

# Gauss-Jordan Elimination for Matrices Represented as Functions

Tobias Nipkow

August 7, 2022

## 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  $\Rightarrow$  nat  $\Rightarrow$  (nat  $\Rightarrow$  'a)  $\Rightarrow$  bool **where**  
*solution* A n x = ( $\forall i < n. (\sum_{j=0..<n.} A\ i\ j * x\ j) = A\ i\ n$ )

**definition** *unit* :: ('a::field)matrix  $\Rightarrow$  nat  $\Rightarrow$  nat  $\Rightarrow$  bool **where**  
*unit* A m n =  
( $\forall i\ j::nat. m \leq j \longrightarrow j < n \longrightarrow A\ i\ j = (\text{if } i=j \text{ then } 1 \text{ else } 0)$ )

**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**(cases p1=p2)

**case** True **thus** ?thesis **by** simp

**next**

**case** False

**show** ?thesis

**proof**

**assume** ?R **thus** ?L **using** *assms* False **by**(simp add: *solution-def* Fun.swap-def)

**next**

**assume** ?L

**show** ?R

**proof**(auto simp: *solution-def*)

**fix** i **assume** i < n

**show** ( $\sum_{j=0..<n.} A\ i\ j * x\ j$ ) = A i n

**proof** cases

**assume** i=p1

**with** <?L> *assms* False **show** ?thesis

**by**(fastforce simp add: *solution-def* Fun.swap-def)

**next**

**assume** i≠p1

**show** ?thesis

**proof** cases

**assume** i=p2

**with** <?L> *assms* False **show** ?thesis

**by**(fastforce simp add: *solution-def* Fun.swap-def)

**next**

**assume** i≠p2

**with** <i≠p1> <?L> <i < n> *assms* False **show** ?thesis

**by**(fastforce simp add: *solution-def* Fun.swap-def)

**qed**

**qed**

**qed**

**qed**

**qed**

**lemma** *solution-upd1*:

  c ≠ 0  $\implies$  *solution* (A(p:=( $\lambda j. A\ p\ j / c$ ))) n x = *solution* A n x

**apply**(cases p < n)

```

prefer 2
apply(simp add: solution-def)
apply(clarsimp simp add: solution-def)
apply rule
apply clarsimp
apply(case-tac i=p)
  apply (simp add: sum-divide-distrib[symmetric] eq-divide-eq field-simps)
  apply simp
apply (simp add: sum-divide-distrib[symmetric] eq-divide-eq field-simps)
done

```

```

lemma solution-upd-but1:  $\llbracket ap = A p; \forall i j. i \neq p \longrightarrow a i j = A i j; p < n \rrbracket \implies$ 
  solution  $(\lambda i. \text{if } i=p \text{ then } ap \text{ else } (\lambda j. a i j - c i * ap j)) n x =$ 
  solution  $A n x$ 
apply(clarsimp simp add: solution-def)
apply rule
prefer 2
  apply (simp add: field-simps sum-subtractf sum-distrib-left[symmetric])
  apply(clarsimp)
  apply(case-tac i=p)
    apply simp
  apply (auto simp add: field-simps sum-subtractf sum-distrib-left[symmetric] all-conj-distrib)
done

```

## 1.1 Correctness

The correctness proof:

```

lemma gauss-jordan-lemma:  $m \leq n \implies \text{unit } A m n \implies \text{gauss-jordan } A m = \text{Some } B \implies$ 
  unit  $B 0 n \wedge \text{solution } A n (\lambda j. B j n)$ 
proof(induct m arbitrary: A B)
  case 0
  { fix a and b c d :: 'a
    have (if a then b else c) * d = (if a then b*d else c*d) by simp
  } with 0 show ?case by(simp add: unit-def solution-def sum.If-cases)
next
  case (Suc m)
  let ?Ap' p =  $(\lambda j. A p j / A p m)$ 
  let ?A' p =  $(\lambda i. \text{if } i=p \text{ then } ?Ap' p \text{ else } (\lambda j. A i j - A i m * ?Ap' p j))$ 
  from  $\langle \text{gauss-jordan } A (Suc m) = \text{Some } B \rangle$ 
  obtain p ks where dropWhile  $(\lambda i. A i m = 0) [0..<Suc m] = p\#\text{ks}$  and
    rec:  $\text{gauss-jordan } (\text{Fun.swap } p m (\text{?A' } p)) m = \text{Some } B$ 
  by (auto split: list.splits)
  from this have p:  $p \leq m \wedge A p m \neq 0$ 
  apply(simp-all add: dropWhile-eq-Cons-conv del:upt-Suc)
  by (metis set-upt atLeast0AtMost atLeastLessThanSuc-atLeastAtMost atMost-iff
in-set-conv-decomp)
  have  $m \leq n \wedge m < n$  using  $\langle Suc m \leq n \rangle$  by arith+
  have unit  $(\text{Fun.swap } p m (\text{?A' } p)) m n$  using Suc.prem(2) p

```

**unfolding** *unit-def Fun.swap-def Suc-le-eq* **by** (*auto simp: le-less*)  
**from** *Suc.hyps[OF ‹m ≤ n› this rec] ‹m < n› p*  
**show** *?case*  
**by** (*simp only: solution-swap*) (*simp-all add: solution-swap solution-upd-but1*)  
**[where**  $A = A(p := ?Ap' p)$ **]** *solution-upd1*)  
**qed**

**theorem** *gauss-jordan-correct*:  
*gauss-jordan A n = Some B*  $\implies$  *solution A n* ( $\lambda j. B j n$ )  
**by**(*simp add:gauss-jordan-lemma[of n n] unit-def field-simps*)

**definition** *solution2* :: (*'a::field*)*matrix*  $\Rightarrow$  *nat*  $\Rightarrow$  *nat*  $\Rightarrow$  (*nat*  $\Rightarrow$  *'a*)  $\Rightarrow$  *bool*  
**where** *solution2 A m n x* = ( $\forall i < m. (\sum_{j=0..<m.} A i j * x j) = A i n$ )

**definition** *usolution A m n x*  $\longleftrightarrow$   
*solution2 A m n x*  $\wedge$  ( $\forall y. \text{solution2 } A m n y \longrightarrow (\forall j < m. y j = x j)$ )

**lemma** *non-null-if-pivot*:  
**assumes** *usolution A m n x* **and**  $q < m$  **shows**  $\exists p < m. A p q \neq 0$   
**proof**(*rule ccontr*)  
**assume**  $\neg(\exists p < m. A p q \neq 0)$   
**hence**  $1: \bigwedge p. p < m \implies A p q = 0$  **by** *simp*  
**{ fix** *y* **assume**  $2: \forall j. j \neq q \longrightarrow y j = x j$   
**{ fix** *i* **assume**  $i < m$   
**with** *assms(1)* **have**  $A i n = (\sum_{j=0..<m.} A i j * x j)$   
**by** (*auto simp: solution2-def usolution-def*)  
**with**  $1[OF \langle i < m \rangle]$   $2$   
**have**  $(\sum_{j=0..<m.} A i j * y j) = A i n$   
**by** (*auto intro!: sum.cong*)  
**}**  
**hence** *solution2 A m n y* **by**(*simp add: solution2-def*)  
**}**  
**hence** *solution2 A m n* ( $x(q:=0)$ ) **and** *solution2 A m n* ( $x(q:=1)$ ) **by** *auto*  
**with** *assms(1)* *zero-neq-one*  $\langle q < m \rangle$   
**show** *False*  
**by** (*simp add: usolution-def*)  
*(metis fun-upd-same zero-neq-one)*  
**qed**

**lemma** *lem1*:  
**fixes**  $f :: 'a \Rightarrow 'b::field$   
**shows**  $(\sum_{x \in A. f x * (a * g x)}) = a * (\sum_{x \in A. f x * g x)$   
**by** (*simp add: sum-distrib-left field-simps*)

**lemma** *lem2*:  
**fixes**  $f :: 'a \Rightarrow 'b::field$   
**shows**  $(\sum_{x \in A. f x * (g x * a)}) = a * (\sum_{x \in A. f x * g x)$   
**by** (*simp add: sum-distrib-left field-simps*)

## 1.2 Complete

**lemma** *gauss-jordan-complete*:

$m \leq n \implies \text{usolution } A \ m \ n \ x \implies \exists B. \text{ gauss-jordan } A \ m = \text{Some } B$

**proof**(*induction m arbitrary: A*)

**case** 0 **show** ?case **by** *simp*

**next**

**case** (*Suc m A*)

**from**  $\langle \text{Suc } m \leq n \rangle$  **have**  $m \leq n$  **and**  $m < \text{Suc } m$  **by** *arith+*

**from** *non-null-if-pivot[OF Suc.prem1(2)  $\langle m < \text{Suc } m \rangle$ ]*

**obtain**  $p'$  **where**  $p' < \text{Suc } m$  **and**  $A \ p' \ m \neq 0$  **by** *blast*

**hence**  $\text{dropWhile } (\lambda i. A \ i \ m = 0) \ [0..<\text{Suc } m] \neq []$

**by** (*simp add: atLeast0LessThan*) (*metis lessThan-iff linorder-neqE-nat not-less-eq*)

**then obtain**  $p \ xs$  **where** 1:  $\text{dropWhile } (\lambda i. A \ i \ m = 0) \ [0..<\text{Suc } m] = p \# \ xs$

**by** (*metis list.exhaust*)

**from** *this* **have**  $p \leq m$   $A \ p \ m \neq 0$

**by** (*simp-all add: dropWhile-eq-Cons-conv del: upt-Suc*)

(*metis set-upt atLeast0AtMost atLeastLessThanSuc-atLeastAtMost atMost-iff in-set-conv-decomp*)

**then have**  $p < \text{Suc } m$   $A \ p \ m \neq 0$

**by** *auto*

**let**  $?Ap' = (\lambda j. A \ p \ j / A \ p \ m)$

**let**  $?A' = (\lambda i. \text{if } i=p \text{ then } ?Ap' \text{ else } (\lambda j. A \ i \ j - A \ i \ m * ?Ap' \ j))$

**let**  $?A = \text{Fun.swap } p \ m \ ?A'$

**have**  $A$ : *solution2*  $A \ (\text{Suc } m) \ n \ x$  **using** *Suc.prem1(2)* **by**(*simp add: usolution-def*)

{ **fix**  $i$  **assume**  $le-m$ :  $p < \text{Suc } m$   $i < \text{Suc } m$   $A \ p \ m \neq 0$

**have**  $(\sum j = 0..<m. (A \ i \ j - A \ i \ m * A \ p \ j / A \ p \ m) * x \ j) =$

$(\sum j = 0..<\text{Suc } m. A \ i \ j * x \ j) - A \ i \ m * x \ m -$

$(\sum j = 0..<\text{Suc } m. A \ p \ j * x \ j) - A \ p \ m * x \ m) * A \ i \ m / A \ p \ m$

**by** (*simp add: field-simps sum-subtractf sum-divide-distrib sum-distrib-left*)

**also have**  $\dots = A \ i \ n - A \ p \ n * A \ i \ m / A \ p \ m$

**using**  $A \ le-m$

**by** (*simp add: solution2-def field-simps del: sum.op-ivl-Suc*)

**finally have**  $(\sum j = 0..<m. (A \ i \ j - A \ i \ m * A \ p \ j / A \ p \ m) * x \ j) =$   
 $A \ i \ n - A \ p \ n * A \ i \ m / A \ p \ m . }$

**then have** *solution2*  $?A \ m \ n \ x$  **using**  $p$

**by** (*auto simp add: solution2-def Fun.swap-def field-simps*)

**moreover**

{ **fix**  $y$  **assume**  $a$ : *solution2*  $?A \ m \ n \ y$

**let**  $?y = y(m := A \ p \ n / A \ p \ m - (\sum j = 0..<m. A \ p \ j * y \ j) / A \ p \ m)$

**have** *solution2*  $A \ (\text{Suc } m) \ n \ ?y$  **unfolding** *solution2-def*

**proof** *safe*

**fix**  $i$  **assume**  $i < \text{Suc } m$

**show**  $(\sum j=0..<\text{Suc } m. A \ i \ j * ?y \ j) = A \ i \ n$

**proof** (*cases*  $i = p$ )

**assume**  $i = p$  **with**  $p$  **show** *thesis* **by** (*simp add: field-simps*)

**next**

**assume**  $i \neq p$

```

show ?thesis
proof (cases i = m)
  assume i = m
  with p ⟨i ≠ p⟩ have p < m by simp
  with a[unfolded solution2-def, THEN spec, of p] p(2)
  have A p m * (A m m * A p n + A p m * (∑ j = 0..<m. y j * A m j))
= A p m * (A m n * A p m + A m m * (∑ j = 0..<m. y j * A p j))
  by (simp add: Fun.swap-def field-simps sum-subtractf lem1 lem2
sum-divide-distrib[symmetric]
split: if-splits)
  with ⟨A p m ≠ 0⟩ show ?thesis unfolding ⟨i = m⟩
  by simp (simp add: field-simps)
next
  assume i ≠ m
  then have i < m using ⟨i < Suc m⟩ by simp
  with a[unfolded solution2-def, THEN spec, of i] p(2)
  have A p m * (A i m * A p n + A p m * (∑ j = 0..<m. y j * A i j)) =
A p m * (A i n * A p m + A i m * (∑ j = 0..<m. y j * A p j))
  by (simp add: Fun.swap-def split: if-splits)
  (simp add: field-simps sum-subtractf lem1 lem2 sum-divide-distrib
[symmetric])
  with ⟨A p m ≠ 0⟩ show ?thesis
  by simp (simp add: field-simps)
qed
qed
qed
with ⟨usolution A (Suc m) n x⟩
have ∀ j < Suc m. ?y j = x j by (simp add: usolution-def)
hence ∀ j < m. y j = x j
  by simp (metis less-SucI nat-neq-iff)
} ultimately have usolution ?A m n x
  by (simp add: usolution-def)
note * = Suc.IH [OF ⟨m ≤ n⟩ this]
from 1 show ?case
  by auto (use * in blast)
qed

```

Future work: extend the proof to matrix inversion.

**hide-const** (open) unit

**end**