Notes on adjoint methods for recurrence relations

Steven G. Johnson

October 11, 2007

1 Introduction

In these notes, we consider solutions $\mathbf{x}^n$ ($\mathbf{x}$ is some K-component vector at a discrete "time" n) to a discrete recurrence relation that depends upon some parameters $\mathbf{p}$ (a P-component vector), and derive how to compute gradients of functions $g(\mathbf{x})$ with respect to $\mathbf{p}$ by using an adjoint method. Adjoint methods and gradients are used to perform sensitivity analyses, gradient-based optimization, etcetera, and the key fact is that they provide ways to compute derivatives of g with respect to many parameters $\mathbf{p}$ in about the same time as it takes to compute g twice. (In contrast, finite-difference differentiation, or even straightforward symbolic differentiation, would have a cost proportional to the cost of evaluating g multiplied by the number P of parameters to differentiate by.)

For a general overview of adjoint methods in simple circumstances, see our notes for 18.336 at MIT [1]. The method presented here for recurrence relations, however, is more complicated, and is based in spirit on a related method developed for differential-algebraic equations [2].

2 Problem formulation and notation

A general recurrence relation for $\mathbf{x}^n$ is a relationship between $\mathbf{x}^n$ and $\mathbf{x}^{n-1}$ of the form:

$$\mathbf{f}^n \triangleq \mathbf{f}(\mathbf{x}^n, \mathbf{x}^{n-1}, n, \mathbf{p}) = \mathbf{0}, \tag{1}$$

where $\mathbf{f}(\mathbf{x}, \mathbf{y}, n, \mathbf{p})$ is some given function, possibly depending explicitly on n and on the parameters $\mathbf{p}$. We use superscripts n to indicate the "time" of evaluation. For example, a simple recurrence relation $\mathbf{x}^n = 2\mathbf{x}^{n-1}$ would be represented by $\mathbf{f}(\mathbf{x}, \mathbf{y}, n, \mathbf{p}) = \mathbf{x} - 2\mathbf{y}$. One might imagine that a more general recurrence relation could be obtained in which $\mathbf{x}^n$ depends on both $\mathbf{x}^{n-1}$ and $\mathbf{x}^{n-2}$, and so on, but this can still be expressed in the form (1) by replacing $\mathbf{x}$ with a new vector of twice the length containing both $\mathbf{x}^n$ and $\mathbf{x}^{n-1}$ (in much the same way that a 2nd-order ODE can be converted into two 1st-order ODEs). Of course, we also need an initial condition (which may depend on $\mathbf{p}$):

$$\mathbf{x}^0 = \mathbf{b}(\mathbf{p}).$$

Now suppose that we have some function $g^n \triangleq g(\mathbf{x}^n, n, \mathbf{p})$ that we want to compute derivatives of with respect to $\mathbf{p}$. Similar to Cao et al. [2], it turns out to be easier to first consider a more general function, by "integrating" g with respect to n:

$$G^N \triangleq G(N, \mathbf{p}) = \sum_{n=0}^{N} g(\mathbf{x}^n, n, \mathbf{p}) = \sum_{n=0}^{N} g^n.$$

Now, we want to compute the gradient $\frac{dG^N}{d\mathbf{p}}$, which is formally given by:

$$\frac{dG^N}{d\mathbf{p}} = \sum_{n=0}^{N} \left[g_{\mathbf{p}}^n + g_{\mathbf{x}}^n \mathbf{x}_{\mathbf{p}}^n \right],$$

where subscripts denote partial derivatives, and should be thought of as row vectors (versus column vectors $\mathbf{x}$ and $\mathbf{p}$). The problem with the expression above is that computing $\mathbf{x}_{\mathbf{p}}$ (a $K \times P$ matrix) is hard. The idea of the adjoint method is to use an algebraic trick to eliminate this troublesome term.

3 The adjoint method for G

The basic trick behind the adjoint method is to insert a new auxiliary vector $\boldsymbol{\lambda}^n$ (also K components), similar in some ways to a Lagrange multiplier, and an auxiliary function $\tilde{G}^N$, chosen so that the derivatives of $\tilde{G}^N$ match the derivatives of G but without the troublesome $\mathbf{x}_{\mathbf{p}}$ term. $\boldsymbol{\lambda}^n$ will turn out to be the solution of another recurrence relation, going *backwards* from $n = N$ to $n = 0$, so that to find both G and its gradient you only need to solve two recurrence relations. In particular, we define $\tilde{G}^N$ by:

$$\tilde{G}^N \triangleq G^N - \sum_{n=1}^{N} \boldsymbol{\lambda}^{n-1} \cdot \mathbf{f}^n.$$

Note that, since $\mathbf{x}^n$ solves the recurrence relation $\mathbf{f}^n = \mathbf{0}$, we have just added zero to G^N, which obviously doesn't change the gradient *regardless* of the value of $\boldsymbol{\lambda}^n$. However, it *does* allow us to express the gradient in a different way, by choosing $\boldsymbol{\lambda}^n$ appropriately. In particular:

$$\frac{d\tilde{G}^N}{d\mathbf{p}} = g_{\mathbf{p}}^0 + g_{\mathbf{x}}^0 \mathbf{b}_{\mathbf{p}} + \sum_{n=1}^{N} \left[g_{\mathbf{p}}^n - \boldsymbol{\lambda}^{n-1} \cdot \mathbf{f}_{\mathbf{p}}^n + \left(g_{\mathbf{x}}^n - \boldsymbol{\lambda}^{n-1} \cdot \mathbf{f}_{\mathbf{x}}^n \right) \mathbf{x}_{\mathbf{p}}^n - \boldsymbol{\lambda}^{n-1} \cdot \mathbf{f}_{\mathbf{y}}^n \mathbf{x}_{\mathbf{p}}^{n-1} \right],$$

in which the last term can be re-indexed ($n - 1 \to n$) in order to group all of the $\mathbf{x}_{\mathbf{p}}$ terms together:

$$\begin{aligned}\frac{d\tilde{G}^N}{d\mathbf{p}} = {} & g_{\mathbf{p}}^0 + g_{\mathbf{x}}^0 \mathbf{b}_{\mathbf{p}} - \boldsymbol{\lambda}^0 \cdot \mathbf{f}_{\mathbf{y}}^1 \mathbf{b}_{\mathbf{p}} + g_{\mathbf{p}}^N - \boldsymbol{\lambda}^{N-1} \cdot \mathbf{f}_{\mathbf{p}}^N + \left(g_{\mathbf{x}}^N - \boldsymbol{\lambda}^{N-1} \cdot \mathbf{f}_{\mathbf{x}}^N \right) \mathbf{x}_{\mathbf{p}}^N \\ & + \sum_{n=1}^{N-1} \left[g_{\mathbf{p}}^n - \boldsymbol{\lambda}^{n-1} \cdot \mathbf{f}_{\mathbf{p}}^n + \left(g_{\mathbf{x}}^n - \boldsymbol{\lambda}^{n-1} \cdot \mathbf{f}_{\mathbf{x}}^n - \boldsymbol{\lambda}^n \cdot \mathbf{f}_{\mathbf{y}}^{n+1} \right) \mathbf{x}_{\mathbf{p}}^n \right].\end{aligned}$$

This expression looks rather complicated, but it can be vastly simplified: so far, $\boldsymbol{\lambda}^n$ has been completely arbitrary, so we can choose it to be *anything* we want to make the gradient easier to compute. In particular, we will choose $\boldsymbol{\lambda}^n$ to eliminate the troublesome $\mathbf{x}_\mathbf{p}$ terms, by simply setting the parenthesized expressions $(\cdots)$ to zero. This gives us two equations for $\boldsymbol{\lambda}^n$:

$$\boldsymbol{\lambda}^{N-1} \cdot \mathbf{f}_\mathbf{x}^N = g_\mathbf{x}^N, \tag{2}$$

$$\boldsymbol{\lambda}^{n-1} \cdot \mathbf{f}_\mathbf{x}^n = g_\mathbf{x}^n - \boldsymbol{\lambda}^n \cdot \mathbf{f}_\mathbf{y}^{n+1}. \tag{3}$$

The second equation is a *backwards recurrence* equation for $\boldsymbol{\lambda}^{n-1}$ in terms of $\boldsymbol{\lambda}^n$ (or rather, a set of K equations in K unknowns), and the first equation is the "initial" condition for this "adjoint" recurrence.

Once we have solved these adjoint recurrences (2–3), we can then plug them back in to find our gradient:

$$\frac{dG^N}{d\mathbf{p}} = \frac{d\tilde{G}^N}{d\mathbf{p}} = g_\mathbf{p}^0 + g_\mathbf{x}^0 \mathbf{b}_\mathbf{p} - \boldsymbol{\lambda}^0 \cdot \mathbf{f}_\mathbf{y}^1 \mathbf{b}_\mathbf{p} + \sum_{n=1}^{N} \left[g_\mathbf{p}^n - \boldsymbol{\lambda}^{n-1} \cdot \mathbf{f}_\mathbf{p}^n \right]. \tag{4}$$

4 The adjoint method for g

In many cases, we only care about the gradient of g^N, not G^N. That is, we have a function just of the final result of the recurrence, not of all the intermediate steps. We can immediately compute this from our result (4) above since, by definition, $g^N = G^N - G^{N-1}$. However, one must be careful of one thing: the $\boldsymbol{\lambda}^n$ adjoint variables are different when taking the gradient of G^N and G^{N-1}, since the initial condition (2) changes. Denote the latter adjoint variable by μ^n (the solution to the adjoint recurrence with N replaced by $N-1$). The expressions that will appear in our gradient equation will involve $\tilde{\boldsymbol{\lambda}}^n = \boldsymbol{\lambda}^n - \mu^n$. In particular, we obtain:

$$\frac{dg^N}{d\mathbf{p}} = \frac{dG^N}{d\mathbf{p}} - \frac{dG^{N-1}}{d\mathbf{p}} = g_\mathbf{p}^N - \boldsymbol{\lambda}^{N-1} \cdot \mathbf{f}_\mathbf{p}^N - \tilde{\boldsymbol{\lambda}}^0 \cdot \mathbf{f}_\mathbf{y}^1 \mathbf{b}_\mathbf{p} - \sum_{n=1}^{N-1} \tilde{\boldsymbol{\lambda}}^{n-1} \cdot \mathbf{f}_\mathbf{p}^n. \tag{5}$$

In order to evaluate this expression, we need $\boldsymbol{\lambda}^{N-1}$ from eq. (2), and we also need $\tilde{\boldsymbol{\lambda}}^n$. However, because the adjoint recurrence (3) is *linear* in $\boldsymbol{\lambda}$ (*regardless* of whether the original recurrence in $\mathbf{x}$ was linear), we can simply subtract the $\boldsymbol{\lambda}$ and μ recurrences to get a single simplified recurrence in $\tilde{\boldsymbol{\lambda}}$:

$$\tilde{\boldsymbol{\lambda}}^{n-1} \cdot \mathbf{f}_\mathbf{x}^n = -\tilde{\boldsymbol{\lambda}}^n \cdot \mathbf{f}_\mathbf{y}^{n+1}. \tag{6}$$

The initial condition for this backwards recurrence is the following equation for $\tilde{\boldsymbol{\lambda}}^{N-2}$:

$$\tilde{\boldsymbol{\lambda}}^{N-2} \cdot \mathbf{f}_\mathbf{x}^{N-1} = -\boldsymbol{\lambda}^{N-1} \cdot \mathbf{f}_\mathbf{y}^N \tag{7}$$

in terms of $\boldsymbol{\lambda}^{N-1}$ from eq. (2).

Equation (5) doesn't seem as nice as one might like when one thinks of the computational cost. First, one solves two recurrences, one for $\mathbf{x}^n$ and one for $\tilde{\boldsymbol{\lambda}}^n$, which isn't bad. Then, to evaluate equation (5) the first three terms require $O(PK)$ work independent of N. However, the last term, the $\sum_n$, seems problematic: $\mathbf{f}_\mathbf{p}$ is a $K \times P$ matrix, so evaluating each summand requires $O(PK)$ work and thus the sum requires $O(NPK)$ work in general. Assuming that each solution of the recurrence takes $O(NK)$ time, which is typical (although it could be worse, e.g. if a $K \times K$ matrix must be inverted for each n), then the gradient evaluation seems to require roughly P times the work of evaluating g^N once—this is the thing we are trying to *avoid* with adjoint methods!

However, the situation in practice is unlikely to be this bad, because often the matrix $\mathbf{f}_\mathbf{p}$ will be very sparse. For example, each component of $\mathbf{p}$ may affect only a few n, or a few components of $\mathbf{x}$. An example of this is given below.

5 A simple example

The above is rather abstract, so it is good to illustrate it with a simple example. Consider the following two-component ($K = 2$) recurrence:

$$\mathbf{x}^n = \begin{pmatrix} 1 & 0.1 \\ 0.1 & 1 \end{pmatrix} \mathbf{x}^{n-1} + \begin{pmatrix} 0 \\ p_n \end{pmatrix} = A\mathbf{x}^{n-1} + \begin{pmatrix} 0 \\ p_n \end{pmatrix}$$

with initial conditions $\mathbf{b} = (1, 0)$. That is, in this case our parameters $\mathbf{p}$ are a "source" term p_n at every n (so that $P = N$). The corresponding function $\mathbf{f}^n$ is:

$$\mathbf{f}^n(\mathbf{x}, \mathbf{y}, \mathbf{p}) = \mathbf{x} - A\mathbf{y} - \begin{pmatrix} 0 \\ p_n \end{pmatrix}.$$

Now, we must pick some function g to take the gradient of. For simplicity, let's just take

$$g(\mathbf{x}) = x_1,$$

with no explicit dependence on $\mathbf{p}$.

At this point, we merely take the appropriate derivatives to write down our adjoint equations (2,6–7). $\mathbf{f}_\mathbf{x}$ here is the identity matrix, and $\mathbf{f}_\mathbf{y} = -A$, so:

$$\boldsymbol{\lambda}^{N-1} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad \tilde{\boldsymbol{\lambda}}^{N-2} = A\boldsymbol{\lambda}^{N-1},$$

$$\tilde{\boldsymbol{\lambda}}^{n-1} = A\tilde{\boldsymbol{\lambda}}^{n},$$

thereby obtaining our gradient:

$$\frac{dg^N}{dp_N} = \boldsymbol{\lambda}^{N-1} \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix},$$

$$\frac{dg^N}{dp_{n<N}} = \tilde{\boldsymbol{\lambda}}^{n-1} \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix},$$

where we have used the fact that $\mathbf{f}_\mathbf{p}^n$ is a $2 \times N$ matrix that is all zeros except for its nth column $(0, -1)$. In this way, evaluating the gradient requires only $O(N)$ time, as opposed to $O(N^2)$ time if we had proceeded naively.

References

[1] S. G. Johnson, “Notes on adjoint methods for 18.336.” Online at `http://math.mit.edu/ stevenj/18.336/adjoint.pdf`, May 2006.

[2] Y. Cao, S. Li, L. Petzold, and R. Serban, “Adjoint sensitivity analysis for differential-algebraic equations: The adjoint DAE system and its numerical solution,” *SIAM J. Sci. Comput.*, vol. 24, no. 3, pp. 1076–1089, 2003.