r| \leq const$ is a provable theorem in Coq.

The VCFloat predicate used in the statement of Coq theorems bounding the absolute local round-off error of a deep-embedded expression tree $expr$ over floating-point types by the real value bnd is (`prove_round−off_bound` $map1$ $map2$ $expr$ bnd), where $map2$ maps identifiers for variables in the deep-embedded expression tree to their corresponding floating-point valued variables in the shallow-embedded floating-point expression, and $map1$ maps these floating-point valued variables to their real-valued bounds; real-valued bounds on variables are provided by the user, and are necessary both for proving the absence of overflow of the expression and for generating tight error bounds. A full demonstration of VCFloat's functionality is provided by Appel and Kellison [27].

B. The Mathcomp Coq Library

The Mathcomp mathematical components [28] library formalizes theories of sequences, matrices, and vectors, and provides an abstraction over algebraic structures like rings and fields. Properties like transpose, conjugate, matrix space theory, eigenspace theory are also provide. 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. Our formalization of existing gaps in the theory relating to matrix and vector norms is described in Appedix A.

Matrices in Mathcomp are formally represented by a row-major list of their coefficients. This implementation is hidden by wrappers so that matrices and vectors may be treated as abstract. In particular,

- `'M[R](m, n)` is the type of $m \times n$ matrices with coefficients in $\mathbb{R}$.
- `'M[R]_n` is the type of $n \times n$ square matrices.
- `'rV[R]_n` is the type of $1 \times n$ row vectors.
- `'cV[R]_n` is the type of $n \times 1$ column vectors.
- `\matrix_(i < m, j < n)` $Expr$`(i, j)` is the $m \times n$ matrix with coefficients defined in $Expr$`(i, j)`

As an example, consider the definition [1] of Matrix_A which defines the 2×2 real valued matrix $A = [1, 2; 3, 4]$:

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

The notation (val %Re) is used to denote that val is a real number and the notation (val %N) is used to denote that val is a natural number. Note that the [R] in the above listed abstractions is typically omitted so that types are displayed as, e.g., 'M(m, n).

IV. Functional Models

Our functional models for Jacobi iterations are recursive functional programs in Coq that model the iterative algorithm (3). These are implemented using Coq's Fixpoint operator, which defines a recursive function.

We define the real-valued functional model in Coq as X_m_real:

```
Fixpoint X_m_real (m n: nat) (x0 b: 'cV_n) (h: R) :
    'cV_n :=
      match m with
      | O ⇒ x0
      | p+1 ⇒ (S_mat n h) × (X_m_real p x0 b h) +
              (inv_A1 n h) × b
      end.
```

The function 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 discretization parameter h (which is globally set to $h = 1$ for our model problem). The function X_m_real returns a real valued column vector of size n represented by the type `cV[R]_n. The match statement is equivalent to an if then else statement: 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 the matrix M.

We define the floating-point functional model in Coq as X_m:

```
Fixpoint X_m (m : nat) (x0 b: 'cV_n) (h: R) :
    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 denotes multiplication in single-precision between the matrix $S_{mat} \triangleq -M^{-1}N$ of floating-point values and the vector x_{m-1} of floating-point values, and A1_inv_mul_b denotes multiplication in single-precision between the matrix M^{-1} of floating-point values and the vector b of floating-point values. The function vec_add adds elements in a list recursively. We used CompCert [47] lists to represent matrices and vectors of floating-point values; the return type of the floating-point function model X_m is therefore a list of single-precision values: list(ftype Tsingle). This choice is governed by the ease with which we can switch between CompCert lists and Mathcomp vectors. Defining a mapping between CompCert lists over the real numbers and Mathcomp column vectors is straightforward. We map CompCert lists of floating-point values to Mathcomp column vectors using VCFloat functions that inject the floating-point numbers into the reals. We note that VCFloat is able to handle functional models with multiple precisions; our choice to use only single-precision operations and values for our model problem was arbitrary.

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*:

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

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 [31]. In this work, we derive a bound 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 relation holds.

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

Our formalization in Coq of the infinity norm $|| \cdot ||_\infty$ is described in Appendix A.

If $\max(f_n)$ is the maximum local error over all k iterations, then the norm-wise error terms satisfy

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

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

In order to obtain a concrete maximum absolute floating-point error vector $\max_{n \le k}(f_n)$ using VCFloat, 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 (see Section III-A). For our model problem, we chose a loose bound of $|\hat{x}_k^i| \le 100$, where $\hat{x}_k^i$ denotes the injection of the i-th component of the floating-point solution vector at iteration k into the reals. In general, this initial guess should be determined as follows. Consider that equation (9) can be rewritten as

$$|\hat{x}_{k+1}| \le \max_{n \le k}(f_n) \sum_{i=0}^{k} ||M^{-1}N||_{\infty}^{i} + |x_{k+1}|. \tag{10}$$

Estimates on the second term on the right hand side of equation (10) should follow from qualitative information about the system, and estimates on the first term on the right hand side should follow from standard results on the floating-point error for matrix-vector multiplication, as described by Higham and Knight [8, §2]. We will show 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 our estimated bound.

The initial guess for a component-wise bound on the absolute value of the floating-point solution vector is encoded into a data-structure, which we denote as $bmap$, which maps the identifiers used to construct the deep-embedded expression tree for 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 as briefly described in Section III-A while constructing the proof in Coq. If $\tilde{x}^1$, $\tilde{x}^2$, and $\tilde{x}^3$ are deep-embedded expression trees generated by VCFloat from the shallow-embedded expression for a single iteration of the floating-point functional model (i.e., for $k = 1$ in (`X_m` k x_0 $\tilde{b}$ h)), then the Coq theorems for the absolute 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).
```

The theorem `prove_round_off_bound_x1_aux` gives rounding error in first component of the solution vector $\tilde{x}$; `prove_round_off_bound_x2_aux` and `prove_round_off_bound_x3_aux` give rounding error in the second and third components of the solution vector $\tilde{x}$, respectively. The maximum local rounding error $\max_{n \le k}(f_n)$ is the maximum of the component-wise round-off errors. In this particular case, using these theorems, we construct the vector $\max_{n \le k}(f_n)$ of component-wise round-off errors as

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

A core component of the definition of the predicate (`prove_round_off_bound` $map1$ $map2$ $expr$ bnd) is the predicate (`boundsmap_denote` $map1$ ($map2$ $args$)). If (`boundsmap_denote` $map1$ ($map2$ $args$) = `true`), then the floating-point valued variables in $args$ are bounded by the user supplied bounds used to construct $map1$.

We state the following theorems using some Coq syntax, but 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 $h = 1$.

The theorem step_round_off_error bounds the error on the infinity norm of the shallow-embedded expressions for the functional models; the proof of the theorem follows by invoking each of the prior lemmas (e.g., prove_round_off_bound_x2_aux) for the component-wise error on the deep-embedded expressions:

```
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_max||∞.
```

Our main accuracy theorem bounds the floating-point error over $k \le 100$ iterations:

```
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).
```

A proof of the theorem `iterative_round_off_error` follows by induction. The base case follows trivially: no error is introduced between the input starting vector $\tilde{x}_0$ and its injection $\hat{x}_0$ to the reals. For the inductive step, we first prove the left conjunct,

$$||x_{k+1} - \tilde{x}_{k+1}||_{\infty} \le ||f_{k+1}||_{\infty} \sum_{m=0}^{k+1} ||M^{-1}N||_{\infty}^{m}.$$

Decomposing $||x_{k+1} - \hat{x}_{k+1}||_{\infty}$ as single iterations over the inputs x_k and $\tilde{x}_k$ yields

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

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

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

The desired conclusion

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

then follows in two steps. To bound the global accumulation of error term we only need to 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 bounds encoded in *bmap*; 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, the right conjunct of the conclusion `boundsmap_denote bmap` (`varmap` $\tilde{x}_{k+1}$), follows by proving that each component i of the floating-point solution vector at step $k+1$ is bounded by the user supplied bounds: $|\hat{x}_k^i| \le 100$. To do this, we decompose the error bound at step $k+1$:

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

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

```
Lemma sol_up_bound_exists:
  ∀ (x_0 b : lists ℝ) (k: ℕ),
      (||x0||∞ ≤ 48 ∧ ||b||∞ ≤ 1 ∧ k ≤ 100) →
      || X_m_real(k + 1, x0, 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}

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

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 $||M^{-1}||_\infty \le \frac{1}{2}$. Hence,

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

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

VI. 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 guarantee the floating-point accuracy of a small model problem whose solution was found using Jacobi iterates. As future work, we have two 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. This requires formalizing standard results on the floating-point error for matrix-vector multiplication, as described by Higham and Knight [8, §2].

Second, we plan to connect our accuracy proof to proofs of program correctness and iterative convergence error, as described in steps 1-7 of the verification outline given in Section II. Previous work [31] 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 with the effects of iterative errors into a proof of a *total error* bound. We plan to connect our total error bound 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 error within the proven bounds. We intend to carry out the proof of program correctness using the Verified Software Toolchain (VST) [48], which is proven sound with respect to the formal operational semantics of CompCert C [47].

Appendix

A. *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 desribe here our formalization of the properties of infinity norms.

The `seq` library in Mathcomp allows us to define finite sequences. In our formalization, we use sequences to reason about matrix and vector infinity norms. We therefore introduce here some relevant operations from the sequence library. The following notation `[seq E | x ← s] := map (fun x ⇒ E) s` defines a map for each element `x` in the sequence `s`. To extract an n^{th} element in the sequence, we use the notation `nth x0 s i`.

Mathcomp allows us to define iterated sums and products by instantiating the `op` operator and the appropriate identity `idx`:

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

Here, `F` is a function of `i` chosen from a finite sequence `r` when the predicate `P` holds true.

We define the vector infinity norm $||v||_\infty = \max_i |v_i|$ and the matrix infinity norm

$$||A||_\infty = \max_i \sum_{j=1}^{n} |A_{ij}|. \tag{12}$$

in Coq as

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

and

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

The Mathcomp abstraction `bigmaxr` is used here to define the maximum of elements in a 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 as `row_sum`,

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

which takes a square matrix A and an index i and returns a sum of the absolute values of the components of A in row i. In this case, the `big` operator returns an iterated sum of finite components in the row i.

Table I and Table II illustrate the properties of the vector infinity norm and matrix infinity 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_n+1, vec_inf_norm(a + b) ≤ vec_inf_norm a + vec_inf_norm b.`
$0 \leq \|\|v\|\|_\infty$	`Lemma vec_norm_pd {n:nat} (v : 'cV_n+1): 0 ≤ vec_inf_norm v.`
$\|\|-v\|\|_\infty = \|\|v\|\|_\infty$	`Lemma vec_inf_norm_opp {n:nat}: forall v: 'cV_n, vec_inf_norm v = vec_inf_norm (-v).`

TABLE II
FORMALIZATION OF PROPERTIES OF MATRIX INFINITY NORM

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

REFERENCES

[1] 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

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

[3] 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.

[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. 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.
[28] A. Mahboubi and E. Tassi, "Mathematical components," 2017.
[29] 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
[30] A. E. Kellison and A. W. Appel, "Verified numerical methods for ordinary differential equations," in *15th International Workshop on Numerical Software Verification*, 2022.
[31] 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
[32] J. Harrison, "Hol light: A tutorial introduction," in *International Conference on Formal Methods in Computer-Aided Design*. Springer, 1996, pp. 265–269.
[33] K. Slind and M. Norrish, "A brief overview of hol4," in *International Conference on Theorem Proving in Higher Order Logics*. Springer, 2008, pp. 28–32.
[34] S. Owre, J. M. Rushby, and N. Shankar, "PVS: A prototype verification system," in *International Conference on Automated Deduction*. Springer, 1992, pp. 748–752.
[35] J. Harrison, "Floating-point verification using theorem proving," in *International School on Formal Methods for the Design of Computer, Communication and Software Systems*. Springer, 2006, pp. 211–242.
[36] H. Becker, N. Zyuzin, R. Monat, E. Darulova, M. O. Myreen, and A. Fox, "A verified certificate checker for finite-precision error bounds in coq and hol4," in *2018 Formal Methods in Computer Aided Design (FMCAD)*, 2018, pp. 1–10.
[37] H. Becker, M. Tekriwal, E. Darulova, A. Volkova, and J.-B. Jeannin, "Dandelion: Certified Approximations of Elementary Functions," in *13th International Conference on Interactive Theorem Proving (ITP 2022)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), J. Andronick and L. de Moura, Eds., vol. 237. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, pp. 6:1–6:19. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2022/16715
[38] J. Harrison, "Floating point verification in HOL Light: The exponential function," *Formal Methods in System Design*, vol. 16, no. 3, pp. 271–305, 2000.
[39] Z. Shi, Y. Zhang, Z. Liu, X. Kang, Y. Guan, J. Zhang, and X. Song, "Formalization of matrix theory in hol4," *Advances in Mechanical Engineering*, vol. 6, p. 195276, 2014.
[40] J. Aransay and J. Divasón, "Formalisation of the computation of the echelon form of a matrix in Isabelle/HOL," *Formal Aspects of Computing*, vol. 28, no. 6, pp. 1005–1026, 2016.
[41] P. S. Miner, "Defining the IEEE-854 floating-point standard in PVS," NASA Langley Research Center, Tech. Rep., 1995.
[42] M. M. Moscato, M. A. Feliú, C. A. Muñoz *et al.*, "Provably correct floating-point implementation of a point-in-polygon algorithm," in *International Symposium on Formal Methods*. Springer, 2019, pp. 21–37.
[43] M. M. Moscato, C. A. Muñoz, A. Dutle, F. Bobot *et al.*, "A formally verified floating-point implementation of the compact position reporting algorithm," in *International Symposium on Formal Methods*. Springer, 2018, pp. 364–381.
[44] H. Herencia-Zapana, R. Jobredeaux, S. Owre, P.-L. Garoche, E. Feron, G. Perez, and P. Ascariz, "Pvs linear algebra libraries for verification of control software algorithms in C/ACSL," in *NASA Formal Methods Symposium*. Springer, 2012, pp. 147–161.
[45] 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.
[46] G. Melquiond, "Floating-point arithmetic in the coq system," *Information and Computation*, vol. 216, pp. 14–23, 2012.
[47] 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.
[48] A. W. Appel, "Verified software toolchain," in *Programming Languages and Systems*, G. Barthe, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 1–17.