Gauss-Jordan Elimination for Matrices Represented as Functions

Tobias Nipkow

March 17, 2025

Abstract

This theory provides a compact formulation of Gauss-Jordan elimination for matrices represented as functions. Its distinctive feature is succinctness. It is not meant for large computations.

1 Gauss-Jordan elimination algorithm

theory *Gauss-Jordan-Elim-Fun*
imports
HOL−Combinatorics.Transposition
begin

Matrices are functions:

type-synonym $'a$ *matrix* = *nat* ⇒ *nat* ⇒ $'a$

In order to restrict to finite matrices, a matrix is usually combined with one or two natural numbers indicating the maximal row and column of the matrix.

Gauss-Jordan elimination is parameterized with a natural number n. It indicates that the matrix A has n rows and columns. In fact, A is the augmented matrix with $n+1$ columns. Column n is the "right-hand side", i.e. the constant vector b. The result is the unit matrix augmented with the solution in column n; see the correctness theorem below.

```
fun gauss-jordan :: ('a::field)matrix ⇒ nat ⇒ ('a)matrix option where
gauss-jordan A 0 = Some(A) |
gauss-jordan A (Suc m) =
 (case dropWhile (λi. A i m = 0) [0..<Suc m] of
   [] ⇒ None |
   p # - ⇒
    (let Ap' = (λj. A p j / A p m);
         A' = (λi. if i=p then Ap' else (λj. A i j − A i m ∗ Ap' j))
     in gauss-jordan (Fun.swap p m A') m))
```

Some auxiliary functions:

definition *solution* :: ('a::field)matrix ⇒ nat ⇒ (nat ⇒ 'a) ⇒ bool **where**
solution A n x = (∀ i<n. (∑ j=0..<n. A i j ∗ x j) = A i n)

definition *unit* :: ('a::field)matrix ⇒ nat ⇒ nat ⇒ bool **where**
unit A m n =
(∀ i j::nat. m≤j ⟶ j<n ⟶ A i j = (if i=j then 1 else 0))

lemma *solution-swap*:
assumes *p1* < *n p2* < *n*
shows *solution* (*Fun.swap p1 p2 A*) *n x* = *solution A n x* (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *solution-upd1*:
c ≠ *0* ⟹ *solution* (*A*(*p*:=(λj. *A p j* / *c*))) *n x* = *solution A n x*
⟨*proof*⟩

lemma *solution-upd-but1*: ⟦ *ap* = *A p*; ∀ i j. i≠p ⟶ *a i j* = *A i j*; p<n ⟧ ⟹
solution (λi. *if i=p then ap else* (λj. *a i j* − *c i* ∗ *ap j*)) *n x* =
solution A n x
⟨*proof*⟩

1.1 Correctness

The correctness proof:

lemma *gauss-jordan-lemma*: m≤n ⟹ *unit A m n* ⟹ *gauss-jordan A m* = *Some B* ⟹
unit B 0 n ∧ *solution A n* (λj. *B j n*)
⟨*proof*⟩

theorem *gauss-jordan-correct*:
gauss-jordan A n = *Some B* ⟹ *solution A n* (λj. *B j n*)
⟨*proof*⟩

definition *solution2* :: ('a::field)matrix ⇒ nat ⇒ nat ⇒ (nat ⇒ 'a) ⇒ bool
where *solution2 A m n x* = (∀ i<m. (∑ j=0..<m. A i j ∗ x j) = A i n)

definition *usolution A m n x* ⟷
solution2 A m n x ∧ (∀ y. *solution2 A m n y* ⟶ (∀ j<m. y j = x j))

lemma *non-null-if-pivot*:
assumes *usolution A m n x* **and** *q* < *m* **shows** ∃ p<m. *A p q* ≠ *0*
⟨*proof*⟩

lemma *lem1*:
fixes *f* :: 'a ⇒ 'b::field
shows (∑ x∈A. f x ∗ (a ∗ g x)) = a ∗ (∑ x∈A. f x ∗ g x)
⟨*proof*⟩

lemma *lem2*:
  fixes $f :: 'a \Rightarrow 'b::field$
  shows $(\sum x \in A.\ f\ x * (g\ x * a)) = a * (\sum x \in A.\ f\ x * g\ x)$
  ⟨*proof*⟩

1.2 Complete

lemma *gauss-jordan-complete*:
  $m \leq n \Longrightarrow usolution\ A\ m\ n\ x \Longrightarrow \exists B.\ gauss\text{-}jordan\ A\ m = Some\ B$
⟨*proof*⟩

Future work: extend the proof to matrix inversion.

hide-const (**open**) *unit*

end