Verified Computer Linear Algebra

Jesús Aransay, Jose Divasón

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

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

Gauss-Jordan

QR Decomposition

Conclusions
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)
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 *naïve* 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
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

```isar
typedef \( (\alpha, \beta) \) vec = UNIV :: ((\beta::finite) \Rightarrow \alpha) \) set
morphism vec--nth vec--lambda ..
```

- Type System vs Logic
Introduction

Gauss-Jordan

QR Decomposition

Conclusions
The Gauss-Jordan algorithm

nullspace \{ x \cdot Ax = 0 \}

\text{dim}(n - r)

\text{Col}(A) \quad \text{dim } r

\text{nullspace} \quad \text{column space} \quad \{ y \cdot y = Ax \}

A \in \mathbb{M}_{m,n}(F)

\text{Figure: Rank Nullity Theorem}
From theorems to algorithms

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

\[
A = \begin{pmatrix}
1 & -2 & 1 & -3 & 0 \\
3 & -6 & 2 & -7 & 0 \\
5 & -1 & 3 & 2 & 5 \\
0 & 7 & 4 & 5 & 1 \\
3 & -6 & 2 & -7 & 0
\end{pmatrix} \rightarrow
A = \begin{pmatrix}
1 & 0 & 0 & 0 & 3 \\
0 & 1 & 0 & 0 & 2 \\
0 & 0 & 1 & 0 & -2 \\
0 & 0 & 0 & 1 & -1 \\
0 & 0 & 0 & 0 & 0
\end{pmatrix}
\]
From theorems to algorithms

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

Gauss-Jordan example

\[
A = \begin{pmatrix}
1 & -2 & 1 & -3 & 0 \\
3 & -6 & 2 & -7 & 0 \\
5 & -1 & 3 & 2 & 5 \\
0 & 7 & 4 & 5 & 1 \\
3 & -6 & 2 & -7 & 0
\end{pmatrix}
\rightarrow
A = \begin{pmatrix}
1 & 0 & 0 & 0 & 3 \\
0 & 1 & 0 & 0 & 2 \\
0 & 0 & 1 & 0 & -2 \\
0 & 0 & 0 & 1 & -1 \\
0 & 0 & 0 & 0 & 0
\end{pmatrix}
\]

$\dim(C(A)) = 4$
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* $A :: \mathbb{R}^{\times \text{cols}} \Rightarrow \text{finite, wellorder}^{\times \text{rows}}$ and $P :: \mathbb{R}^{\times \text{rows}} \times \text{rows}$

*assumes* inv-\(P\): invertible $P$

*shows* col-rank $A = \text{col-rank} (P \times A)$
Relying on \textit{invariants}

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

Isabelle lemmas about elementary operations

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

\textbf{lemma} \texttt{det-mult-row}:
\texttt{shows det (mult-row A a k) = k \ast 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 $i \geq \text{rank } A$\}

**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 $A = \{\text{row } i \ (P\text{-Gauss-Jordan (transpose } A)) \ | \ i \ \text{to-nat } i \geq \text{rank } A\}$

**definition** basis-row-space $A = \{\text{row } i \ (\text{Gauss-Jordan } A) \ | i. \ \text{row } i \ (\text{Gauss-Jordan } A) \neq 0\}$

**definition** solve-system $A \ b =$

(\text{let } GJ = \text{Gauss-Jordan-PA } A \ \text{in } (\text{snd } GJ, (\text{fst } GJ) \ast v \ b))

**definition** solve $A \ b = (\text{if consistent } A \ b \ \text{then}$

Some (solve-consistent-rref (fst (solve-system $A \ b$)) (snd (solve-system $A \ b$)),

basis-null-space $A$)

else None)
Refinement

Abstract representation
Refinement

Abstract representation →
Refinement

Abstract representation \(\rightarrow\) Abstract definitions
Refinement

Abstract representation $\rightarrow$ Abstract definitions $\rightarrow$ Proof

A refinement has been carried out so that operations over the abstract type $\text{vec}$ can be executed.
Refinement

Abstract representation $\rightarrow$ Abstract definitions $\rightarrow$ Proof

Concrete representation
Refinement

Abstract representation → Abstract definitions → Proof

Projection

Concrete representation
Refinement

Abstract representation $\rightarrow$ Abstract definitions $\rightarrow$ Proof

Projection

Concrete representation
A refinement has been carried out so that operations over the abstract type \( \text{vec} \) can be executed.
A refinement has been carried out so that operations over the abstract type $\text{vec}$ can be executed.
A refinement has been carried out so that operations over the abstract type \( \text{vec} \) can be executed.
Refinement

A refinement has been carried out so that operations over the abstract type \textit{vec} can be executed

1. Mapping \textit{vec} to \textit{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 \textit{iarray} which represents immutable arrays
- \textit{iarray} is implemented in both SML (\textit{Vector structure}) and Haskell (\textit{IArray class})
- We have refined \textit{vec} elements and operations to \textit{iarray} ones (proving the corresponding morphisms)

Features of this refinement

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

Abstract representation \(\alpha^\text{cols}\) \(\hat{\alpha}\) \(\hat{\text{rows}}\) \(\Gauss\text{-Jordan}\) Isabelle

Projection \(\Downarrow\)

Concrete representation \(\alpha\) \(\text{iarray}\) \(\text{iarray}\) Gauss\_Jordan\_iarrays Isabelle,SML,Haskell

Abstract definitions \(\longrightarrow\) Proof

Code lemmas \(\Downarrow\)

Concrete definitions \(\longrightarrow\) Execution
## Serialisations

<table>
<thead>
<tr>
<th>Isabelle/HOL</th>
<th>SML</th>
<th>Haskell</th>
</tr>
</thead>
<tbody>
<tr>
<td>iarray</td>
<td>Vector.vector</td>
<td>IArray.Array</td>
</tr>
<tr>
<td>rat</td>
<td>IntInf.int / IntInf.int</td>
<td>Rational</td>
</tr>
<tr>
<td>real</td>
<td>Real.real</td>
<td>Double</td>
</tr>
<tr>
<td>bit</td>
<td>Bool.bool</td>
<td>Bool</td>
</tr>
</tbody>
</table>

**Table:** Type serialisations
<table>
<thead>
<tr>
<th>Size (n)</th>
<th>Poly/ML</th>
<th>MLton</th>
<th>GHC</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.04</td>
<td>0.05</td>
<td>0.36</td>
</tr>
<tr>
<td>200</td>
<td>0.25</td>
<td>0.32</td>
<td>2.25</td>
</tr>
<tr>
<td>400</td>
<td>2.01</td>
<td>2.35</td>
<td>17.17</td>
</tr>
<tr>
<td>800</td>
<td>15.96</td>
<td>18.57</td>
<td>131.73</td>
</tr>
<tr>
<td>1200</td>
<td>62.33</td>
<td>70.45</td>
<td>453.57</td>
</tr>
<tr>
<td>1600</td>
<td>139.70</td>
<td>152.41</td>
<td>1097.41</td>
</tr>
<tr>
<td>2000</td>
<td>284.28</td>
<td>287.44</td>
<td>2295.30</td>
</tr>
</tbody>
</table>

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

- Both in SML and GHC we have serialised $\mathbb{Z}_2$ to `bool`
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)
### Table: Elapsed time (in seconds) to compute the $rref$ of randomly generated $\mathbb{Q}^{n \times n}$ matrices.

<table>
<thead>
<tr>
<th>Size (n)</th>
<th>Poly/ML</th>
<th>Haskell</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>20</td>
<td>0.02</td>
<td>0.03</td>
</tr>
<tr>
<td>40</td>
<td>0.21</td>
<td>0.24</td>
</tr>
<tr>
<td>60</td>
<td>1.16</td>
<td>1.09</td>
</tr>
<tr>
<td>80</td>
<td>3.77</td>
<td>3.53</td>
</tr>
<tr>
<td>100</td>
<td>9.75</td>
<td>9.03</td>
</tr>
</tbody>
</table>
Judgment Day

Some relevant facts

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

- In SML we have serialised $\mathbb{Q}$ to quotients of \textit{IntInf.int}
- In GHC we have serialised $\mathbb{Q}$ to \textit{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 \textit{divmod} is done in Poly/ML by means of \textit{IntInf.quotrem})
### Judgment Day

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

<table>
<thead>
<tr>
<th>Size (n)</th>
<th>Poly/ML</th>
<th>Haskell</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.03</td>
<td>0.38</td>
</tr>
<tr>
<td>200</td>
<td>0.25</td>
<td>2.62</td>
</tr>
<tr>
<td>300</td>
<td>0.81</td>
<td>8.47</td>
</tr>
<tr>
<td>400</td>
<td>1.85</td>
<td>19.51</td>
</tr>
<tr>
<td>500</td>
<td>3.51</td>
<td>37.13</td>
</tr>
<tr>
<td>600</td>
<td>6.03</td>
<td>64.13</td>
</tr>
<tr>
<td>700</td>
<td>9.57</td>
<td>100.59</td>
</tr>
<tr>
<td>800</td>
<td>13.99</td>
<td>148.20</td>
</tr>
</tbody>
</table>
Some relevant facts

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

<table>
<thead>
<tr>
<th>Matrix sizes</th>
<th>C++ version</th>
<th>Verified version</th>
</tr>
</thead>
<tbody>
<tr>
<td>600 × 600</td>
<td>01.33s.</td>
<td>06.16s.</td>
</tr>
<tr>
<td>1000 × 1000</td>
<td>05.94s.</td>
<td>32.08s.</td>
</tr>
<tr>
<td>1200 × 1200</td>
<td>10.28s.</td>
<td>62.33s.</td>
</tr>
<tr>
<td>1400 × 1400</td>
<td>16.62s.</td>
<td>97.16s.</td>
</tr>
</tbody>
</table>

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 $\mathbb{R}^n$, $N(A) = C(A^T)^\perp$ 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 $\mathbb{R}^n$, $N(A) = C(A^T)^{\perp}$ that is, the nullspace is the orthogonal complement of the row space
- In $\mathbb{R}^m$, $N(A^T) = C(A)^{\perp}$, 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** \( A :: \text{real}^{\text{cols} \times \text{rows}} \)
  - **Shows** \( \text{null-space } A = \text{orthogonal-complement (row-space } A) \)
Second Part of the Fundamental Theorem of Linear Algebra

- **theorem** null-space-orthogonal-complement-row-space:
  - **fixes** $A :: \text{real}^\text{cols,rows}$
  - **shows** $\text{null-space } A = \text{orthogonal-complement (row-space } A)$

- **theorem** left-null-space-orthogonal-complement-col-space:
  - **fixes** $A :: \text{real}^\text{cols,rows}$
  - **shows** $\text{left-null-space } A = \text{orthogonal-complement (col-space } A)$
Second Part of the Fundamental Theorem of Linear Algebra

- **Theorem** null-space-orthogonal-complement-row-space:
  - **Fixes** $A :: \text{real}^{\text{cols} \times \text{rows}}$
  - **Shows** $\text{null-space } A = \text{orthogonal-complement (row-space } A)$

- **Theorem** left-null-space-orthogonal-complement-col-space:
  - **Fixes** $A :: \text{real}^{\text{cols} \times \text{rows}}$
  - **Shows** $\text{left-null-space } A = \text{orthogonal-complement (col-space } A)$

From Mathematical results to algorithms

The *Gram-Schmidt process* allows us to compute the mentioned orthogonal bases
**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. $Q = \text{Apply Gram-Schmidt to the columns of } A$, normalise the vectors
2. Compute $R$ as $R = Q^T A$
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.
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**  
\[
A :: \text{real}^{4 \times 4}
\]

**where**  
\[\begin{align*}
\text{foo} &= \text{list-of-list-to-matrix} \begin{bmatrix} 1,2,4,6, \\ 9,4,5,2, \\ 0,0,4,3, \\ 3,2,4,1 \end{bmatrix}, \\
\end{align*}\]

**value**  
\[
\text{matrix-to-list-of-list} \left( \text{show-matrix-real} \left( \text{fst} \left( \text{QR-decomposition} \ A \right) \right) \right)
\]

**output**
\[
\begin{bmatrix}
\frac{1}{91} \sqrt{91}, \frac{69}{5642} \sqrt{5642}, -\frac{9}{15934} \sqrt{15934}, \frac{6}{257} \sqrt{257} \\
\frac{9}{91} \sqrt{91}, -\frac{8}{2821} \sqrt{5642}, -\frac{3}{7967} \sqrt{15934}, \frac{4}{257} \sqrt{257} \\
0, 0, \frac{2}{257} \sqrt{15934}, \frac{3}{257} \sqrt{257} \\
\frac{3}{91} \sqrt{91}, \frac{25}{5642} \sqrt{5642}, \frac{21}{15934} \sqrt{15934}, -\frac{14}{257} \sqrt{257} \\
\end{bmatrix}:: \text{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
```

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

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

- We have formalised that $\hat{x} = R^{-1}Q^Tb$
- $\hat{x}$ can be computed symbolically, $R^{-1}$ is computed by means of the Gauss-Jordan algorithm
Judgment day (using rationals for \( \mathbb{R} \))

\[ H_6 x = b \]

\[
H_6 = \begin{pmatrix}
1 & 1/2 & 1/3 & 1/4 & 1/5 & 1/6 \\
1/2 & 1/3 & 1/4 & 1/5 & 1/6 & 1/7 \\
1/3 & 1/4 & 1/5 & 1/6 & 1/7 & 1/8 \\
1/4 & 1/5 & 1/6 & 1/7 & 1/8 & 1/9 \\
1/5 & 1/6 & 1/7 & 1/8 & 1/9 & 1/10 \\
1/6 & 1/7 & 1/8 & 1/9 & 1/10 & 1/11
\end{pmatrix}, \quad b = \begin{pmatrix}
1 \\
0 \\
0 \\
0 \\
0 \\
5
\end{pmatrix}
\]
Judgment day (using rationals for $\mathbb{R}$)

$$H_6 x = b$$

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

The least squares approximation

$$x = [-13824, 415170, -2907240, 7754040, -8724240, 3489948]$$
Judgment day (using rationals for $\mathbb{R}$)

$$H_6x = b$$

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

The least squares approximation

$$x = [-13824, 415170, -2907240, 7754040, -8724240, 3489948]$$

Performance

- Poly/ML with optimisations: 0.013 s; without optimisations: 0.022 s.
- Mathematica\textsuperscript{®}: 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 unnecessary 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 unnecessary operations such as \textit{remdups} over rows and columns indexes)

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

Comparison of the solutions to \( H_6x = b \)

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

\[
\begin{array}{cccccc}
1 &: -13824 & 415170 & -2907240 & 7754040 & -8724240 & 3489948 \\
2 &: -13824.0 & 415170.0001 & -2907240.0 & 7754040.001 & -8724240.001 & 3489948.0 \\
3 &: -13808.6421 & 414731.7866 & -2904277.468 & 7746340.301 & -8715747.432 & 3486603.907 \\
\end{array}
\]
Judgment Day

<table>
<thead>
<tr>
<th>Size (n)</th>
<th>Poly/ML (s.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.88</td>
</tr>
<tr>
<td>200</td>
<td>10.88</td>
</tr>
<tr>
<td>300</td>
<td>84.40</td>
</tr>
<tr>
<td>400</td>
<td>184.11</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Size ($n$)</th>
<th>Poly/ML (s.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.88</td>
</tr>
<tr>
<td>200</td>
<td>10.88</td>
</tr>
<tr>
<td>300</td>
<td>84.40</td>
</tr>
<tr>
<td>400</td>
<td>184.11</td>
</tr>
</tbody>
</table>

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

Introduction

Gauss-Jordan

QR Decomposition

Conclusions
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!