Automate convergence rate proof for gradient descent on quadratic functions

Yan Peng
Department of Computer Science
University of British Columbia
Vancouver, B.C. V6T 1Z4 Canada
yanpeng@cs.ubc.ca

Abstract

Iterative optimization algorithms are widely used in machine learning. Machine learning researchers derive the convergence rate manually to show better performance. The proofs are usually done with repeated proof strategies with minor changes. Recent proofs are even becoming tedious and hard-to-read. Given these observations, we believe machine proofs can largely automate the process thus can improve algorithm study efficiency. Furthermore, machine proofs may help researchers come up with better proof results. The prelimitary experiment in this work proposes a framework and shows the possibility of automating the convergence rate proofs using a theorem prover.

1 Introduction

Machine learning, emerging and develping as a power technique, has been successfully applied to a variety of poblems and applications. For many scenarios, a machine learning problem is modelled as an optimization problem of minimizing some loss function under certain constraints. This motivates great interest and huge effort into optimization algorithm research. In the simplest case, pure mathematical deduction can provide the closed-form solution to our problem. Under more complex cases, we won't be able to solve the problem exactly, but we can approximately approach the optimal solution by doing proper iterative updates. Gradient descent and etc. are good examples of this kind of method.

There are two things we generally care about in an iterative optimation algorithm. What kind of problems the optimization method fit to and how fast the optimization method will converge to the optimal solution. Based on these two criterion, one can produce a "complexity zoo" for modern optimization methods. Basically, for deterministic gradient method, we can achieve superlinear convergence rate when the target function is strongly-smooth and stronly-convex. For strongly-smooth but only convex problems, sublinear convergence rate can be achieved. Subgradient or stochastic gradient methods are generally supposed to be slower than deterministic ones. Recently, Mark Schmidt, Nicolas Le Roux, Francis Bach [2] have proposed a stochastic gradient method called SAG and proved linear convergence rate under assumption of a strongly-smooth and strongly-convex problem.

All these convergence rate results discussed above have been manually proven. More will come up as this area grows and evolves. We observe several properties about these proofs. First, they are very similar. They are similar in a way that they use common assumptions, they apply similar mathematical deductions, and they derive similar intermediate results. Second, they vary between methods. New methods proposed are always different from old ones. In order to prove the new methods' convergence rate, researchers have to bring up a new proof that takes into consideration of the new facts. Third, they scale as the proposed methods become more sophisticated. If one takes a look at [2]'s proof. This is an 18-page convergence proof based on a Lyapunov argument that involves derivations on a very huge formula. These properties motivates this project on automating convergence rate proofs. Since computers are extremely good at replicated, polymorphic and tedious work, why don't we give the proof job to a machine and simply let it run for it.

Automating the convergence rate proofs brings up many benefits when viewed from different aspects. As already discussed, it can potentially relieve researchers from doing tedious and repeated work. Second, human beings can feel frustrated reading a time-consuming and intellectual-challenging proof of tens of pages. Their trust on the proof may fall if they fail to finish reading it. Providing a machine validated proof, in this way, can largely establish the credability of one's proof. Further from that, automating existing proofs may even inspire new algorithms or new convergence rate results because machine proofs give us the freedom to tweak around and push things to the limit. This is something that can not be easily achieved when we are doing manual proof, because we have to reserve enough space for obvious correctness.

In this project, we build the foundation for automating convergence rate proofs on quadratic functions. We prove the linear convergence rate of gradient descent on general quadratic functions. Our experiment result shows the possibility to form existing convergence proof inside a theorem prover and the general structure can be easily applied and extended to similar proofs or even more complicated proofs. Not trivially, my work also add more interesting stuff to the theorem proving research world.

2 Related work

As far as I'm awared of, this is the first attempt applying automatic theorem proving to convergence rate proofs in machine learning area. But there does exists former work on applying theorem proving to general machine learning problems. Ruben et al. [5] have done research on the verification of synthesized Kalman filters using the theorem prover, ACL2[3]. Their work is applied to the NASA's spaceship trajectory prediction using syhthesized Kalman filters. They modeled the Kalman filter as a recursive procedure and verified that the outcome of this algorithm should be the best possible estimation. The Kalman filter verified in their work can be viewed as a generalization of least squares techinique. So there are similarities between our work and theirs. But the properties we are trying to prove are different and the aspects of linear algebra we are using are different.

Formally modelling a system then doing reasoning on the system by directly looking at the mathematics is not a novel approach. Tremendous work have been done this way in different domains. Among those, verification problems in analog and mixed-signal (AMS) design are similar to our work. This is because our work and AMS verification involves linear or nonlinear arithmetics, reasoning about inequalities and proofs on resursive functions using inductions. Fabian [10] uses a theorem prover called Isabelle to verify bounds on solutions to simple ODEs from a single initial condition. Their work on representing affine arithmetic fall into similar category of our work. But again, the kind of problems and properties are quite different.

A little bit further away from my work are works that applies machine learning techniques to help automate a theorem prover through learning lemmas from existing theorems. Jonathan et al. [8] use machine learning techniques to help a theorem prover find about potential useful lemmas for certain theorems. In this way, they speed up the proof process by narrowing down the search space.

My master's thesis will be on my recent work of integrating SMT solver[6] and theorem prover for AMS designs. In that work, I use ACL2 to prove global convergence of a digital phase-locked loop (PLL) without talking about the convergence rate.

3 Automate the proof

We start with the simplest convergence proof that we can think of. Future constructions and extensions can be done based on the structure implemented for this simplest problem. The proof come into mind is the linear convergence rate proof for gradient descent on a quadratic function. This should be a good enough example because it maintains nearly all basic linear algebra theorems need for similar proofs. Future variations can be built by replacing the quadratic function with another function or applying another iteration update formula.

3.1 The convergence rate proof

Suppose the we have below optimization problem with a general quadratic function:

$$\min_x f(x) = \frac{1}{2}x^T Ax - b^T x + c \tag{1}$$

given the assumption that $\mu I \preceq A \preceq LI$, x^* is the unique minimizer and $b = Ax^*$.

Suppose k is the number of iterations and x^k is the x value of kth iteration. x^0 is the initialization. For gradient method, we have the next iteration being $x^{k+1} = x^k - \alpha_k f'(x^k)$. Gradient method works based on the observation that the gradient direction is the steepest descending direction. With properly defined step size α_k, one can ensure the target function to be decreasing on each iteration. Gradient method has linear convergence rate. To see why this is true,

$$\begin{aligned} ||x^{k+1} - x^*|| &= ||x^k - \alpha_k f'(x^k) - x^*|| \\ &= ||(x^k - x^*) - \alpha_k(Ax^k - b)|| \\ &= ||(x^k - x^*) - \alpha_k(Ax^k - Ax^*)|| \\ &= ||(I - \alpha_k A)(x^k - x^*)|| \\ &\le ||I - \alpha_k A|| ||x^k - x^*|| \\ &\le \max\{|1 - \alpha_k L|, |1 - \alpha_k \mu|\} ||x^k - x^*|| \end{aligned} \tag{2}$$

If we choose α_k to be $\alpha_k = \frac{1}{L}$, then we have,

$$\begin{aligned} ||x^{k+1} - x^*|| &\le \left(1 - \frac{\mu}{L}\right) ||x^k - x^*|| \\ &\le \left(1 - \frac{\mu}{L}\right)^{k+1} ||x^0 - x^*|| \end{aligned} \tag{3}$$

This results illustrate how the x value on each gradient descent iteration converges to the optimal point x^* with a linear rate of $(1 - \frac{\mu}{L})$. The best linear convergence rate can be achieved with $\alpha_k = \frac{2}{\mu + L}$,

$$||x^{k+1} - x^*|| \le \left(\frac{L - \mu}{L + \mu}\right) ||x^k - x^*|| \tag{4}$$

3.2 Problem setup

We choose the theorem prover ACL2 [3] as our machine tool. ACL2 is a theorem prover with about 20 years of history. It has an actively-developing open-source community that develops libraries for various logic, mathematical and application problems. It is also a theorem prover that has been successfully applied to a number of industrial, system-level modeling and verification problems. My recent research work of verifying global convergence of a digital PLL is done using a combination of ACL2 and a SMT[6] solver.

In order to form the convergence rate proof in ACL2, we identify below problems and give corresponding solutions.

3.2.1 Matrix representation

We choose the basic representation of our proof to be matrices. There are several obvious benefits. First, matrices are more concise in format comparing to one single formula. Second, programmer can do less manual work expanding matrices into formulas when programming the proof. Third, matrices allow arbituary large dimensions. Therefore, it's more friendly to scaling.

We choose to use the ACL2 book *matrix*[4] as our basic data structure. In ACL2, a compiled library of functions and theorems that works as a functional integration is called a book. The *matrix* book uses the ACL2 *array2p* data structure as its foundation and extends from it the support for basic matrix operations and reasonings. *array2p* is in essence a two dimentional array represented by an association list (like a dictonary in Python or a map in C++, but with $O(n)$ search time) adding some meta information describing the matrix(e.g. dimentions). The keys are position indices (i.e. suppose we have matrix A, where each element is named as $a_{i,j}$, then the key will be (i, j)) and the values are $a_{i,j}$'s. This book provides reasoning on basic matrix operations, e.g. matrix addition, multiplication, scaler-matrix multiplication, subtraction, inversion and so on.

3.2.2 Linear algebra theorems

Existing reasoning ability of the matrix book is relatively weak comparing to the complexity of the kind of problem we want to prove. Thus, a very large portion of the basic linear algebra theorems should be developed in order to support our proof. This could be a very large amount of work and it can easily take days and weeks to code. To make efficient use of my project time, we use a technique called *skip-proofs* supported by ACL2 to solve this problem.

skip-proofs gives the programmer a way of postponing part of the proof in the middle of development. This is especially useful when some theorems which are believed to be true seems to take too much effort to prove. Adding a *skip-proofs* into the system of theorems basically makes the assumption that such theorem is true. Continued from that, the programmer can add new proofs assuming such theorem is proved. In this way, the programmer can focus more on the structure of the proof rather than struggling with a particular hard theorem.

However, using *skip-proofs* is introducing unsoundness into the system. To see why this is dangerous, think about a situation when we "skip-proved" a false argument. Following the *principle of explosion*, we know that any theorems inferred by that false argument will be proved regardless of their real truthness, because FALSE can imply anything. Thus in the long run, one still want to remove all the *skip-proofs*.

ACL2 is executable. Except for its reasoning ability, another great use of it is for modelling your system. In order to make sure that our *skip-proofs* are not trivially false arguments, we add tests for the *skip-proofs* we introduce. This way, we largely eliminated the possibility that we are stating something that's very wrong.

3.2.3 Matrix norms, eigenvalues and singular values

The approach to represent matrix norms, eigenvalues and singular values in ACL2 is not obvious. Here, we are talking about 2-norms when we talk about vector norms and spectural norms when we talk about matrix norms.

For the easy case of vector norms, basically, we can define it as $||x||^2 = x^T x$. Matrix norms are much harder since it doesn't have a closed-form representation (if we don't have the definition for eigenvalues or singular values). So we must define it using the set of properties that uniquely define what a matrix norm should be. We use the following definition:

$$||A|| = \sup\left\{\frac{||Ax||}{||x||} : x \in K^n, x \neq 0\right\} \tag{5}$$

where K is the field of real and complex numbers. This can be further expanded into,

$$\begin{aligned} &\forall x, ||A|| \cdot ||x|| \leq ||Ax|| \\ &\exists x' \neq 0, ||A|| \cdot ||x'|| = ||Ax'|| \end{aligned} \tag{6}$$

These two theorems constitute what the norm of a matrix A should be. Notice here we require ACL2 to have the ability of logic reasoning with quantifiers. The technique we are using is a method supported by ACL2 called skolemization.

Skolemization uses a Skolem function to remove existential quantifiers from a predicate. The Skolem function is a magical function that can produce one of such instance, given relavent variables that are universally quantified. Below is an example illustrating a Skolem function, suppose we have the predicate: "Every philosopher writes at least one book"[7],

$$\begin{aligned} &\forall x[Philo(x) \rightarrow \exists y[Book(y) \wedge Write(x, y)]] \\ &\forall x[\neg Philo(x) \vee \exists y[Book(y) \wedge Write(x, y)]] \text{ (Eliminate implication)} \end{aligned} \tag{7}$$

Now suppose we have a function f that takes x as input and magically return a y. This function is defined that if this is a true statement, then for each philosopher x, $f(x)$ returns one of the books y (which can also be called a witness) that the philosopher writes. Thus the statement becomes,

$$\forall x[\neg Philo(x) \vee [Book(f(x)) \wedge Write(x, f(x))]] \tag{8}$$

The syntax of ACL2 is quantifier free and every formula is assumed to be universally quantified over all free variables in the formula. But it can support first-order quantification through skolemization to remove all existential quantifiers. When introducing a skolem function into ACL2. ACL2 basically adds two theorems. One says if there exists a witness to the theorem, then the theorem is true. The other one says if the theorem is true, there must exist a witness to the theorem. Applying this technique, we are able to axiomize our matrix norm definition as stated above.

We apply similar approaches to eigenvalues and singular values. The definition for eigenvalues comes from the eigenvalue decomposition theorem,

$$M = Q\Lambda Q^{-1} \tag{9}$$

where M is a square matrix. Λ is a diagonal matrix whose diagonals are the eigenvalues. Q is a square matrix whose columns are the corresponding eigenvectors.

This isn't quite true because not all matrices are diagonalizable. But ACL2 is still sound by generating axioms that says "If there is a choice of values for the existentially quantified variables that satisfies the constrains on the *exists* argument, then the Skolem function returns an example". If there's no such value, then the function can return anything. So, in order to use the skolemization, one needs to prove that the eigenvalues exist. A *skip-proofs* is introduced to solve the problem. This shouldn't hurt the soundness of ACL2 itself. However, in the long run, we still want to provide an exact definition for eigenvalues. Future work will discuss this issue.

Seemingly more complex than eigenvalues, singular values are actually more well-defined under all possible matrices. We take the singular value decomposition theorem,

$$M = U\Sigma V^* \tag{10}$$

where M is a m-by-n matrix whose entries are real numebrs or complex numbers. U and V satisfies $U^*U = I$ and $V^*V = I$, where * stands for conjugate transpose. Σ is a m-by-n diagonal matrix with non-negative real numbers on the diagonal. The non-negative real numbers are called singular values.

3.3 Detailed implementation

As discussed in last subsection, the implementation is composed of three parts. One book of theorems for basic linear algebra function definitions and theorems. This book is mostly *skip-proofs*. Future work will talk about proving all them. One file is for testing *skip-proofs* in the basic linear algebra theorem book so that we don't have trivially wrong theorems. The third book of theorems are the main lemmas and theorems that form the linear convergence rate proof from Section 3.1. We'll discuss each of them.

3.3.1 Linear algebra book

The basic linear algebra book consists a list of function definitions and theorems about linear algebra. Table 1 and Table 2 show what they are. (Note: no need to read the table if not interested to see what functions or theorems there are)

3.3.2 Test file

The tests are done in this manner. Suppose we have a theorem statement $p(x, y, z) \to q(x, y, z)$ called "simple-thm" as illustrated in Program 1. We decompose this theorem statement into a corresponding function and a theorem defined using this function as in Program 2.

This code is using LISP's prefix syntax. *defun* defines a function that takes a name and a list of arguments. *defthm* defines a theorem that ACL2 has to prove. By dividing the theorem into a executable function and a theorem statement, we are able to do tests on the functions in another

Table 1: basic linear algebra functions

FUNCTIONS	DEFINITION
m-is-real	check if a matrix is real
m-is-nneg-real	check if a matrix is non-negative real matrix
m-is-square	check if a matrix is square
m-is-symmetric	check if a matrix is symmetric
m-is-orthogonal	check if a matrix is orthogonal
m-is-vector	check if a matrix is a vector
m-is-diag	check if a matrix is diagonal
m-diag	given a matrix, return it's diagonal vector
m-can-be-added	check if two matrices can be added
m-can-be-multiplied	check if two matrices can be multiplied
m-rs	check if a matrix is real symmetric
m-pd	define a matrix to be positive definite
m-rspd	define a matrix to be real symmetric and positive definite
m-psd	define a matrix to be positive semi-definite
m-rspsd	define a matrix to be real symmetric and positive semi-definite
m-prec	define $\prec$ relation
m-preceq	define $\preceq$ relation
m-eig	define eigenvalue of a matrix
m-singular	define singular value of a matrix
v-2norm	define 2-norm of a vector
m-normp	define a matrix norm
multi-max	return the maximum value of a matrix

Listing 1: Simple theorem

```
(defthm simple-thm
  (implies (p x y z)
           (q x y z)))
```

test file. A typicle test looks like Program 3, The code basically assigns $X = [4,\ 3]$, $Y = [1,\ 2]$ and $Z = [5,\ 10]$ and tests "simple-thm" with these assignment. "(:HEADER ...)" are the meta information about the matrix. Due to project time limitation, we didn't have time to test every *skip-proofs*. But this task should be easy to do.

3.3.3 Convergence rate proof

The convergence rate proof is composed of two parts. First, we define 4 functions for this problem: the target function f, its gradient function *f-p*, its Hessian function *f-pp* and the iterative update function *inc-x*. Notice we choose to use the gradient function and the Hessian function defined as axioms without proving them. A more strict approach would be to prove that the defined functions are the gradient and Hessian of the quadratic function using the definition of a gradient and a Hessian.

Then the main body of the proof is composed of two main parts. One part does rewriting proofs to prove

$$||x^{k+1} - x^*|| = ||x^k - \alpha_k f'(x^k) - x^*|| = ||(I - \alpha_k A)(x^k - x^*)|| \tag{11}$$

For a given theorem, rewriting replace a current subterm with a new term by applying a rewrite or definition rule. This is based on a fundamental logic concept called substitution. E.g. $p(x, y, z) \wedge q(x, y, z)$ is a substitution instance of $S \wedge T$ by replacing S with $p(x, y, z)$ and T with $q(x, y, z)$. Rewriting rules are existing equivalence relations in the proof system. Doing substitutions with these rewriting rules let us simplify our theorems. A typical theorem prover will get lost in a large set of rewriting rules, so user provided hints are needed once in a while. Hints are written by the programmer, specifying which set of rewriting rules are really needed by this theorem.

Table 2: basic linear algebra theorems

THEOREMS	MEANING
m-scale-rs	real symmetric matrix multiplied by a real scaler is still a real symmetric matrix
m-add-rs	real symmetric matrix add a real symmetric matrix is still a real symmetric matrix
m-preceq-scale	preceq is preserved when multiplied by a positive real
2norm-congruence	if two vectors are equal, their 2-norms will be equal
v-norm->=-0	vector norms are non-negative
all-matrix-has-m-norm	all matrices have matrix norm
m-norm->=-0	matrix norms are non-negative
m-norm-is-max-sv	matrix norm using a 2-norm is the maximum sigular value
m-rs-sv-=-|eig|	If a matrix is real symmetric, then its singular values are the absolate values of its eigenvalues
M-Norm-unique	Matrix norm is unique
challenging-proof	An important theorem for the inequality in the proof
m-=-transitivity	matrix equal is transitive
left-s-*-distributive	scaler multiplication is distributive
+-associative-4elem-1	trivial matrix linear algebra
+-associative-4elem-2	trivial matrix linear algebra
extract-common-vector	trivial matrix linear algebra
s-*-associative	scaler multiplication is associative
subtraction-congruence	If two sets of matrices are equal, then their corresponding difference is also the same
m-*-distributive-over-m-−	matrix multiplication can be distributed over matrix subtraction
alist2p-m-+-alist2p	the difference of two matrices is also a matrix
<=-square	if $0 \leq m \leq n$, then $m^2 \leq n^2$
<=-*	if $k \geq 0$ and $m \leq n$, then $mk \leq nk$

Listing 2: Decompose simple theorem

```
(defun f-simple-thm (x y z)
  (implies (p x y z)
           (q x y z)))
(defthm simple-thm
  (f-simple-thm x y z))
```

The other half of the proof focuses on proving the inequality,

$$\begin{aligned} ||(I-\alpha_k A)(x^k - x^*)|| &\leq ||I-\alpha_k A|| ||x^k - x^*|| \\ &\leq \max\{|1-\alpha_k L|, |1-\alpha_k \mu|\} ||x^k - x^*|| \end{aligned} \tag{12}$$

This part is much harder because of the more complicated reasoning with matrix norms, eigenvalues and singular values. Because of time limitation of this project, we *skip-proof*ed one key lemma needed for this part of proof. This let us work on the rest of the proof to fullfil the project. Future work should be done to resolve the *skip-proofs*.

4 Results and analysis

4.1 Some statistics

Table 3 shows some statistics about this proof.

Listing 3: A test

```
(let ((X '(((0 . 0) . 4)
            ((0 . 1) . 3)
                (:HEADER :DIMENSIONS (1 2)
                         :MAXIMUM-LENGTH 10
                         :DEFAULT 0
                         :NAME X)))
      (Y '(((0 . 0) . 1)
            ((0 . 1) . 2)
            (:HEADER :DIMENSIONS (1 2)
                     :MAXIMUM-LENGTH 10
                     :DEFAULT 0
                     :NAME Y)))
      (Z '(((0 . 0) . 5)
            ((0 . 1) . 10)
            (:HEADER :DIMENSIONS (1 2)
                     :MAXIMUM-LENGTH 10
                     :DEFAULT 0
                     :NAME Z))))
 (fmx "m-=-transitivity is tested to be ~x0"
      (equal (f-simple-thm X Y Z) t)))
```

Table 3: Statistics of this proof

feature	number
manual proof length	21
# of funcs	43
# of thms	64
# of skip-proofs	21
LOC	1403
runtime	≈ 20min

We can get several conclusions from above numbers. First, the machine proof is 50 times longer than the manual proof. The common sense is that usually the machine proof will be 10-20 times longer than the manual proof. This is due to the reason that human reasoning are always making assumptions that they believed to be true. Humans also jump in deductions because they can think many steps between each line of proof. Machine needs more guidance to form the proof. Our proof seems a little bit longer due to the reason that we have to implement a book of basic linear algebra theorems. This portion of code is reusable for future proofs. Another observation is that the number of *skip-proofs* comparing to the total number of theorems is still low. This is a good news for us. The number of functions is relatively high, which might not be a good sign because some of them are introducing axioms into the system. Runtime is reasonable, but still larger proofs are needed to see how the runtime scales.

4.2 Scalability analysis

Scalability is one of the main issues we keep in mind when implementing this framework. There are several good points about this project that ensures scalability.

First, matrix representation gives us the space to scale our method into arbituarily large problems. Second, seperating basic linear algebra theorems from the main convergence rate proof give us the freedom to build new proofs based on existing linear algebra books. Third, the theorems are implemented in a way that if the new proof maintains similar structure to this problem but with new constraints, one can easily add the new constraints to build a new proof.

5 Conclusion and future work

In this paper, we summarized our project work on automating the convergence rate proof. We build the initial foudation and have proved a very simple example of gradient descent on quadratic functions. This work shows the possibility of automating convergence rate proofs of general iterative optimization algorithms. However, this work is still prelimitary. Lots of future work awaits to be done in order to achieve reasonable automation.

First, we need to remove *skip-proofs* from our linear algebra book. This can be a huge amount of work. For the definition of matrix norms, eigenvalues and singular values using skolemization. We either need to prove the existence of such norms, eigenvalues and singular values by actually providing them; or we have to circumvent directly providing them through other mathematical techniques. E.g. using secant method to prove the existence of a root for the characteristic polynomial.

Second, we still need to try proving a much larger proof to see what new problems might come up. It can be that the larger proof already becomes too tedius that we have to combine a more efficient solver, e.g. a SMT solver, to achieve the proof.

Third, all work done here are following below procodure. We first figure out a proof that's detailed enough for the machine based on the machine learning researcher's manual proof. Then the programmer code each theorem into the prover and see if the prover agrees. This process is still far from full automation. Future work should emphasize more on automation, e.g. how to let the machine generate theorems that are needed. This is possible given there's only a limited set of proof techniques and the proof structure is quite similar between different proofs.

Fourth, although it's our belief that machine proofs can give us lots of benefits, the simple example shown here hasn't quite give us those benefits. Future work need to be done to see how machine proofs can save time, effort and improve the human reasoning results.

Acknowledgments

Thank professor Mark Schimidt for holding a wonderful course this term. Thank the ACL2 community for immediate reply and help. Thank my supervisor, professor Mark Greenstreet, for insightful guidance and endless patience.

References

[1] Nicolas Le Roux, Mark Schmidt, Francis Bach. A Stochastic Gradient Method with an Exponential Convergence Rate for Finite Training Sets. *NIPS'12 - 26 th Annual Conference on Neural Information Processing Systems (2012), Dec 2011, Lake Tahoe, United States.* <hal- 00674995v4>

[2] Mark Schmidt, Nicolas Le Roux, Francis Bach. Minimizing Finite Sums with the Stochastic Average Gradient. 2013. <hal-00860051>

[3] Matt Kaufmann, J Moore. An Industrial Strength Theorem Prover for a Logic Based on Common Lisp. *IEEE Transactions on Software Engineering 23, no. 4, April 1997, 203–213.*

[4] Ruben Gamboa, John Cowles, Jeff Van Baalen. Using ACL2 Arrays to Formalize Matrix Algebra. *ACL2 workshop 2003, July 2003, Boulder Colorado, USA*.

[5] Ruben Gamboa, John Cowles, Jeff Van Baalen. On the Verification of Synthesized Kalman Filters. *ACL2 workshop 2003, July 2003, Boulder Colorado, USA*

[6] Leonardo de Moura and Nikolaj Bjrner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 337-340. Springer, 2008.

[7] Torsten Hahmann. Skolemization, Most General Unifiers, First-Order Resolution. *CSC384, University of Toronto. http://www.cs.toronto.edu/ sheila/384/w11/Lectures/csc384w11-KR-tutorial.pdf*, 2011.

[8] Jonathan Heras, Ekaterina Komendantskaya. ACL2(ml): Machine-Learning for ACL2. *ACL2 workshop 2014, as an ITP-affiliated workshop of FLOC(part of Vienna Summer of Logic), July 2014, Vienna, Austria*.