Verified Computer Linear Algebra[1]

Jesús Aransay, Jose Divasón

UNIVERSIDAD DE LA RIOJA

XV Encuentro de Álgebra Computacional y Aplicaciones
Universidad de La Rioja, Logroño, 23rd June 2016

[1]This work has been supported by project MTM2014-54151-P from Ministerio de Economía y Competitividad (Gobierno de España)

Motivation

- Isabelle/HOL has a number of Libraries that deal with Algebra and Multivariate Analysis

Motivation

- Isabelle/HOL has a number of Libraries that deal with Algebra and Multivariate Analysis
- On the contrary, Linear Algebra algorithms had been superficially explored or implemented

Motivation

- Isabelle/HOL has a number of Libraries that deal with Algebra and Multivariate Analysis
- On the contrary, Linear Algebra algorithms had been superficially explored or implemented
- Linear Algebra algorithms can be applied to compute properties of linear maps and matrices

Motivation

- Isabelle/HOL has a number of Libraries that deal with Algebra and Multivariate Analysis
- On the contrary, Linear Algebra algorithms had been superficially explored or implemented
- Linear Algebra algorithms can be applied to compute properties of linear maps and matrices
- We aim at improving the performance of verified algorithms with:

Motivation

- Isabelle/HOL has a number of Libraries that deal with Algebra and Multivariate Analysis
- On the contrary, Linear Algebra algorithms had been superficially explored or implemented
- Linear Algebra algorithms can be applied to compute properties of linear maps and matrices
- We aim at improving the performance of verified algorithms with:
    - the available tools in conventional functional programming languages (SML, Haskell)

Motivation

- Isabelle/HOL has a number of Libraries that deal with Algebra and Multivariate Analysis
- On the contrary, Linear Algebra algorithms had been superficially explored or implemented
- Linear Algebra algorithms can be applied to compute properties of linear maps and matrices
- We aim at improving the performance of verified algorithms with:
  - the available tools in conventional functional programming languages (SML, Haskell)
  - the available tools in Isabelle/HOL

Methodology

- A Linear Algebra result is chosen and formalised

Methodology

- A Linear Algebra result is chosen and formalised
- A *naive* algorithm, performing matrix elementary operations, is implemented

Methodology

- A Linear Algebra result is chosen and formalised
- A *naive* algorithm, performing matrix elementary operations, is implemented
- The algorithm and its mathematical meaning are formalised

Methodology

- A Linear Algebra result is chosen and formalised
- A *naive* algorithm, performing matrix elementary operations, is implemented
- The algorithm and its mathematical meaning are formalised
- The algorithm and its data structures are *refined* to better performing ones

Methodology

- A Linear Algebra result is chosen and formalised
- A *naive* algorithm, performing matrix elementary operations, is implemented
- The algorithm and its mathematical meaning are formalised
- The algorithm and its data structures are *refined* to better performing ones
- The optimised version is generated to functional languages, and applied to case studies

Toolkit

- Proof assistant: Isabelle (L. Paulson, T. Nipkow, M. Wenzel)
- Underlying logic: Higher-order logic (HOL) + type classes
- Additional libraries: HOL Multivariate Analysis (HMA, J. Harrison)
- Code generation infrastructure (F. Haftmann)
- Proof language: Intelligible semi-automated reasoning (Isar, M. Wenzel)
- Execution environments: GH(askell)C, PolyML (D. Matthews) and MLton

HMA - Multivariate Analysis session

- Our formalisations are based on the HOL Multivariate Analysis session
- Nice vector and matrix representation from the formalisation point of view

HMA - Multivariate Analysis session

- Our formalisations are based on the HOL Multivariate Analysis session
- Nice vector and matrix representation from the formalisation point of view

typedef (α,β) vec = UNIV :: ((β::finite) $\Rightarrow$ α) set
 morphisms vec−nth vec−lambda ..

- Type System vs Logic

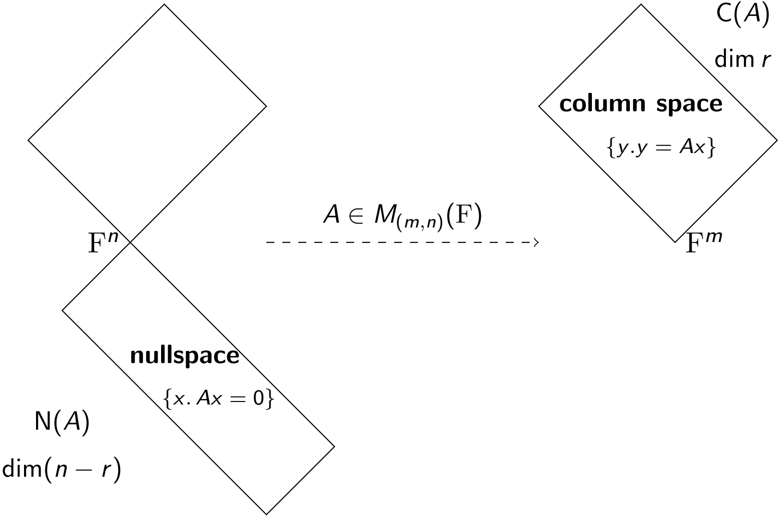

Figure : Rank Nullity Theorem

From theorems to algorithms

- Gauss-Jordan elimination provides a direct way to compute $\dim(\mathrm{C}\,(A))$ by means of *elementary row operations* over $A \in M_{(m,n)}(\mathbb{F})$

From theorems to algorithms

- Gauss-Jordan elimination provides a direct way to compute $\dim(\mathrm{C}(A))$ by means of *elementary row operations* over $A \in M_{(m,n)}(\mathbb{F})$

Gauss-Jordan example

$$A = \begin{pmatrix} 1 & -2 & 1 & -3 & 0 \\ 3 & -6 & 2 & -7 & 0 \\ 5 & -1 & 3 & 2 & 5 \\ 0 & 7 & 4 & 5 & 1 \\ 3 & -6 & 2 & -7 & 0 \end{pmatrix} \longrightarrow A = \begin{pmatrix} 1 & 0 & 0 & 0 & 3 \\ 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 1 & 0 & -2 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

From theorems to algorithms

- Gauss-Jordan elimination provides a direct way to compute $\dim(\mathrm{C}\,(A))$ by means of *elementary row operations* over $A \in M_{(m,n)}(\mathbb{F})$

Gauss-Jordan example

$$A = \begin{pmatrix} 1 & -2 & 1 & -3 & 0 \\ 3 & -6 & 2 & -7 & 0 \\ 5 & -1 & 3 & 2 & 5 \\ 0 & 7 & 4 & 5 & 1 \\ 3 & -6 & 2 & -7 & 0 \end{pmatrix} \longrightarrow A = \begin{pmatrix} 1 & 0 & 0 & 0 & 3 \\ 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 1 & 0 & -2 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\dim(\mathrm{C}\,(A)) = 4$$

Elementary operations

Most Linear Algebra algorithms can be implemented using exclusively elementary row (column) operations on matrices, *i.e.*

Elementary row (column) operations

- Interchange two different rows (columns)
- Multiply a row (column) by an invertible element
- Add to a row (column) another one multiplied by a constant

We have implemented these operations and their properties in Isabelle. These are later on used to formalise, execute and refine algorithms.

The following matrices (over real numbers) computations can be performed by means of the Gauss-Jordan algorithm

The following matrices (over real numbers) computations can be performed by means of the Gauss-Jordan algorithm

Gauss-Jordan algorithm applications

- Ranks
- Determinants
- Inverses
- Dimensions and bases of the null space, left null space, column space and row space
- Solution(s) of systems of linear equations

Relying on *invariants*

Elementary row operations do not modify, or modify in a predictable way, the previous computations

Isabelle lemmas about elementary operations

lemma crk-is-preserved:
 fixes A::real^'cols::{finite, wellorder}^'rows **and** P::real^'rows^'rows
 assumes inv-P: invertible P
 shows col-rank A = col-rank (P ** A)

Relying on *invariants*

Elementary row operations do not modify, or modify in a predictable way, the previous computations

Isabelle lemmas about elementary operations

lemma crk-is-preserved:
 fixes A::real^′cols::{finite, wellorder}^′rows **and** P::real^′rows^′rows
 assumes inv-P: invertible P
 shows col-rank A = col-rank (P ∗∗ A)

lemma det-mult-row:
 shows det (mult-row A a k) = k ∗ det A

Relying on *invariants*

Elementary row operations do not modify, or modify in a predictable way, the previous computations

Isabelle lemmas about elementary operations

lemma crk-is-preserved:
 fixes A::real^'cols::{finite, wellorder}^'rows **and** P::real^'rows^'rows
 assumes inv-P: invertible P
 shows col-rank A = col-rank (P ** A)

lemma det-mult-row:
 shows det (mult-row A a k) = k * det A

lemma matrix-inv-Gauss-Jordan-PA:
 fixes A::real^'n::{mod-type}^'n::{mod-type}
 assumes inv-A: invertible A
 shows matrix-inv A = fst (Gauss-Jordan-PA A)

Relying on the properties of Gauss-Jordan

Isabelle lemmas to compute the previous properties

definition basis-null-space A $=$
$\{$row i (P-Gauss-Jordan (transpose A)) $|$ i. to-nat i $\geq$ rank A$\}$
definition basis-row-space A $=$
$\{$row i (Gauss-Jordan A) $|$i. row i (Gauss-Jordan A) $\neq 0\}$

Relying on the properties of Gauss-Jordan

Isabelle lemmas to compute the previous properties

definition basis-null-space A =
$\{$row i (P-Gauss-Jordan (transpose A)) $|$ i. to-nat i $\geq$ rank A$\}$
definition basis-row-space A =
$\{$row i (Gauss-Jordan A) $|$i. row i (Gauss-Jordan A) $\neq$ 0$\}$

definition solve-system A b =
(let GJ = Gauss-Jordan-PA A in (snd GJ, (fst GJ) $*$v b))
definition solve A b = (if consistent A b then
Some (solve-consistent-rref (fst (solve-system A b)) (snd (solve-system A b)), basis-null-space A)
else None)

Refinement

$\boxed{\text{Abstract representation}}$

Refinement

$\boxed{\text{Abstract representation}} \cdots\cdots\!\!>$

Refinement

$\boxed{\text{Abstract representation}} \dashrightarrow \boxed{\text{Abstract definitions}}$

Refinement

$\boxed{\text{Abstract representation}} \dashrightarrow \boxed{\text{Abstract definitions}} \longrightarrow \text{Proof}$

Refinement

$$\begin{tikzcd} \boxed{\text{Abstract representation}} \arrow[r, dotted] & \boxed{\text{Abstract definitions}} \arrow[r] & \text{Proof} \\ \boxed{\text{Concrete representation}} & & \end{tikzcd}$$

Refinement

$$\begin{tikzcd}
\boxed{\text{Abstract representation}} \arrow[r, dotted] \arrow[d, "\textit{Projection}"] & \boxed{\text{Abstract definitions}} \arrow[r] & \text{Proof} \\
\boxed{\text{Concrete representation}} & &
\end{tikzcd}$$

Refinement

$$
\begin{tikzcd}
\boxed{\text{Abstract representation}} \arrow[r, dotted] \arrow[d, "\textit{Projection}"] & \boxed{\text{Abstract definitions}} \arrow[r] & \text{Proof} \\
\boxed{\text{Concrete representation}} \arrow[r, dotted] & {} &
\end{tikzcd}
$$

Refinement

$$
\begin{tikzcd}
\boxed{\text{Abstract representation}} \arrow[r, dotted] \arrow[d, "\textit{Projection}"] & \boxed{\text{Abstract definitions}} \arrow[r] & \text{Proof} \\
\boxed{\text{Concrete representation}} \arrow[r, dotted] & \boxed{\text{Concrete definitions}} &
\end{tikzcd}
$$

Refinement

$$\begin{tikzcd} \boxed{\text{Abstract representation}} \arrow[r, dotted] \arrow[d, "\textit{Projection}"] & \boxed{\text{Abstract definitions}} \arrow[r] \arrow[d, "\textit{Code lemmas}"] & \text{Proof} \\ \boxed{\text{Concrete representation}} \arrow[r, dotted] & \boxed{\text{Concrete definitions}} & \end{tikzcd}$$

Refinement

$$
\begin{tikzcd}
\boxed{\text{Abstract representation}} \arrow[r, dotted] \arrow[d, "\textit{Projection}"] & \boxed{\text{Abstract definitions}} \arrow[r] \arrow[d, "\textit{Code lemmas}"] & \text{Proof} \\
\boxed{\text{Concrete representation}} \arrow[r, dotted] & \boxed{\text{Concrete definitions}} \arrow[r] & \text{Execution}
\end{tikzcd}
$$

Refinement

$$
\begin{tikzcd}
\boxed{\text{Abstract representation}} \arrow[r, dotted] \arrow[d, "\textit{Projection}"] & \boxed{\text{Abstract definitions}} \arrow[r] \arrow[d, "\textit{Code lemmas}"] & \text{Proof} \\
\boxed{\text{Concrete representation}} \arrow[r, dotted] & \boxed{\text{Concrete definitions}} \arrow[r] & \text{Execution}
\end{tikzcd}
$$

A refinement has been carried out so that operations over the abstract type *vec* can be executed

1. Mapping *vec* to *iarray*

From vec to iarray

In order to achieve better performance, a new refinement has been developed using immutable arrays

- There exists a datatype in the Isabelle library called *iarray* which represents immutable arrays
- *iarray* is implemented in both SML (*Vector structure*) and Haskell (*IArray class*)
- We have refined *vec* elements and operations to *iarray* ones (proving the corresponding morphisms)

From vec to iarray

In order to achieve better performance, a new refinement has been developed using immutable arrays

- There exists a datatype in the Isabelle library called *iarray* which represents immutable arrays
- *iarray* is implemented in both SML (*Vector structure*) and Haskell (*IArray class*)
- We have refined *vec* elements and operations to *iarray* ones (proving the corresponding morphisms)

Features of this refinement

1. Code can be generated to both SML and Haskell
2. Both *vec* and *iarray* have a similar functional flavour (for instance, in access operations)

$$\begin{tikzcd}
\alpha\ \hat{}\text{cols}\ \hat{}\text{rows} & \text{Gauss_Jordan} & \text{Isabelle} \\
\boxed{\text{Abstract representation}} \arrow[r, dotted] \arrow[d, "\textit{Projection}"] & \boxed{\text{Abstract definitions}} \arrow[r] \arrow[d, "\textit{Code lemmas}"] & \text{Proof} \\
\boxed{\text{Concrete representation}} \arrow[r, dotted] & \boxed{\text{Concrete definitions}} \arrow[r] & \text{Execution} \\
\alpha\ \text{iarray iarray} & \text{Gauss_Jordan_iarrays} & \text{Isabelle,SML,Haskell}
\end{tikzcd}$$

Serialisations

Isabelle/HOL	SML	Haskell
iarray	*Vector.vector*	**IArray.Array**
rat	*IntInf.int / IntInf.int*	**Rational**
real	*Real.real*	**Double**
bit	**Bool.bool**	**Bool**

Table : Type serialisations

Judgement Day

Size (n)	Poly/ML	MLton	GHC
100	0.04	0.05	0.36
200	0.25	0.32	2.25
400	2.01	2.35	17.17
800	15.96	18.57	131.73
1 200	62.33	70.45	453.57
1 600	139.70	152.41	1097.41
2 000	284.28	287.44	2295.30

Table : Elapsed time (in seconds) to compute the *rref* of randomly generated $\mathbb{Z}_2^{n \times n}$ matrices using iarrays

Judgement Day

Some relevant facts

- Both in SML and GHC we have serialised $\mathbb{Z}_2$ to *bool*

Judgement Day

Some relevant facts

- Both in SML and GHC we have serialised $\mathbb{Z}_2$ to *bool*
- Compilation in GHC is performed using compilation optimisations such as *-o3* (`https://wiki.haskell.org/Performance/GHC`)

Judgement Day

Size (n)	Poly/ML	Haskell
10	0.01	0.01
20	0.02	0.03
40	0.21	0.24
60	1.16	1.09
80	3.77	3.53
100	9.75	9.03

Table : Elapsed time (in seconds) to compute the *rref* of randomly generated $\mathbb{Q}^{n \times n}$ matrices.

Judgment Day

Some relevant facts

- In SML we have serialised $\mathbb{Q}$ to quotients of *IntInf.int*
- In GHC we have serialised $\mathbb{Q}$ to *Rational*

Judgment Day

Some relevant facts

- In SML we have serialised $\mathbb{Q}$ to quotients of *IntInf.int*
- In GHC we have serialised $\mathbb{Q}$ to *Rational*
- Both Poly/ML and GHC internally use GMP (`https://gmplib.org/`) to enhance performance

Judgment Day

Some relevant facts

- In SML we have serialised $\mathbb{Q}$ to quotients of *IntInf.int*
- In GHC we have serialised $\mathbb{Q}$ to *Rational*
- Both Poly/ML and GHC internally use GMP (`https://gmplib.org/`) to enhance performance
- We have optimised SML code by using some particular serialisations (computing the Isabelle *divmod* is done in Poly/ML by means of *IntInf.quotrem*)

Judgment Day

Size (n)	Poly/ML	Haskell
100	0.03	0.38
200	0.25	2.62
300	0.81	8.47
400	1.85	19.51
500	3.51	37.13
600	6.03	64.13
700	9.57	100.59
800	13.99	148.20

Table : Time to compute the *rref* of randomly generated $\mathbb{R}$ matrices.

Judgment Day

Some relevant facts

- In SML we have serialised $\mathbb{R}$ to *Real.real*
- In GHC we have serialised $\mathbb{R}$ to *Prelude.Double* (see for instance `http://www.isa-afp.org/browser_info/current/AFP/Gauss_Jordan/Code_Real_Approx_By_Float_Haskell.html`)

Judgment Day

Some relevant facts

- In SML we have serialised $\mathbb{R}$ to *Real.real*
- In GHC we have serialised $\mathbb{R}$ to *Prelude.Double* (see for instance `http://www.isa-afp.org/browser_info/current/AFP/Gauss_Jordan/Code_Real_Approx_By_Float_Haskell.html`)
- WARNING: use the obtained code at your own risk

C++ vs Verified version

Matrix sizes	C++ version	Verified version
600×600	01.33s.	06.16s.
$1\,000 \times 1\,000$	05.94s.	32.08s.
$1\,200 \times 1\,200$	10.28s.	62.33s.
$1\,400 \times 1\,400$	16.62s.	97.16s.

Table : C++ vs verified version of the Gauss-Jordan algorithm.

Both programs show a cubic performance, even if the verified version is using immutable arrays

Introduction

Gauss-Jordan

QR Decomposition

Conclusions

Theorem (Second Part of the Fundamental Theorem of Linear Algebra)

Given a matrix $A \in M_{(m,n)}(\mathbb{R})$

- *In* $\mathbb{R}^n$, $\mathrm{N}(A) = \mathrm{C}(A^T)^\perp$ *that is, the nullspace is the orthogonal complement of the row space*

Theorem (Second Part of the Fundamental Theorem of Linear Algebra)

Given a matrix $A \in M_{(m,n)}(\mathbb{R})$

- *In* $\mathbb{R}^n$, $\mathrm{N}(A) = \mathrm{C}(A^T)^\perp$ *that is, the nullspace is the orthogonal complement of the row space*
- *In* $\mathbb{R}^m$, $\mathrm{N}(A^T) = \mathrm{C}(A)^\perp$, *that is, the left nullspace is the orthogonal complement of the column space*

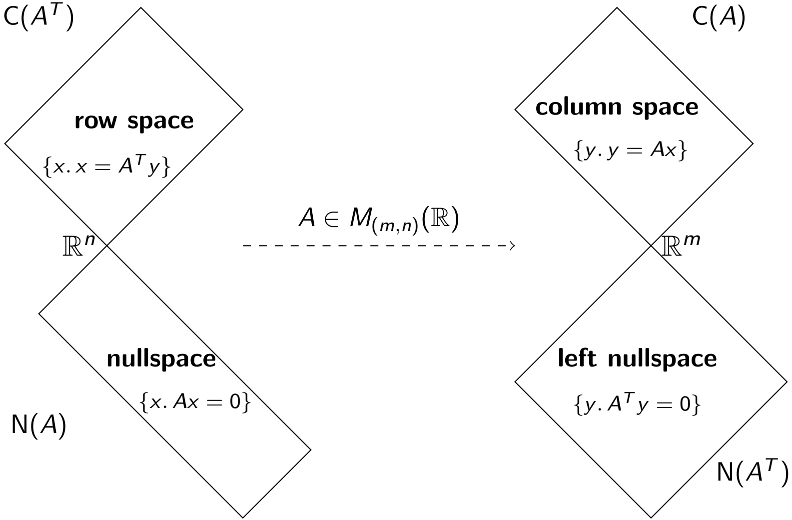

Figure : Orthogonality of the Four Fundamental subspaces

Second Part of the Fundamental Theorem of Linear Algebra

- **theorem** null-space-orthogonal-complement-row-space:
  fixes A :: real^'cols^'rows
  shows null-space A = orthogonal-complement (row-space A)

Second Part of the Fundamental Theorem of Linear Algebra

- **theorem** null-space-orthogonal-complement-row-space:
  fixes A :: real^'cols^'rows
  shows null-space A = orthogonal-complement (row-space A)
- **theorem** left-null-space-orthogonal-complement-col-space:
  fixes A :: real^'cols^'rows
  shows left-null-space A = orthogonal-complement (col-space A)

Second Part of the Fundamental Theorem of Linear Algebra

- **theorem** null-space-orthogonal-complement-row-space:
  fixes A :: real^'cols^'rows
  shows null-space A = orthogonal-complement (row-space A)
- **theorem** left-null-space-orthogonal-complement-col-space:
  fixes A :: real^'cols^'rows
  shows left-null-space A = orthogonal-complement (col-space A)

From Mathematical results to algorithms

The *Gram-Schmidt process* allows us to compute the mentioned orthogonal bases

QR decomposition

Definition (QR decomposition)

The QR decomposition of a full column rank matrix $A \in M_{n\times m}(\mathbb{R})$ is a pair of matrices (Q, R) such that

1. $A = QR$
2. $Q \in M_{n\times m}(\mathbb{R})$ is a matrix whose columns are orthonormal vectors
3. $R \in M_{m\times m}(\mathbb{R})$ is upper triangular and invertible

QR decomposition

Definition (QR decomposition)

The QR decomposition of a full column rank matrix $A \in M_{n\times m}(\mathbb{R})$ is a pair of matrices (Q, R) such that

1. $A = QR$
2. $Q \in M_{n\times m}(\mathbb{R})$ is a matrix whose columns are orthonormal vectors
3. $R \in M_{m\times m}(\mathbb{R})$ is upper triangular and invertible

Algorithm

1. $Q =$ Apply Gram-Schmidt to the columns of A, normalise the vectors
2. Compute R as $R = Q^T A$

QR Decomposition

- We have formalised the previous algorithm in Isabelle, and refined it to immutable arrays
- Computations can be carried out using either floats or (for suitable inputs) symbolically

QR Decomposition

- We have formalised the previous algorithm in Isabelle, and refined it to immutable arrays
- Computations can be carried out using either floats or (for suitable inputs) symbolically

René Thiemann. Implementing field extensions of the form $\mathbb{Q}[\sqrt{b}]$.
Archive of Formal Proofs (2014)

QR Decomposition

definition A :: real^4^4
  where foo = list-of-list-to-matrix [[1,2,4,6],
[9,4,5,2],
[0,0,4,3],
[3,2,4,1]]

value matrix-to-list-of-list (show-matrix-real (fst (QR-decomposition A)))
output

[[1/91∗sqrt(91), 69/5642∗sqrt(5642), −9/15934∗sqrt(15934), 6/257∗sqrt(257)],
[9/91∗sqrt(91), −8/2821∗sqrt(5642), −3/7967∗sqrt(15934), 4/257∗sqrt(257)],
[0, 0, 2/257∗sqrt(15934), 3/257∗sqrt(257)],
[3/91∗sqrt(91), 25/5642∗sqrt(5642), 21/15934∗sqrt(15934),−14/257∗sqrt(257)]]
:: char list list list

QR decomposition

value matrix-to-list-of-list (show-matrix-real (snd (QR-decomposition A)))

output
```
[[sqrt(91),  44/91*sqrt(91),      61/91*sqrt(91),        27/91*sqrt(91)],
 [       0, 2/91*sqrt(5642), 148/2821*sqrt(5642),   407/5642*sqrt(5642)],
 [       0,               0,     1/31*sqrt(15934), 327/15934*sqrt(15934)],
 [       0,               0,                    0,      39/257*sqrt(257)]] :: char list
list list
```

value A == (fst (QR-decomposition A)) ** (snd (QR-decomposition A))

output True :: prop

Application: Least Squares Approximation

- Let us consider a system $Ax = b$ without solution
- We can approximate the solution minimizing the error (least squares approximation). That is, compute $\hat{x}$ such that minimises $|| A\hat{x} - b ||$

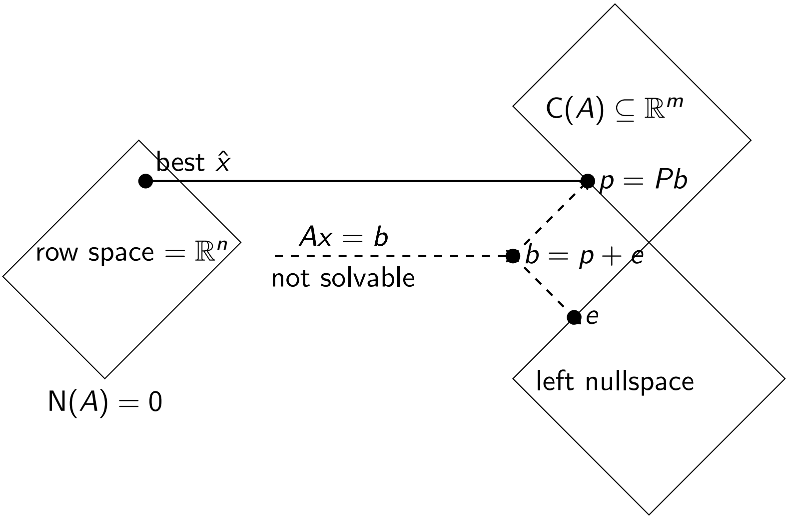

Figure : The projection $p = A\hat{x}$ is the closest point to b in $C(A)$

Application: Least Squares Approximation

- We have formalised that $\hat{x} = R^{-1}Q^T b$
- $\hat{x}$ can be computed symbolically, R^{-1} is computed by means of the Gauss-Jordan algorithm

Judgment day (using rationals for $\mathbb{R}$)

$H_6 x = b$

$$H_6 = \begin{pmatrix} 1 & 1/2 & 1/3 & 1/4 & 1/5 & 1/6 \\ 1/2 & 1/3 & 1/4 & 1/5 & 1/6 & 1/7 \\ 1/3 & 1/4 & 1/5 & 1/6 & 1/7 & 1/8 \\ 1/4 & 1/5 & 1/6 & 1/7 & 1/8 & 1/9 \\ 1/5 & 1/6 & 1/7 & 1/8 & 1/9 & 1/10 \\ 1/6 & 1/7 & 1/8 & 1/9 & 1/10 & 1/11 \end{pmatrix},\ b = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 5 \end{pmatrix}$$

Judgment day (using rationals for $\mathbb{R}$)

$H_6 x = b$

$$H_6 = \begin{pmatrix} 1 & 1/2 & 1/3 & 1/4 & 1/5 & 1/6 \\ 1/2 & 1/3 & 1/4 & 1/5 & 1/6 & 1/7 \\ 1/3 & 1/4 & 1/5 & 1/6 & 1/7 & 1/8 \\ 1/4 & 1/5 & 1/6 & 1/7 & 1/8 & 1/9 \\ 1/5 & 1/6 & 1/7 & 1/8 & 1/9 & 1/10 \\ 1/6 & 1/7 & 1/8 & 1/9 & 1/10 & 1/11 \end{pmatrix}, \; b = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 5 \end{pmatrix}$$

The least squares approximation

$\mathrm{x} = [-13824, 415170, -2907240, 7754040, -8724240, 3489948]$

Judgment day (using rationals for $\mathbb{R}$)

$H_6 x = b$

$$H_6 = \begin{pmatrix} 1 & 1/2 & 1/3 & 1/4 & 1/5 & 1/6 \\ 1/2 & 1/3 & 1/4 & 1/5 & 1/6 & 1/7 \\ 1/3 & 1/4 & 1/5 & 1/6 & 1/7 & 1/8 \\ 1/4 & 1/5 & 1/6 & 1/7 & 1/8 & 1/9 \\ 1/5 & 1/6 & 1/7 & 1/8 & 1/9 & 1/10 \\ 1/6 & 1/7 & 1/8 & 1/9 & 1/10 & 1/11 \end{pmatrix}, \; b = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 5 \end{pmatrix}$$

The least squares approximation

$\mathrm{x} = [-13824, 415170, -2907240, 7754040, -8724240, 3489948]$

Performance

- Poly/ML with optimisations: 0.013 s; without optimisations: 0.022 s.
- Mathematica®: 0.017 s.

Judgment Day (using iarrays and rationals for $\mathbb{R}$)

Notes

- Unfortunately, using fractions to represent (a subset of) reals requires a lot of intermediary arithmetic operations, affecting performance

Judgment Day (using iarrays and rationals for $\mathbb{R}$)

Notes

- Unfortunately, using fractions to represent (a subset of) reals requires a lot of intermediary arithmetic operations, affecting performance
- For small matrices, the verified program challenges Mathematica®

Judgment Day (using iarrays and rationals for $\mathbb{R}$)

Notes

- Unfortunately, using fractions to represent (a subset of) reals requires a lot of intermediary arithmetic operations, affecting performance
- For small matrices, the verified program challenges Mathematica®

Optimisations performed (and proved!)

- Avoid some repeated operations (defining the same variable twice)

Judgment Day (using iarrays and rationals for $\mathbb{R}$)

Notes

- Unfortunately, using fractions to represent (a subset of) reals requires a lot of intermediary arithmetic operations, affecting performance
- For small matrices, the verified program challenges Mathematica®

Optimisations performed (and proved!)

- Avoid some repeated operations (defining the same variable twice)
- Optimising computations over lists (avoiding unnecesary operations such as *remdups* over rows and columns indexes)

Judgment Day (using iarrays and rationals for $\mathbb{R}$)

Notes

- Unfortunately, using fractions to represent (a subset of) reals requires a lot of intermediary arithmetic operations, affecting performance
- For small matrices, the verified program challenges Mathematica®

Optimisations performed (and proved!)

- Avoid some repeated operations (defining the same variable twice)
- Optimising computations over lists (avoiding unnecesary operations such as *remdups* over rows and columns indexes)

J. Aransay and J. Divasón. A formalisation in HOL of the Fundamental Theorem of Linear Algebra and its application to the solution of the least squares problem. Journal of Automated Reasoning. 2016

Judgment Day (iarrays and floating-point numbers for $\mathbb{R}$)

Comparison of the solutions to $H_6 x = b$

- 1: Least squares approximation using rationals
- 2: *QR* approximation using floating-point numbers
- 3: Gauss-Jordan approximation using floating-point numbers

1 :	-13824	415170	-2907240	7754040	-8724240	3489948
2 :	-13824.0	415170.0001	-2907240.0	7754040.001	-8724240.001	3489948.0
3 :	-13808.6421	414731.7866	-2904277.468	7746340.301	-8715747.432	3486603.907

Judgment Day

Size (n)	**Poly/ML** (s.)
100	0.88
200	10.88
300	84.40
400	184.11

Table : Elapsed time (in seconds) to compute the QR decomposition of H_n with floating-point precision

Judgment Day

Size (n)	**Poly/ML** (s.)
100	0.88
200	10.88
300	84.40
400	184.11

Table : Elapsed time (in seconds) to compute the QR decomposition of H_n with floating-point precision

J. Aransay and J. Divasón. *QR Decomposition*. Archive of Formal Proofs. 2015

Conclusions

- Linear Algebra algorithms can be implemented in HMA (linked to mathematical results)
- Algorithms are executable inside of Isabelle
- Better performance can be obtained thanks to code generation in SML and Haskell
- The use of immutable arrays does not pose a drawback, even in comparison to imperative programming

Further Work

- Explore the possibilities of optimization of rational numbers operations in SML

Further Work

- Explore the possibilities of optimization of rational numbers operations in SML
- Improve the ease to produce proofs of refinements

Further Work

- Explore the possibilities of optimization of rational numbers operations in SML
- Improve the ease to produce proofs of refinements
- Explore floating-point numbers possibilities

Good Luck!!!

thank you!