LAProof: A Library of Formal Proofs of Accuracy and Correctness for Linear Algebra Programs

Ariel Kellison[1,2], Andrew W. Appel[3], Mohit Tekriwal[4], and David Bindel[1]

1, Dept. of Computer Science, Cornell University
2, Sandia National Laboratories, Livermore, CA
3, Dept. of Computer Science, Princeton University
4, Dept. of Aerospace Engineering, University of Michigan

What is LAProof?

LAProof is a library of machine-checked accuracy proofs for basic linear algebra operations.

VeriNum / **LAProof**

What is LAProof?

LAProof is a library of machine-checked accuracy proofs for basic linear algebra operations.

The accuracy proofs from LAProof

- assume only a low-level formal model of IEEE-754 arithmetic,

VeriNum / **LAProof**

What is LAProof?

LAProof is a library of machine-checked accuracy proofs for basic linear algebra operations.

The accuracy proofs from LAProof

- assume only a low-level formal model of IEEE-754 arithmetic,
- are mixed (backward-forward) rounding error bounds that account for underflow,

VeriNum / LAProof

What is LAProof?

LAProof is a library of machine-checked accuracy proofs for basic linear algebra operations.

The accuracy proofs from LAProof

- assume only a low-level formal model of IEEE-754 arithmetic,
- are mixed (backward-forward) rounding error bounds that account for underflow,
- capture low order error terms exactly – not approximating as $O(u^2)$,

VeriNum / LAProof

What is LAProof?

LAProof is a library of machine-checked accuracy proofs for basic linear algebra operations.

The accuracy proofs from LAProof

- assume only a low-level formal model of IEEE-754 arithmetic,
- are mixed (backward-forward) rounding error bounds that account for underflow,
- capture low order error terms exactly – not approximating as $O(u^2)$,
- can be used to formally verify the accuracy of programs implementing operations defined by the basic linear algebra subprograms (BLAS) specification, and

VeriNum / LAProof

What is LAProof?

LAProof is a library of machine-checked accuracy proofs for basic linear algebra operations.

The accuracy proofs from LAProof

- assume only a low-level formal model of IEEE-754 arithmetic,
- are mixed (backward-forward) rounding error bounds that account for underflow,
- capture low order error terms exactly – not approximating as $O(u^2)$,
- can be used to formally verify the accuracy of programs implementing operations defined by the basic linear algebra subprograms (BLAS) specification, and
- are developed entirely within the Coq proof assistant.

VeriNum / LAProof

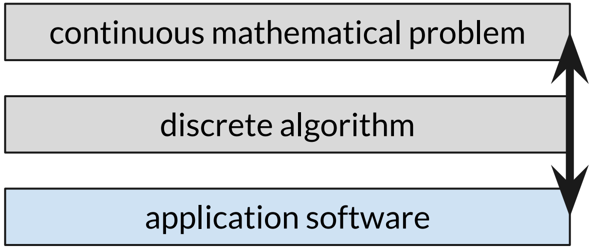
Why verify the accuracy of programs implementing BLAS?

BLAS implementations are vital in *numerical analysis*

"...our mission is to compute quantities that are typically uncomputable, from an analytic point of view, and to do it with lightning speed." - Trefethen 1992

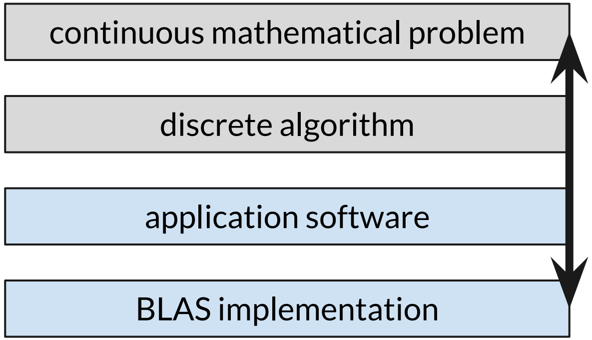
BLAS implementations are vital in *numerical analysis*

"...our mission is to compute quantities that are typically uncomputable, from an analytic point of view, and to do it with lightning speed." - Trefethen 1992

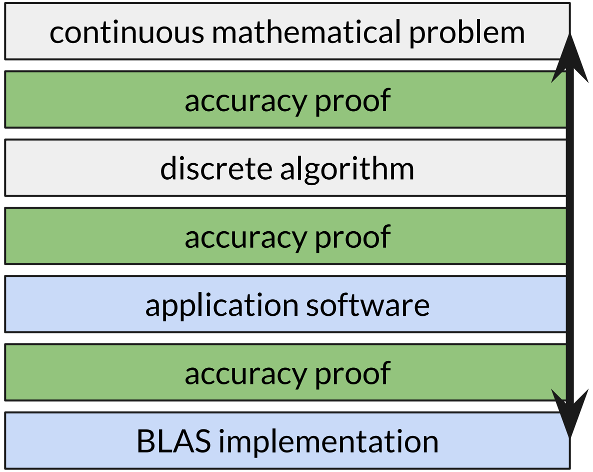


BLAS implementations are vital in *numerical analysis*

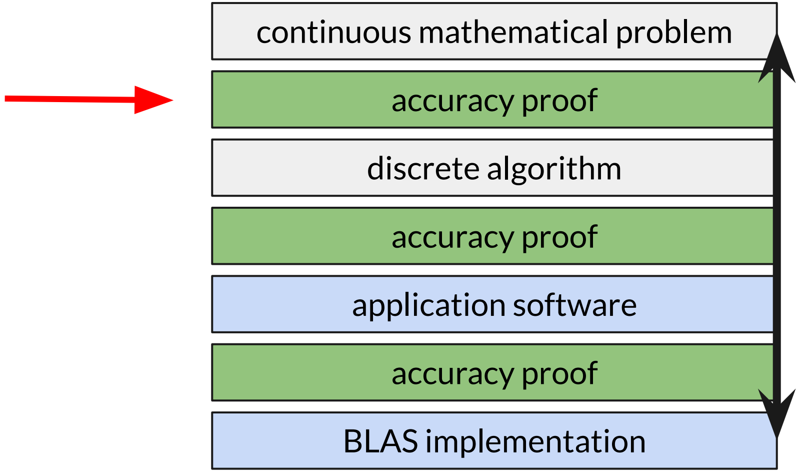
"...our mission is to compute quantities that are typically uncomputable, from an analytic point of view, and to do it with lightning speed." - Trefethen 1992



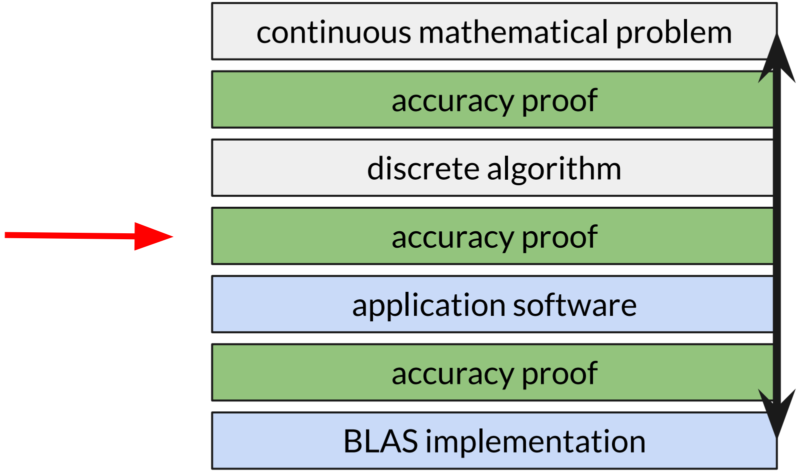
Accuracy proofs give us confidence in these algorithms and their implementations



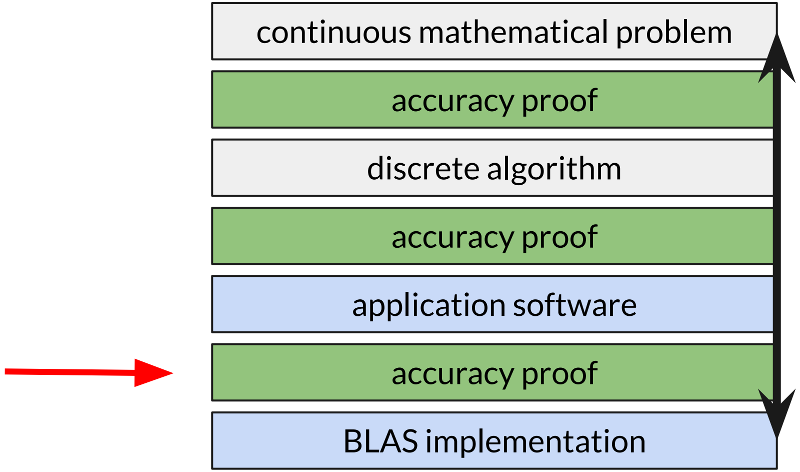
Accuracy proofs give us confidence in these algorithms and their implementations



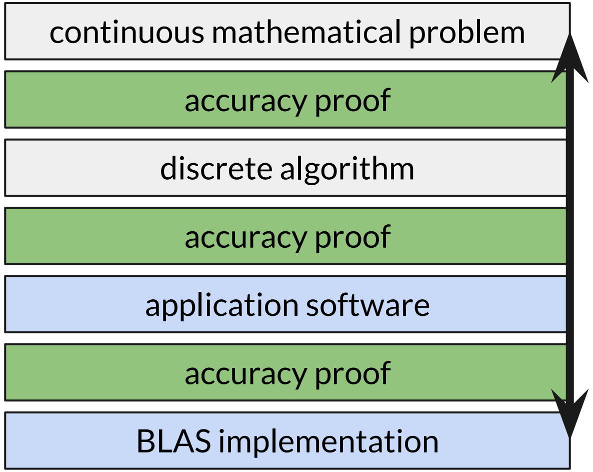
Accuracy proofs give us confidence in these algorithms and their implementations



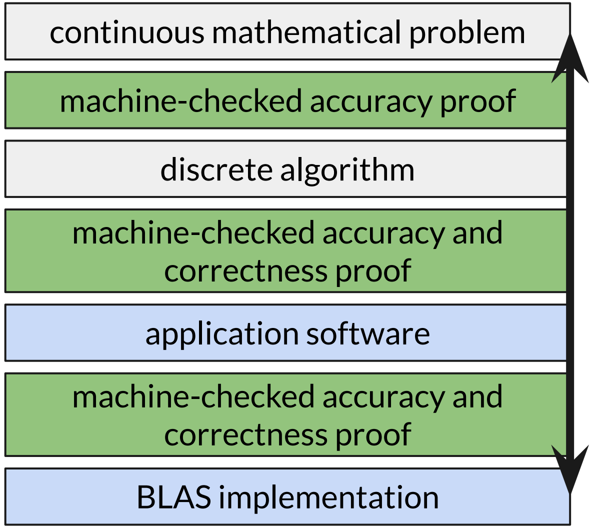
Accuracy proofs give us confidence in these algorithms and their implementations



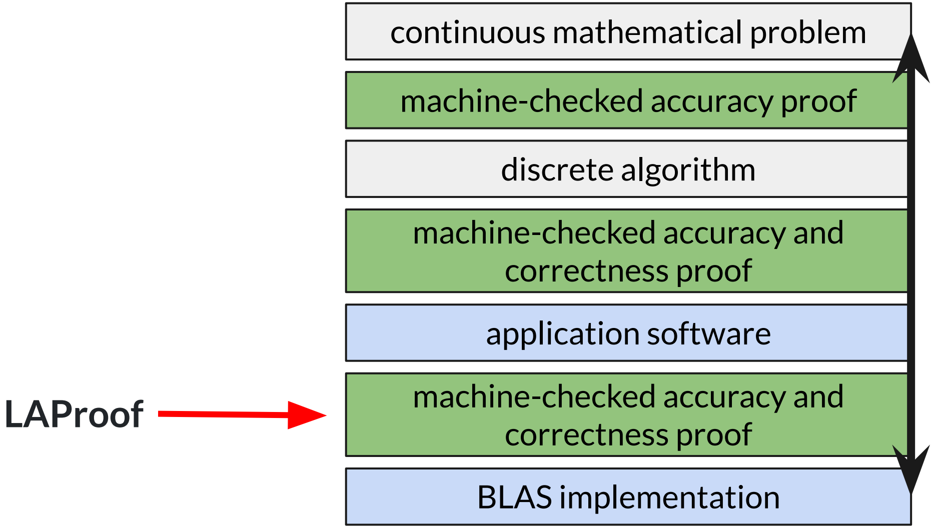
In high-consequence settings, machine-checked proofs are the gold standard



In high-consequence settings, machine-checked proofs are the gold standard



In high-consequence settings, machine-checked proofs are the gold standard

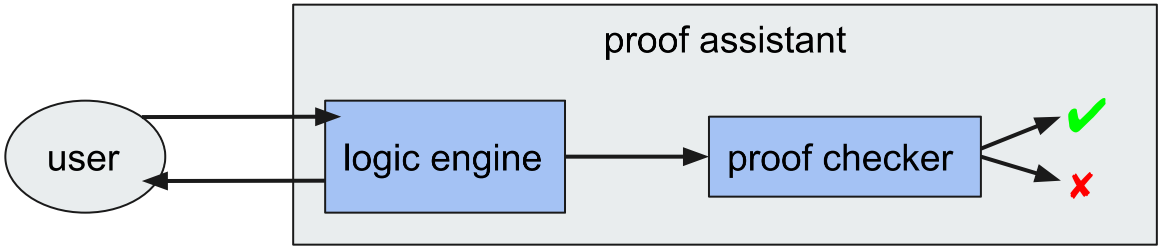


What is a machine-checked proof?

- A formal derivation in a formal logical system, checked by a proof-checking program

What is a machine-checked proof?

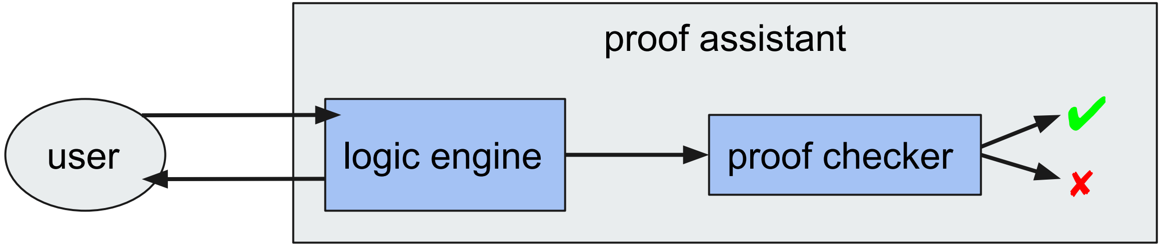
- **A formal derivation in a formal logical system, checked by a proof-checking program**
- A common tool for developing machine checked proofs is a *proof assistant*
- Proof assistants provide
  - a DSL (domain-specific language) for building proofs
  - a program that verifies whether proofs are valid derivations in a formal logic
  - libraries of definitions, theorems, and programs for proof automation



* adapted from Ringer et al, 2019.

What is a machine-checked proof?

- **A formal derivation in a formal logical system, checked by a proof-checking program**
- A common tool for developing machine checked proofs is a *proof assistant*
- Proof assistants provide
  - a DSL (domain-specific language) for building proofs
  - a program that verifies whether proofs are valid derivations in a formal logic
  - libraries of definitions, theorems, and programs for proof automation



* adapted from Ringer et al, 2019.

- The **Coq proof assistant** – free, open-source software; development largely supported by INRIA (The National Institute for Research in Digital Science and Technology) since 1989.

Which linear algebra operations does LAProof provide machine-checked accuracy proofs for?

What operations does LAProof provide?

TABLE I
LAPROOF VECTOR OPERATIONS

DOT	$r \leftarrow x \cdot y$
sVec	$r \leftarrow \alpha x$
SUM	$r \leftarrow \sum_i x_i$
VecAdd	$r \leftarrow x + y$
VecAXPBY	$r \leftarrow \alpha x + \beta y$
VecNRM1	$r \leftarrow \lVert x \rVert_1$
VecNRM2	$r \leftarrow \lVert x \rVert_2$

TABLE II
LAPROOF MATRIX-VECTOR OPERATIONS

MV	$r \leftarrow Ax$
sMV	$r \leftarrow \alpha Ax$
GEMV	$r \leftarrow \alpha Ax + \beta y$

TABLE III
LAPROOF MATRIX OPERATIONS

sMat	$R \leftarrow \alpha A$
MM	$R \leftarrow AB$
MatAdd	$R \leftarrow A + B$
sMM	$R \leftarrow \alpha AB$
MatAXPBY	$R \leftarrow \alpha X + \beta Y$
GEMM	$R \leftarrow \alpha AX + \beta Y$

What operations does LAProof provide?

TABLE I
LAPROOF VECTOR OPERATIONS

DOT	$r \leftarrow x \cdot y$
sVec	$r \leftarrow \alpha x$
SUM	$r \leftarrow \sum_i x_i$
VecAdd	$r \leftarrow x + y$
VecAXPBY	$r \leftarrow \alpha x + \beta y$
VecNRM1	$r \leftarrow \|x\|_1$
VecNRM2	$r \leftarrow \|x\|_2$

TABLE II
LAPROOF MATRIX-VECTOR OPERATIONS

MV	$r \leftarrow Ax$
sMV	$r \leftarrow \alpha Ax$
GEMV	$r \leftarrow \alpha Ax + \beta y$

TABLE III
LAPROOF MATRIX OPERATIONS

sMat	$R \leftarrow \alpha A$
MM	$R \leftarrow AB$
MatAdd	$R \leftarrow A + B$
sMM	$R \leftarrow \alpha AB$
MatAXPBY	$R \leftarrow \alpha X + \beta Y$
GEMM	$R \leftarrow \alpha AX + \beta Y$

What operations does LAProof provide?

TABLE I
LAPROOF VECTOR OPERATIONS

DOT	$r \leftarrow x \cdot y$
sVec	$r \leftarrow \alpha x$
SUM	$r \leftarrow \sum_i x_i$
VecAdd	$r \leftarrow x + y$
VecAXPBY	$r \leftarrow \alpha x + \beta y$
VecNRM1	$r \leftarrow \|x\|_1$
VecNRM2	$r \leftarrow \|x\|_2$

TABLE II
LAPROOF MATRIX-VECTOR OPERATIONS

MV	$r \leftarrow Ax$
sMV	$r \leftarrow \alpha Ax$
GEMV	$r \leftarrow \alpha Ax + \beta y$

TABLE III
LAPROOF MATRIX OPERATIONS

sMat	$R \leftarrow \alpha A$
MM	$R \leftarrow AB$
MatAdd	$R \leftarrow A + B$
sMM	$R \leftarrow \alpha AB$
MatAXPBY	$R \leftarrow \alpha X + \beta Y$
GEMM	$R \leftarrow \alpha AX + \beta Y$

What operations does LAProof provide?

TABLE I
LAPROOF VECTOR OPERATIONS

DOT	$r \leftarrow x \cdot y$
sVec	$r \leftarrow \alpha x$
SUM	$r \leftarrow \sum_i x_i$
VecAdd	$r \leftarrow x + y$
VecAXPBY	$r \leftarrow \alpha x + \beta y$
VecNRM1	$r \leftarrow \|x\|_1$
VecNRM2	$r \leftarrow \|x\|_2$

TABLE II
LAPROOF MATRIX-VECTOR OPERATIONS

MV	$r \leftarrow Ax$
sMV	$r \leftarrow \alpha Ax$
GEMV	$r \leftarrow \alpha Ax + \beta y$

TABLE III
LAPROOF MATRIX OPERATIONS

sMat	$R \leftarrow \alpha A$
MM	$R \leftarrow AB$
MatAdd	$R \leftarrow A + B$
sMM	$R \leftarrow \alpha AB$
MatAXPBY	$R \leftarrow \alpha X + \beta Y$
GEMM	$R \leftarrow \alpha AX + \beta Y$

What operations does LAProof provide?

TABLE I
LAPROOF VECTOR OPERATIONS

DOT	$r \leftarrow x \cdot y$
sVec	$r \leftarrow \alpha x$
SUM	$r \leftarrow \sum_i x_i$
VecAdd	$r \leftarrow x + y$
VecAXPBY	$r \leftarrow \alpha x + \beta y$
VecNRM1	$r \leftarrow \|x\|_1$
VecNRM2	$r \leftarrow \|x\|_2$

TABLE II
LAPROOF MATRIX-VECTOR OPERATIONS

MV	$r \leftarrow Ax$
sMV	$r \leftarrow \alpha Ax$
GEMV	$r \leftarrow \alpha Ax + \beta y$

TABLE III
LAPROOF MATRIX OPERATIONS

sMat	$R \leftarrow \alpha A$
MM	$R \leftarrow AB$
MatAdd	$R \leftarrow A + B$
sMM	$R \leftarrow \alpha AB$
MatAXPBY	$R \leftarrow \alpha X + \beta Y$
GEMM	$R \leftarrow \alpha AX + \beta Y$

What operations does LAProof provide?

TABLE I
LAPROOF VECTOR OPERATIONS

DOT	$r \leftarrow x \cdot y$
sVec	$r \leftarrow \alpha x$
SUM	$r \leftarrow \sum_i x_i$
VecAdd	$r \leftarrow x + y$
VecAXPBY	$r \leftarrow \alpha x + \beta y$
VecNRM1	$r \leftarrow \|x\|_1$
VecNRM2	$r \leftarrow \|x\|_2$

TABLE II
LAPROOF MATRIX-VECTOR OPERATIONS

MV	$r \leftarrow Ax$
sMV	$r \leftarrow \alpha Ax$
GEMV	$r \leftarrow \alpha Ax + \beta y$

TABLE III
LAPROOF MATRIX OPERATIONS

sMat	$R \leftarrow \alpha A$
MM	$R \leftarrow AB$
MatAdd	$R \leftarrow A + B$
sMM	$R \leftarrow \alpha AB$
MatAXPBY	$R \leftarrow \alpha X + \beta Y$
GEMM	$R \leftarrow \alpha AX + \beta Y$

What operations does LAProof provide?

TABLE I
LAPROOF VECTOR OPERATIONS

DOT	$r \leftarrow x \cdot y$
sVec	$r \leftarrow \alpha x$
SUM	$r \leftarrow \sum_i x_i$
VecAdd	$r \leftarrow x + y$
VecAXPBY	$r \leftarrow \alpha x + \beta y$
VecNRM1	$r \leftarrow \|\|x\|\|_1$
VecNRM2	$r \leftarrow \|\|x\|\|_2$

TABLE II
LAPROOF MATRIX-VECTOR OPERATIONS

MV	$r \leftarrow Ax$
sMV	$r \leftarrow \alpha Ax$
GEMV	$r \leftarrow \alpha Ax + \beta y$

TABLE III
LAPROOF MATRIX OPERATIONS

sMat	$R \leftarrow \alpha A$
MM	$R \leftarrow AB$
MatAdd	$R \leftarrow A + B$
sMM	$R \leftarrow \alpha AB$
MatAXPBY	$R \leftarrow \alpha X + \beta Y$
GEMM	$R \leftarrow \alpha AX + \beta Y$

How are LAProof operations defined?

Matrices and vectors are defined using polymorphic lists.

```
Definition matrix (T : Type) := list (list T).
Definition vector (T : Type) := list T.
```

- Coq's **axiomatic reals**
- Flocq's [Boldo & Melquiond, 2011] **IEEE-754 binary floats at *any* precision**

How are LAProof operations defined?

Matrices and vectors are defined using polymorphic lists.

```
Definition matrix (T : Type) := list (list T).
Definition vector (T : Type) := list T.
```

Operations are simple higher-order polymorphic functional programs.

matrix-vector product

```
Variable dot : vector T → vector T → T.
Variables (A : matrix T) (v : vector T).

Definition MV : vector T
        := map (fun a ⇒ dot a v) A.
```

How do we ensure the correctness of LAProof operations?

Extension to the Mathematical Components (MathComp) Library [Mahboubi et al., 2022]

The Mathematical Components repository

The Mathematical Components Library is an extensive and coherent repository of formalized mathematical theories. It is based on the Coq proof assistant, powered with the Coq/SSReflect language.

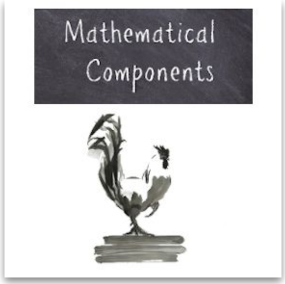
- Originally developed for formal proofs of the four color theorem and the odd order theorem
- Contains well developed and maintained libraries for real analysis and basic linear algebra

How do we ensure the correctness of LAProof operations?

How do we ensure the correctness of LAProof operations?

Extension to the Mathematical Components (MathComp) Library [Mahboubi et al., 2022]

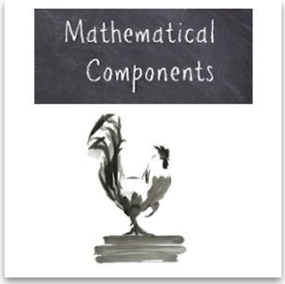
- Define injections from LAProof matrices and vectors over Coq's axiomatic reals to MathComp's matrices and vectors [Cohen et al., 2022]
- Prove that injections from LAProof operations to MathComp operations are correct
- Provides added confidence and functionality



How do we ensure the correctness of LAProof operations?

Extension to the Mathematical Components (MathComp) Library [Mahboubi et al., 2022]

- Define injections from LAProof matrices and vectors over Coq's axiomatic reals to MathComp's matrices and vectors [Cohen et al., 2022]
- Prove that injections from LAProof operations to MathComp operations are correct
- Provides added confidence and functionality



Why not use MathComp in the first place?

- Coq lists are easier to use in proofs of program correctness [Cohen et al., 2022]
- More Coq users are familiar with Coq lists than MathComp (and SSReflect)

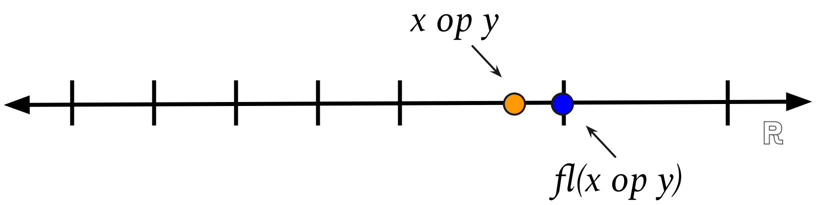
What accuracy theorems does LAProof provide?

What accuracy theorems does LAProof provide?

- Mixed (backward-forward) rounding error bounds that account for underflow
- Low order error terms captured exactly, not approximating as $O(u^2)$
- Assume only a low-level formal model of IEEE-754 arithmetic provided by the Flocq library [Boldo & Melquiond, 2011].

LAProof accuracy theorems rely on the correctness of the standard rounding error model

Flocq theorem: for IEEE arithmetic,



$$\mathrm{fl}(a \ op \ b) = (a \ op \ b)(1+\delta) + \epsilon$$

$$|\delta| \le u, \ |\epsilon| \le \eta, \ \delta\epsilon = 0, \ op \in \{+, -, \times, /, \surd\}$$

unit roundoff

underflow unit

What accuracy theorems does LAProof provide?

- Backward error bounds when possible (e.g., summation).

$$\phi_{\mathbb{F}_{p,e}}(x) = \phi_{\mathbb{R}}(x + \Delta x)$$

Floating-point operation:
precision p**, maximum exponent** e

Rounding error is attributed to a small change in the input.

What accuracy theorems does LAProof provide?

- Mixed (backward-forward) rounding error bounds that account for underflow.

$$\phi_{\mathbb{F}_{p,e}}(x) = \phi_{\mathbb{R}}(x + \Delta x) + \hat{\delta}$$

Floating-point operation:
precision p**, maximum exponent** e

accounts for underflow

Rounding error is attributed to a small change in the input, **plus some small term that accounts for underflow**.

What accuracy theorems does LAProof provide?

- Forward rounding error bounds are derived from mixed (or backward) error bounds.

$$F \triangleq |\phi_{\mathbb{R}}(x) - \phi_{\mathbb{F}_{p,e}}(x)|$$

Floating-point operation:
precision p, maximum exponent e

An example: accuracy of matrix-vector product

- Mixed (backward-forward) rounding error bounds that account for underflow.

$$\phi_{\mathbb{F}_{p,e}}(x) = \phi_{\mathbb{R}}(x + \Delta x) + \hat{\delta}$$

```
Definition MV : vector T
        := map (fun a ⇒ dot a v) A.
```

Rounding error is attributed to a small change in the input, **plus some small term that accounts for underflow**.

An example: accuracy of matrix-vector product

Theorem 3 (bfMV). *For any vector* $\boldsymbol{v} \in \mathbb{F}_{p,e}^{n}$, *and matrix* $\boldsymbol{A} \in \mathbb{F}_{p,e}^{m \times n}$, *there exists a matrix* $\boldsymbol{\Delta A} \in \mathbb{R}^{m \times n}$ *and vector* $\boldsymbol{\eta} \in \mathbb{R}^{n}$ *such that*

$$\mathrm{fl}(\boldsymbol{A}\boldsymbol{v}) = (\boldsymbol{A} + \boldsymbol{\Delta A})\boldsymbol{v} + \boldsymbol{\eta}, \tag{11}$$

where every element of the vector $\boldsymbol{\eta}$ *respects the bound* $|\eta| \leq g(n, n)$ *and each element of the matrix* $\boldsymbol{\Delta A}$ *respects the bound* $|\Delta A| \leq h(n)|A|$.

An example: accuracy of matrix-vector product

Theorem 3 (bfMV). *For any vector* $\boldsymbol{v} \in \mathbb{F}^n_{p,e}$, *and matrix* $\boldsymbol{A} \in \mathbb{F}^{m\times n}_{p,e}$, *there exists a matrix* $\boldsymbol{\Delta A} \in \mathbb{R}^{m\times n}$ *and vector* $\boldsymbol{\eta} \in \mathbb{R}^n$ *such that*

$$\mathrm{fl}(\boldsymbol{Av}) = (\boldsymbol{A} + \boldsymbol{\Delta A})\boldsymbol{v} + \boldsymbol{\eta}, \tag{11}$$

where every element of the vector $\boldsymbol{\eta}$ *respects the bound* $|\eta| \le g(n,n)$ *and each element of the matrix* $\boldsymbol{\Delta A}$ *respects the bound* $|\Delta A| \le h(n)|A|$.

$$h(n) = (1+u)^n - 1$$
$$g(n,m) = n\eta(1 + h(m))$$

What does the theorem look like in Coq?

```
Variable (A: @matrix (ftype t)).
Variable (v: @vector (ftype t)).

Hypothesis Hfin : is_finite_vec (A *f v).
Hypothesis Hlen: forall row, In row A -> length row = length v.

Lemma mat_vec_mul_mixed_error:
  exists (E : matrix) (eta : vector),
    A *fr v =  (Ar +m E) *r vr +v eta
    /\ (forall i j, (i < m)%nat -> (j < n)%nat ->
      Rabs (E _(i,j)) <= g n * Rabs (Ar _(i,j)))
    /\ (forall k, In k eta -> Rabs k <= g1 n n)
    /\ eq_size E A
    /\ length eta = m.
```

What does the theorem look like in Coq?

```
Variable (A: @matrix (ftype t)).
Variable (v: @vector (ftype t)).

Hypothesis Hfin : is_finite_vec (A *f v).
Hypothesis Hlen: forall row, In row A -> length row = length v.

Lemma mat_vec_mul_mixed_error:
  exists (E : matrix) (eta : vector),
    A *fr v =  (Ar +m E) *r vr +v eta
    /\ (forall i j, (i < m)%nat -> (j < n)%nat ->
      Rabs (E _(i,j)) <= g n * Rabs (Ar _(i,j)))
    /\ (forall k, In k eta -> Rabs k <= g1 n n)
    /\ eq_size E A
    /\ length eta = m.
```

What does the theorem look like in Coq?

```
Variable (A: @matrix (ftype t)).
Variable (v: @vector (ftype t)).

Hypothesis Hfin : is_finite_vec (A *f v).
Hypothesis Hlen: forall row, In row A -> length row = length v.

Lemma mat_vec_mul_mixed_error:
  exists (E : matrix) (eta : vector),
    A *fr v =  (Ar +m E) *r vr +v eta
    /\ (forall i j, (i < m)%nat -> (j < n)%nat ->
      Rabs (E _(i,j)) <= g n * Rabs (Ar _(i,j)))
    /\ (forall k, In k eta -> Rabs k <= g1 n n)
    /\ eq_size E A
    /\ length eta = m.
```

What does the theorem look like in Coq?

```
Variable (A: @matrix (ftype t)).
Variable (v: @vector (ftype t)).

Hypothesis Hfin : is_finite_vec (A *f v).
Hypothesis Hlen: forall row, In row A -> length row = length v.

Lemma mat_vec_mul_mixed_error:
  exists (E : matrix) (eta : vector),
    A *fr v =  (Ar +m E) *r vr +v eta
    /\ (forall i j, (i < m)%nat -> (j < n)%nat ->
      Rabs (E _(i,j)) <= g n * Rabs (Ar _(i,j)))
    /\ (forall k, In k eta -> Rabs k <= g1 n n)
    /\ eq_size E A
    /\ length eta = m.
```

What does the theorem look like in Coq?

```
Variable (A: @matrix (ftype t)).
Variable (v: @vector (ftype t)).

Hypothesis Hfin : is_finite_vec (A *f v).
Hypothesis Hlen: forall row, In row A -> length row = length v.

Lemma mat_vec_mul_mixed_error:
  exists (E : matrix) (eta : vector),
    A *fr v =  (Ar +m E) *r vr +v eta
    /\ (forall i j, (i < m)%nat -> (j < n)%nat ->
      Rabs (E _(i,j)) <= g n * Rabs (Ar _(i,j)))
    /\ (forall k, In k eta -> Rabs k <= g1 n n)
    /\ eq_size E A
    /\ length eta = m.
```

What does the theorem look like in Coq?

```
Variable (A: @matrix (ftype t)).
Variable (v: @vector (ftype t)).

Hypothesis Hfin : is_finite_vec (A *f v).
Hypothesis Hlen: forall row, In row A -> length row = length v.

Lemma mat_vec_mul_mixed_error:
  exists (E : matrix) (eta : vector),
    A *fr v =  (Ar +m E) *r vr +v eta
    /\ (forall i j, (i < m)%nat -> (j < n)%nat ->
      Rabs (E _(i,j)) <= g n * Rabs (Ar _(i,j)))
    /\ (forall k, In k eta -> Rabs k <= g1 n n)
    /\ eq_size E A
    /\ length eta = m.
```

What does the theorem look like in Coq?

```
Variable (A: @matrix (ftype t)).
Variable (v: @vector (ftype t)).

Hypothesis Hfin : is_finite_vec (A *f v).
Hypothesis Hlen: forall row, In row A -> length row = length v.

Lemma mat_vec_mul_mixed_error:
  exists (E : matrix) (eta : vector),
    A *fr v =  (Ar +m E) *r vr +v eta
    /\ (forall i j, (i < m)%nat -> (j < n)%nat ->
      Rabs (E _(i,j)) <= g n * Rabs (Ar _(i,j)))
    /\ (forall k, In k eta -> Rabs k <= g1 n n)
    /\ eq_size E A
    /\ length eta = m.
```

What does the theorem look like in Coq?

```
Variable (A: @matrix (ftype t)).
Variable (v: @vector (ftype t)).

Hypothesis Hfin : is_finite_vec (A *f v).
Hypothesis Hlen: forall row, In row A -> length row = length v.

Lemma mat_vec_mul_mixed_error:
  exists (E : matrix) (eta : vector),
    A *fr v =  (Ar +m E) *r vr +v eta
    /\ (forall i j, (i < m)%nat -> (j < n)%nat ->
      Rabs (E _(i,j)) <= g n * Rabs (Ar _(i,j)))
    /\ (forall k, In k eta -> Rabs k <= g1 n n)
    /\ eq_size E A
    /\ length eta = m.
```

- Formal proof ~120 lines of code

How can we connect error bounds from LAProof to concrete programs?

Use LAProof operations in proofs of program correctness

Example from the paper: prove the correctness of the function `csr_mv_multiply`, which implements matrix-vector multiplication using a compressed sparse row (CSR) format .

```
void csr_mv_multiply (struct csr_matrix *m,
        double *v,  double *p) {
  unsigned i, rows = m → rows;
  double *val = m → val;
  unsigned *col_ind = m → col_ind;
  unsigned *row_ptr = m → row_ptr;
  unsigned next=row_ptr[0];
  for (i = 0; i < rows; i++) {
    double s = 0.0;
    unsigned h = next;
    next = row4_ptr[i+1];
    for (h = 0; h < next; h++) {
      double x = val[h];
      unsigned j = col_ind[h];
      double y = v[j];
      s = fma(x,y,s);
    }
    p[i]=s;
} }
```

Use LAProof operations in proofs of program correctness

Example from the paper: prove the correctness of the function `csr_mv_multiply`, which implements matrix-vector multiplication using a compressed sparse row (CSR) format .

- Write a specification of matrix-vector multiplication using the LAProof operation.
- Prove (in Coq) that `csr_mv_multiply` complies with this specification.

```
void csr_mv_multiply (struct csr_matrix *m,
        double *v,  double *p) {
  unsigned i, rows = m → rows;
  double *val = m → val;
  unsigned *col_ind = m → col_ind;
  unsigned *row_ptr = m → row_ptr;
  unsigned next=row_ptr[0];
  for (i = 0; i < rows; i++) {
    double s = 0.0;
    unsigned h = next;
    next = row4_ptr[i+1];
    for (h = 0; h < next; h++) {
      double x = val[h];
      unsigned j = col_ind[h];
      double y = v[j];
      s = fma(x,y,s);
    }
    p[i]=s;
} }
```

Example: Prove the correctness of `csr_mv_multiply` using VST

The Verified Software Toolchain (VST) [Appel et al., 2011]

- A collection of verification tools for the C language
- Implements (in Coq) a program logic for reasoning about the correctness of C programs
- Proved sound with respect to the CompCert C compiler [Leroy et al., 2008]

```
Definition csr_mv_spec :=
DECLARE _csr_mv_multiply
WITH π1: share, π2: share, π3: share,
      m: val, A: matrix Tdouble, v: val,
      x: vector Tdouble, p: val
PRE [ tptr t_csr, tptr tdouble, tptr tdouble ]
```

```
POST [ tvoid ]
 EX y: vector Tdouble,
  PROP(Forall2 feq y (MVF A x))
  RETURN()
  SEP (csr_rep π1 A m;
       data_at π2 (tarray tdouble (Zlength x))
                                (map Vfloat x) v;
       data_at π3 (tarray tdouble (matrix_rows A))
                                (map Vfloat y) p).
```

Example: Prove the correctness of `csr_mv_multiply` using VST

A function is specified by its **precondition** and its **postcondition**

```
Definition csr_mv_spec :=
DECLARE _csr_mv_multiply
WITH π1: share, π2: share, π3: share,
      m: val, A: matrix Tdouble, v: val,
      x: vector Tdouble, p: val
PRE [ tptr t_csr, tptr tdouble, tptr tdouble ]
      ⋮
POST [ tvoid ]
  EX y: vector Tdouble,
   PROP(Forall2 feq y (MVF A x))
   RETURN()
   SEP (csr_rep π1 A m;
        data_at π2 (tarray tdouble (Zlength x))
                                     (map Vfloat x) v;
        data_at π3 (tarray tdouble (matrix_rows A))
                                     (map Vfloat y) p).
```

Example: Prove the correctness of `csr_mv_multiply` using VST

A function is specified by its **precondition** and its **postcondition**

- A, x: formal models of the matrix and vector begin multiplied.

```
Definition csr_mv_spec :=
DECLARE _csr_mv_multiply
WITH π1: share, π2: share, π3: share,
      m: val, A: matrix Tdouble, v: val,
      x: vector Tdouble, p: val
PRE [ tptr t_csr, tptr tdouble, tptr tdouble ]
```

```
POST [ tvoid ]
 EX y: vector Tdouble,
  PROP(Forall2 feq y (MVF A x))
  RETURN()
  SEP (csr_rep π1 A m;
       data_at π2 (tarray tdouble (Zlength x))
                                (map Vfloat x) v;
       data_at π3 (tarray tdouble (matrix_rows A))
                                (map Vfloat y) p).
```

Example: Prove the correctness of `csr_mv_multiply` using VST

A function is specified by its **precondition** and its **postcondition**

- A, x: formal models of the matrix and vector begin multiplied.
- `m`: address where CSR representation of A is stored
- `p`: address where vector x is stored

```
Definition csr_mv_spec :=
DECLARE _csr_mv_multiply
WITH π1: share, π2: share, π3: share,
      m: val, A: matrix Tdouble, v: val,
      x: vector Tdouble, p: val
PRE [ tptr t_csr, tptr tdouble, tptr tdouble ]
```

```
POST [ tvoid ]
 EX y: vector Tdouble,
  PROP(Forall2 feq y (MVF A x))
  RETURN()
  SEP (csr_rep π1 A m;
       data_at π2 (tarray tdouble (Zlength x))
                              (map Vfloat x) v;
       data_at π3 (tarray tdouble (matrix_rows A))
                              (map Vfloat y) p).
```

Example: Prove the correctness of `csr_mv_multiply` using VST

A function is specified by its **precondition** and its **postcondition**

- A, x: formal models of the matrix and vector begin multiplied.
- `m`: address where CSR representation of A is stored
- `p`: address where vector x is stored
- **postcondition**: the vector of y of double precision floats exists, *and…*

```
Definition csr_mv_spec :=
DECLARE _csr_mv_multiply
WITH π1: share, π2: share, π3: share,
      m: val, A: matrix Tdouble, v: val,
      x: vector Tdouble, p: val
PRE [ tptr t_csr, tptr tdouble, tptr tdouble ]
```

```
Definition MV : vector T
      := map (fun a ⇒ dot a v) A.
```

```
POST [ tvoid ]
 EX y: vector Tdouble,
  PROP(Forall2 feq y (MVF A x))
  RETURN()
  SEP (csr_rep π1 A m;
       data_at π2 (tarray tdouble (Zlength x))
                          (map Vfloat x) v;
       data_at π3 (tarray tdouble (matrix_rows A))
                          (map Vfloat y) p).
```

Accuracy and correctness proofs compose

Theorem [accuracy *and* correctness]: the function `csr_mv_multiply` correctly and accurately implements matrix-vector multiplication using a compressed sparse row format.

```
void csr_mv_multiply (struct csr_matrix *m,
        double *v,  double *p) {
  unsigned i, rows = m → rows;
  double *val = m → val;
  unsigned *col_ind = m → col_ind;
  unsigned *row_ptr = m → row_ptr;
  unsigned next=row_ptr[0];
  for (i = 0; i < rows; i++) {
    double s = 0.0;
    unsigned h = next;
    next = row4_ptr[i+1];
    for (h = 0; h < next; h++) {
      double x = val[h];
      unsigned j = col_ind[h];
      double y = v[j];
      s = fma(x,y,s);
    }
    p[i]=s;
} }
```

In summary,

LAProof provides machine-checked proofs of accuracy for basic linear algebra operations and these accuracy proofs can be connected to concrete programs implementing BLAS.

- Accuracy proofs assume only a low-level formal model of IEEE-754 arithmetic.
- The rounding error bounds in the accuracy proofs are mixed (backward-forward) error bounds that account for underflow.
- Rounding error bounds capture low order error terms exactly, not approximating as $O(u^2)$.

VeriNum / LAProof

Thanks for listening!

My co-authors are Andrew W. Appel, Mohit Tekriwal, and David Bindel

PRINCETON UNIVERSITY UNIVERSITY OF MICHIGAN Cornell University

We acknowledge the generous support of

NSF DOE CSGF Sandia National Laboratories