Differentiating through Conjugate Gradient

Bruce Christianson
b.christianson@herts.ac.uk
University of Hertfordshire, Hatfield, England

December 2017

Abstract

We show that, although the Conjugate Gradient (CG) Algorithm has a singularity at the solution, it is possible to differentiate forward through the algorithm automatically by re-declaring all the variables as truncated Taylor series, the type of active variable widely used in Automatic Differentiation (AD) tools such as ADOL-C. If exact arithmetic is used, this approach gives a complete sequence of correct directional derivatives of the solution, to arbitrary order, in a single cycle of at most n iterations, where n is the number of dimensions. In the inexact case the approach emphasizes the need for a means by which the programmer can communicate certain conditions involving derivative values directly to an AD tool.

1 Truncated Taylor CG

It is generally supposed to be problematic to differentiate naively through singularities[1]. The Conjugate Gradient (CG) Algorithm [1] has a singularity at the solution[2], in the sense that an attempt to run the algorithm starting at the solution will produce a divide-by-zero error.

Although practitioners sometimes discuss (in conversation) occasions when their CG code did, or didn't, converge to correct derivative values when it was forward-differentiated, there seems to be no theoretical analysis of this problem in the published literature. Gratton et al [2] consider first derivatives with respect to b of the successive CG approximations x_k to the solution x_* of $Ax = b$, but do not take account of the effect of convergence of the underlying problem on the derivatives, and do not consider derivatives of x_k with respect to A.

We show that, although the CG Algorithm has a singularity at the solution, it is nevertheless possible to differentiate through the algorithm automatically, even in case the algorithm is started at the solution itself. This allows us to

[1] For the case of the Euclidean norm at the origin see [3] p. 357 Table 14.9 and p. 363 Exercise 14.1.

[2] Unlike Newton algorithms, for instance, which do not.

obtain correct directional derivatives of the solution, to arbitrary order r, in a single cycle of at most n iterations, where n is the number of dimensions. Our approach involves re-declaring the active variables as truncated Taylor series, as used in ADOL-C [4] and elsewhere, and making adjustments (which we specify below) to the method of calculating certain coefficients, and to the stopping criteria.

Suppose that we wish to use CG to solve the linear equations $Ax = b$ for x at $t = 0$, where A and b are smooth non-linear functions of some variable t, and at the same time to compute directional derivatives $\dot{x}, \ddot{x}$ with respect to t, also at $t = 0$. These derivatives satisfy

$$A\dot{x} + \dot{A}x = \dot{b} \qquad\qquad A\ddot{x} + 2\dot{A}\dot{x} + \ddot{A}x = \ddot{b}$$

and so on for higher order derivatives. The trick is to re-purpose a conventional implementation of CG. For example, we shall denote the value of b at $t = 0$ by $b^{(0)}$ (simply adding a notational superscript) and write

$$b^{(1)} = \dot{b}, \qquad b^{(2)} = \frac{1}{2!}\ddot{b}, \qquad \ldots \qquad b^{(r)} = \frac{1}{r!}\overset{r\times\bullet}{b}$$

to construct the truncated Taylor vector

$$b = b^{(0)} + b^{(1)}t + b^{(2)}t^2 + \ldots + b^{(r)}t^r$$

Similarly for A. Now solving the truncated Taylor equation $Ax = b$ for x will give the solution $x^{(0)}$ to the original equation at $t = 0$ along with the correct derivatives

$$\dot{x} = x^{(1)}, \qquad \ddot{x} = (2!)x^{(2)}, \qquad \ldots \qquad \frac{d^r x}{dt^r} = (r!)x^{(r)}.$$

We refer to a truncated Taylor series such as α whose elements $\alpha^{(k)}$ are scalars as a *taylor*. In practice any implementation must use Taylor series truncated to some finite order. Although the analysis in this section and the next also applies to untruncated (infinite) Taylor series (subject to some care with radius of convergence arguments), the construction is designed to allow all the taylor values calculated during the course of the algorithm to be computed only to the *same* fixed order r, with a small number of exceptions[3]. There are some occasions in the proofs where, in order to establish a particular identity, we must *imagine* that we have calculated elements of higher order[4] than r, but these values are not used by the algorithm itself. We write $\alpha = 0$ to mean that $\alpha^{(k)} = 0$ for all k; for a truncated taylor of order r, $\alpha^{(k)} = 0$ for $k > r$ by convention.

A taylor vector such as b can be thought of as a truncated Taylor series whose elements $b^{(k)}$ are vectors, although it may in practice be implemented

[3]These exceptions are confined to certain intermediate quantities involved in the calculation of the taylor values designated α_i and β_i; these exceptions require calculation to order less than $2r$ in all cases, as we shall see.

[4]But even in these cases, we need never consider elements of higher order than $4r$.

as a vector whose components are taylors. Similarly A can be thought of as a truncated Taylor series whose entries are the matrices $A^{(k)}$, but implemented as a matrix composed of taylors.

If u, v are taylor vectors then we write $c = u \cdot v$ to denote the taylor with

$$c^{(k)} = \sum_{\ell=0}^{k} u^{(\ell)} \cdot v^{(k-\ell)},$$

u^2 denotes $u \cdot u$, and so on for all the usual vector and matrix linear algebra notations. The usual algebraic identities lift to taylors, so for example $(\alpha u) \cdot v = \alpha(u \cdot v)$. For further information on calculating taylor values see [3], Section 3.2: as an example, if $\alpha^{(0)} \neq 0$ and $\gamma = 1/\alpha$ then the elements of γ can be recursively calculated as

$$\gamma^{(0)} = \frac{1}{\alpha^{(0)}}; \qquad \gamma^{(k)} = \frac{-1}{\alpha^{(0)}} \sum_{\ell=0}^{k-1} \gamma^{(\ell)} \alpha^{(k-\ell)}$$

Truncated Taylor Conjugate Gradient Algorithm (TTCG)

As usual we assume $A^{(0)}$ to be symmetric and positive definite. The TTCG algorithm to solve $Ax = b$ looks just like the conventional CG algorithm, but the variables $A, b, x_i, g_i, p_i, \alpha_i, \beta_i$, etc. are taylors.

start:
  set $i := 0$
  let x_0 be an initial approximation to x
  $g_0 = Ax_0 - b$
  $p_1 = -g_0$
loop:
  $i := i + 1$
  choose α_i to solve $\alpha_i(p_i^T A p_i) = g_{i-1}^2$
  $x_i = x_{i-1} + \alpha_i p_i$
  $g_i = Ax_i - b = g_{i-1} + \alpha_i A p_i$
if $g_i = 0$ then terminate
  choose β_i to solve $\beta_i g_{i-1}^2 = g_i^2$
  $p_{i+1} = \beta_i p_i - g_i$
go to loop

We prove various properties of this algorithm formally in the next section, but give here a brief overview of our route to the main result of this paper. We already know from the conventional (order zero) case of CG that the truncated Taylor CG algorithm produces (if arithmetic is exact) an orthogonal sequence $g_i^{(0)}$. In a space of finite dimension n we therefore must have for some i_0 that

$g_{i_0}^{(0)} = 0$. However truncated Taylor CG need not terminate at this point[5] as the fact that $g_{i_0}^{(0)} = 0$ does not imply that $g = 0$ to order r. Continuing on with the algorithm from this point produces a further sequence of g_i with $g_i^{(0)} = 0$.

For each i, define $g^{(k_i)}$ to be the first non-zero element of g_i if there is one, so $g_i^{(k_i)} \neq 0$ and $g_i^{(\ell)} = 0$ for all $\ell : 0 \leq \ell < k_i \leq r$, and define $g_i^{(*)} = g^{(k_i)}$ in this case; define $g_i^{(*)} = 0$ iff $g_i = 0$, in which case k_i is undefined.

We shall show in section 2 that, provided we use exact arithmetic, the $g_i^{(*)}$ form an orthogonal sequence, with k_i monotone non-decreasing with i, and so for some $i_* \leq n$ we must have $g_{i_*} = 0$. At this point TTCG will terminate and give correct values for $x, \dot{x}, \ddot{x}$ and so on to order r. In section 3 we consider briefly some of the consequences of using inexact arithmetic, particularly on re-formulating the stopping criterion and the adjustment of k_i.

We conclude this section by remarking on the calculation of α_i and β_i. As soon as $k_{i-1} > 0$, these calculations involve taking the quotient of two taylors which are both divisible by a power of t. Although we only need to calculate α_i, β_i themselves as far as the element of order $r - k_{i-1}$, the cancellation of t requires[6] us to compute the intermediate elements $g_{i-1}^2, g_i^2, p_i A p_i$ to order $r + k_{i-1}$. A simple implementation is to store certain truncated taylor vectors in a "normalized form" and re-normalize automatically when we operate on them: for example we may represent $g_i = v t^{k_i}$ explicitly as the pair (k_i, v), where $v^{(0)} \neq 0$ unless $g_i = 0$, and similarly for p_i.

2 Termination and Convergence

Throughout this section, we shall assume that the arithmetic used is exact.

Definition 1 Let v be a taylor (scalar or vector). For $k \geq 0$ we say that t^k divides v, and write $t^k | v$ iff $v^{(\ell)} = 0$ for all $\ell : 0 \leq \ell < k$. For $v \neq 0$ we define the *order* of v to be the largest value of k such that $t^k | v$. If $v^{(0)} \neq 0$ then the order of v is zero; $1 = t^0$ divides every v because in this case the quantification over ℓ is empty.

Definition 2 For each i, we define k_i as the order of g_i.

Theorem 2.1 At each stage $i \geq 1$, α_i and β_i are well defined with $\alpha_i^{(0)} \neq 0$; $k_i \geq k_{i-1}$; p_{i+1} is of order k_i; $g_{i-1} \cdot g_i = 0$; $p_{i+1}^T A p_i = 0$; and $g_i \cdot p_i = 0$.

[5] For example, set $x_0 = 0$ and suppose that $b^{(0)}$ is an eigenvector of $A^{(0)}$ but that $b^{(1)}$ is not. Terminating the algorithm as soon as $g_i^{(0)} = 0$ may give wildly incorrect values for the derivatives of x.

[6] For example if $k_i = r$ then naive computation of α_i using taylors of order r will suffer from an underflow when computing g_i^2, which has order $2r$, even though $\alpha_i^{(0)} \neq 0$.

Proof The proof is by induction: for $i \geq 2$ we suppose the case for $i-1$ and deduce it for i. We first show that α_i is well defined and not divisible by t. The order of p_i is k_{i-1} by the induction hypothesis, the order of g_{i-1} is k_{i-1} by definition. Since $A^{(0)}$ is positive definite, we have that $p_i^T A p_i$ and g_{i-1}^2 both have order exactly $2k_{i-1}$, whence α_i is a well defined taylor not divisible by t.

Hence $t^{k_{i-1}}$ divides $g_i = g_{i-1} + \alpha_i A p_i$ whence $k_i \geq k_{i-1}$. So let $k_i = k_{i-1} + \delta \geq k_{i-1}$ with $\delta \geq 0$. (The case where $\delta > 0$ corresponds to a step i where k_i has a jump.) We have $\beta_i = g_i^2/g_{i-1}^2$ so β_i is well-defined and $t^{2\delta}$ divides β_i, by definition of order. Since $t^{k_{i-1}}$ divides p_i it follows that $t^{k_i} = t^{\delta}.t^{k_{i-1}}$ divides $\beta_i p_i$, and hence t^{k_i} divides $p_{i+1} = \beta_i p_i - g_i$.

We show that $g_{i-1} \cdot g_i = 0$.

$$g_i^T g_{i-1} = (g_{i-1} + \alpha_i A p_i)^T g_{i-1} = g_{i-1}^2 + \alpha_i p_i^T A(\beta_{i-1} p_{i-1} - p_i) = g_{i-1}^2 - \alpha_i p_i^T A p_i = 0$$

by the induction hypothesis $p_i^T A p_{i-1} = 0$ and the definition of α_i.

We show that β_i satisfies $\beta_i p_i^T A p_i = g_i^T A p_i$. Write P for $p_i^T A p_i$, then

$$g_i^2 P \beta_i = g_i^T (g_i - g_{i-1}) P \beta_i = g_i^T A p_i \alpha_i P \beta_i = g_i^T A p_i g_{i-1}^2 \beta_i = g_i^T A p_i g_i^2$$

since $g_{i-1} \cdot g_i = 0$ and $\alpha_i P = g_{i-1}^2$. We never calculate $g_i^2 P \beta_i$ as a truncated taylor in the course of the TTCG algorithm, but let us imagine that we calculate it now, to order $r + 2k_i + k_{i-1}$. We have that $g_i = v t^{k_i}$ with $v^{(0)} \neq 0$, so $g_i^2 = v^2 t^{2k_i}$. Divide both sides of the equation $g_i^2 P \beta_i = g_i^T A p_i g_i^2$ by t^{2k_i} and multiply by the truncated taylor $1/(v^2)$ to give the assertion for β_i to order $r - k_{i-1}$.

We show that $p_{i+1}^T A p_i = 0$.

$$p_{i+1}^T A p_i = (\beta_i p_i - g_i)^T A p_i = \beta_i p_i^T A p_i - g_i^T A p_i = 0$$

We show that $p_i \cdot g_i = 0$.

$$p_i^T g_i = (\beta_{i-1} p_{i-1} - g_{i-1})^T g_i = \beta_{i-1} p_{i-1}^T g_i = \beta_{i-1} p_{i-1}^T (g_{i-1} + \alpha_i A p_i) = \beta_{i-1} p_{i-1} \cdot g_{i-1} = 0$$

by the induction hypotheses.

We show that p_{i+1} has order exactly k_i. We already have that $t^{k_i} | p_{i+1}$. Suppose for a contradiction that $t^{k_i+1} | p_{i+1}$. Certainly $t^{k_i} | g_i$ so then t^{2k_i+1} divides $p_{i+1} \cdot g_i = \beta_i p_i \cdot g_i - g_i^2 = -g_i^2$, but this has order exactly $2k_i$ by definition.

This completes the inductive proof of case i. It remains to establish the base case $i = 1$. Since $p_1 = -g_0$ the order of p_1 is k_0. Since $A^{(0)}$ is positive definite, we have that $p_1^T A p_1$ and g_0^2 both have order exactly $2k_0$, whence α_1 is a well defined taylor not divisible by t. We have $g_1 \cdot g_0 = -g_1 \cdot p_1 = g_0^2 - \alpha_1 (p_1^T A p_1) = 0$.

The same arguments used in the induction step now give $k_1 \geq k_0$; β_1 well-defined; $\beta_1 p_1^T A p_1 = g_1^T A p_1$; $p_2^T A p_1 = 0$; p_2 has order k_1; and we are done.
qed

Corollary 2.2 (of the Proof) For all $i \geq 1$, β_i satisfies $\beta_i p_i^T A p_i = g_i^T A p_i$.

Theorem 2.3 For all i, j with $1 \leq i \leq j$ we have $g_{i-1} \cdot g_j = 0$ and $p_i^T A p_{j+1} = 0$.

Proof Assume that there is a counter-example, and consider one with the smallest value of j.

If $j = i$ then $g_j^T g_{i-1} = 0$ by Theorem 2.1, otherwise

$$g_j^T g_{i-1} = (g_{j-1} + \alpha_j A p_j)^T g_{i-1} = g_{j-1}^T g_{i-1} + \alpha_j p_j^T A(\beta_{i-1} p_{i-1} - p_i) = 0$$

by the assumption on j. In the base case $g_j^T g_0$ when $i = 1$, replace $\beta_0 p_0 - p_1$ by $-p_1$.

Similarly, if $j = i$ then $p_{j+1}^T A p_i = 0$ by Theorem 2.1, otherwise, for $i > 1$

$$p_{j+1}^T A p_i = (\beta_j p_j - g_j)^T A p_i = -g_j^T A p_i$$

by the assumption on j, whence

$$\alpha_i p_{j+1}^T A p_i = -g_j^T A(\alpha_i p_i) = -g_j^T (g_i - g_{i-1}) = -g_j^T g_i + g_j^T g_{i-1} = 0$$

by the case already proven. Now by Theorem 2.1 we have that the taylor α_j is not divisible by t, and hence has a taylor inverse α_j^{-1}. Multiplying by this, it follows that $p_i^T A p_{j+1} = 0$ and so there is no counter-example.
qed

Corollary 2.4 If $j > i$ then $g_i^{(k_i)} \cdot g_j^{(k_j)} = 0$

Proof We do not calculate elements of g_i to order greater than r in the TTCG algorithm itself, but this does not prevent us using their values in a proof. So, let us imagine for a moment that we executed the algorithm to order $2r$ instead of to order r. By Theorem 2.3, $g_i \cdot g_j = 0$, so

$$(g_i \cdot g_j)^{(k_i+k_j)} = g_i^{(0)} \cdot g_j^{(k_i+k_j)} + \ldots + g_i^{(k_i)} \cdot g_j^{(k_j)} + \ldots + g_i^{(k_i+k_j)} \cdot g_j^{(0)} = 0$$

But $g_i^{(\ell)} = 0$ for $\ell < k_i$, and $g_j^{(\ell)} = 0$ for $\ell < k_j$, so $g_i^{(k_i)} \cdot g_j^{(k_j)} = 0$
qed

Lemma 2.5 For some $i_* \leq n$ we have $g_{i_*} = 0$,
i.e. $g_{i_*}^{(k)} = 0$ for all $k : 0 \leq k \leq r$.

Proof This follows from Corollary 2.4 by the orthogonality of the sequence $g_i^{(k_i)}$. There are at most n linearly independent directions in the space.
qed

Theorem 2.6 The TTCG algorithm terminates after at most n steps, regardless of the value of r. Let x_{i_*} be the terminal value for x. Then $x_{i_*}^{(0)}$ is the solution of $A^{(0)} x^{(0)} = b^{(0)}$ and $x_{i_*}^{(1)}$ is the correct directional derivative $\dot{x}$ of x for the given directional derivatives $\dot{A} = A^{(1)}, \dot{b} = b^{(1)}$, and so on for the higher derivatives.

Proof At each stage g_i is the residual $Ax_i - b$. Since for $i = i_*$ we have $g_{i_*} = 0$, it follows that $Ax_{i_*} = b$ to order r, so

$$A^{(0)}x^{(0)}_{i_*} = b^{(0)} \qquad A^{(1)}x^{(0)}_{i_*} + A^{(0)}x^{(1)}_{i_*} = b^{(1)}$$

and so on, as required.
qed

3 Knowing When to Stop

The arithmetic used in AD is not usually exact, and so we must decide at which point to treat elements of g_i as zero: this amounts to deciding whether or not g_i is divisible by $t^{k_{i-1}+1}$, based on the sizes of the relevant elements. For example, when calculating the taylor $\beta_i = g_i^2/g_{i-1}^2$ we must decide whether to take

$$\beta_i^{(0)} = \frac{(g_i^2)^{(0)}}{(g_{i-1}^2)^{(0)}}, \qquad \text{or } \beta_i^{(0)} = \frac{(g_i^2)^{(2)}}{(g_{i-1}^2)^{(2)}}$$

where in the second case we use l'Hospital's rule (twice) to divide through by t^2 on the ground that $g_i^{(0)}$ is appropriately "small".

This is essentially similar to the problem we face with conventional CG in deciding when to terminate, and we may use similar heuristics to decide. For example we may decide to increment k_i whenever $g_i^{(k_{i-1})}$ becomes small relative to $g_0^{(k_0)}$. At each of these jumps in the value of k_i, the exact arithmetic version of the TTCG algorithm implicitly freezes the values for elements of x of order below k_i, which has the same effect as performing an internal CG restart at order k_i using the current value of $g_i^{(k_i)}$. It would be convenient in the case of inexact arithmetic to have a way of doing this explicitly as a result of a run-time test. Similarly we may decide to restart the complete TTCG algorithm from the beginning, with x_0 as the current value of x_i, whenever $g_i^{(k_i)}$ loses orthogonality with $g_0^{(k_0)}$. However the programmer needs a way to communicate these intentions effectively to the AD tool, and we also need a way of deciding when to terminate the algorithm on the ground that g_i is "close enough" to 0. A simple way of representing this is by defining a suitable norm on the taylor vector g_i (see the postscript following.)

The interesting feature of differentiating directly through the CG algorithm, beside conceptual and programming simplicity, is that both the solution and all its directional derivatives are obtained together in at most n iterations, where n is the dimension of x.

The approach of applying AD straight through removable singularities is likely to be exploitable along similar lines for other algorithms, once suitable mechanisms exist to allow AD tools to manage and adapt the order and norm of taylor variables.

A Postscript on Taylor Norms

Whenever we apply forward-mode AD to an iterative algorithm, we must be careful in the stopping condition to use a norm that is appropriate for taylor vectors. (Moré and Wild [5] present examples of what goes wrong if this is not done.) In particular $\|g\|$ must be non-zero and well-conditioned in the case when $g^{(0)}$ is very close to 0 but $g^{(1)}$ is still relatively large. This case may occur as a result of round-off error even when $g^{(0)}$ is analytically zero. One option (see p. 309 of [3]) is to define

$$|\alpha|_\gamma = \sum_{\ell=0}^{r} \gamma^\ell |\alpha^{(\ell)}|, \qquad \|g\|_\gamma = \sum_{\ell=0}^{r} \gamma^\ell \|g^{(\ell)}\|, \qquad \|A\|_\gamma = \sum_{\ell=0}^{r} \gamma^\ell \|A^{(\ell)}\| \quad \text{etc}$$

where we generally take the underlying vector norm $\|.\|$ to be the $1-$ or $2-$norm. Unlike t, which is a formal symbol, γ is a positive real number such as 1.0 or 0.5; an alternative possibility is a Sobolev-style p-norm with $1 \leq p \leq \infty$:

$$\|g\|_{k,p}^p = \|g^{(0)}\|_p^p + \|g^{(1)}\|_p^p + \ldots + \|g^{(k)}\|_p^p$$

As mentioned earlier, for the Euclidean norm we need to be careful when $g^{(0)}$ is close to 0, and we risk numerically unstable behaviour if we first use AD to calculate $\epsilon = \sqrt{g \cdot g}$ as a Taylor series, and then evaluate $|\epsilon|$. The first step gives

$$\epsilon = \|g^{(0)}\| + (\hat{g}^{(0)} \cdot g^{(1)})t + O(t^2) \qquad \text{where} \qquad \hat{g}^{(0)} = \frac{g^{(0)}}{\|g^{(0)}\|}$$

and the term in t, which results from the cross-terms in the inner product $g \cdot g$, may take any value between 0 and $\|g^{(1)}\|$: the moral is that we really do need to calculate the norm of the taylor vector directly, as an atomic step.

References

[1] Loyce Adams and Larry Nazareth (eds), *Linear and Non-linear Conjugate Gradient-Related Methods*, Society for Industrial and Applied Mathematics, 1996.

[2] Serge Gratton et al., *Differentiating the Method of Conjugate Gradients*, SIAM. J. Matrix Anal & Appl., 35(1), 110–126, 2014.

[3] Andreas Griewank and Andrea Walther, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, Society for Industrial and Applied Mathematics, 2nd edition, November 2008.

[4] Andreas Griewank and Andrea Walther, *Getting started with ADOL-C*, Combinatorial Scientific Computing, Chapman-Hall CRC Computational Science, 2012

[5] Jorge Moré and Stefan Wild, *Do You Trust Derivatives or Differences*, J. Comp. Physics, 273, 268–277, September 2014.