REVERSE ACCUMULATION AND ATTRACTIVE FIXED POINTS

Bruce Christianson

School of Information Sciences, University of Hertfordshire
Hatfield, Herts AL10 9AB, England, Europe

Numerical Optimisation Centre Technical Report 258, March 1992
published Optimization Methods and Software (1994) 3,311–326

Abstract

We apply reverse accumulation to obtain automatic gradients and error estimates of functions which include in their computation a convergent iteration of the form $y := \Phi(y, u)$, where y and u are vectors.

We suggest an implementation approach which allows this to be done by a fairly routine extension of existing reverse accumulation code. We show how to re-use the computational graph for the fixed point constructor Φ so as to set explicit stopping criteria for the iterations, based on the gradient accuracy required.

Our construction allows the gradient vector to be obtained to the same order of accuracy as the objective function values (which is in general the best we can hope to achieve), and the same order of computational cost (which does not explicitly depend upon the number of independent variables.) The technique can be applied to functions which contain several iterative constructions, either serially or nested.

1. Introduction. Automatic differentiation [7][10] is a set of techniques for obtaining derivatives of numerical functions to the same order of accuracy as the function values themselves, but without the labour of forming explicit symbolic expressions for the derivative functions. Automatic differentiation works by repeated use of the chain rule, but applied to numerical expressions rather than to symbolic values.

The forward accumulation technique of automatic differentiation associates with each program variable a vector containing partial derivatives of that variable with respect to the independent variables. The reverse accumulation technique builds a computational graph for the function evaluation, with one node for each value ever held by a program variable, and associates with each node an *adjoint* value containing the partial derivative of the function value with respect to the node. The adjoint values are calculated in the opposite order to the function evaluation, whence the term 'reverse accumulation'. Reverse accumulation is of particular interest when the gradient (or sensitivities) of a single target (loss) function is required, as it will allow the evaluation of the entire gradient vector of a scalar function at a computational cost of three function evaluations in total, regardless of the number of independent variables. For further details, see [7], [10] and the references therein.

Many functions include in their computation a convergent iteration of steps of the form $y := \Phi(y, u)$, where u and y are (column) vectors. We can regard the iterative process Φ with starting point $(y_0(u), u)$ as the constructive evaluation at u of a branch of the function $y_*(u)$ defined by $\Phi(y_*(u), u) = y_*(u)$, corresponding to the appropriate basin of attraction for y_0. This fixed point construction for y_* is typically part of a larger computation of a dependent variable $z = f(x, y)$, where the x are the independent variables, $y = y_*(u)$, and the u are also functions of the x.

It is clearly desirable to extend the automatic differentiation techniques so as to determine such quantities as the Jacobean derivative matrix $\nabla_u \, y_*(u)$ and the gradient (row) vector $\nabla_x \, z$. It is well known that convergence of function values to a given tolerance does not imply convergence of the corresponding derivative values to the same tolerance (or indeed at all) and so the question of determining convergence conditions and stopping criteria is nontrivial.

The paper [6] addresses these issues and proposes an approach to the automatic evaluation of first derivatives. In the context of reverse accumulation, the paper raises the problem of adapting the number of iterations so as to obtain the desired degree of accuracy for the derivative values. It is our purpose to address this problem here.

In this note we give a simple technique for using reverse accumulation to obtain the first derivatives of functions which include iterative fixed points in their construction, and we show how to adjust the number of iterations on the reverse pass so as to satisfy an appropriate stopping criterion based upon the derivative accuracies required.

In the next section, following some mathematical preliminaries, we show that the adjoint quantities $\bar{u}$ also satisfy a fixed point equation with appropriate convergence properties. In section 3, we use this to suggest an implementation approach which could be built on top of existing code. In section 4 we show how to obtain accurate roundoff and truncation error bounds on objective function values which are calculated with a fixed point constructor, and in section 5 we apply this to show how to adjust the number of iterations on the forward and reverse passes so as to obtain the desired accuracy for the

gradient values. The implications of this for the implementation of self-adapting code are briefly discussed in section 6. Our conclusions are summarized in the final section.

2. Adjoint Fixed Point Equation. In what follows, we assume a norm $\|b\|$ for column vectors b, and denote by the same symbol the corresponding operator norm $\|A\| = \sup\{\|Ab\| \ st \ \|b\| = 1\}$. We write $\|a\|$ for the norm of a row vector a considered as a linear operator on column vectors, so that $\|a\| = \|a^T\|_{adj}$ where T denotes the adjoint (transpose). Usually we shall be using the Euclidean norm, in which case $\|a\|_2 = \|a^T\|_2$, and the corresponding operator norm is the spectral norm. But when working with interval bounds, it is frequently convenient to use instead a weighted ∞ norm for column vectors, in which case $\|a\|$ will be the corresponding counter-weighted 1 norm and the operator norm $\|A\|$ can be easily calculated from the matrix representation of A. Because we are dealing with finite dimensional spaces, all complete norms are equivalent and we can speak without ambiguity of an open "neighbourhood" of a point. However the equivalence bounds can be arbitrarily large.

We regard the gradient operator ∇ as an adjoint operation, mapping scalars into row vectors and column vectors into Jacobean matrices.

If Φ is an iterative mapping of the form $\Phi(y,u) : R^q \times R^p \to R^q$ then we write Φ_y, Φ_u to denote $\nabla_y\Phi, \nabla_u\Phi$ respectively, when these derivatives exist. At any such point (y,u), $\Phi'(y,u) = (\Phi_y(y,u), \Phi_u(y,u))$ is a map $R^q \times R^p \to \mathcal{L}(R^q \times R^p \to R^q)$. In this case we say that Φ' is Lipschitz with respect to a given norm if the map $(y,u) \to \Phi'(y,u)$ is Lipschitz with respect to the induced norm.

Definition 2.1 Fix the norm $\|.\|$ on $R^q \times R^p$, this induces norms on $R^q = R^q \times \{0\}, R^p = \{0\} \times R^p$ and on $\mathcal{L}(R^q \times R^p \to R^q)$. Let U, Y be bounded open convex subsets of R^p, R^q respectively, and let $y_*(u)$ be a continuous function $U \to Y$. Since we are interested in local behaviour with respect to u, we lose nothing by restricting our attention to domains of this form.

We say that $\Phi : Y \times U \to Y$ is a *well behaved iterative constructor* for the function y_* if the following four conditions are satisfied:

(a) $y_*(u) = \Phi(y_*(u), u)$ for all u in U

(b) Φ is continuously differentiable throughout $Y \times U$

(c) Φ' is Lipschitz with constant C throughout $Y \times U$ for some constant C

(d) y_* is an *attractive* fixed point of Φ for the domain $Y \times U$,

ie there is some constant $\tau < 1$ called the *attraction* constant such that $\|\Phi_y(y,u)\| < \tau$ for all u in U and y in Y

Note that under these conditions there will always exist a constant κ with $\|\Phi_u(y,u)\| < \kappa$ for all u in U and y in Y. Note also that some definitions of attraction only require $\|\Phi_y(y_*(u),u)\| < \tau$ for u in U. However, by (c) this condition is equivalent to our condition (d) for a larger $\tau < 1$ and (possibly) smaller convex sets Y and U containing the point of interest.

In practice, these conditions are not unduly restrictive, since in a sufficiently close neighbourhood of a given point $(y_*(u), u)$ (which is all we require), they will frequently be satisfied by some (smallish) power Φ^n of Φ (iterated on the first argument), even if not by

Φ itself. It is also worth pointing out that the results we give here remain true in the case where Φ' is uniformly continuous (rather than Lipschitz) throughout $Y \times U$. The methods of proof are the same, but the construction of bounds is notationally more cumbersome.

Theorem 2.2 Suppose Φ a well behaved iterative constructor for y_* on the open domain $Y \times U$, with Φ' Lipschitz with constant C, and $\|\Phi_y\|, \|\Phi_u\|$ bounded throughout $Y \times U$ by τ, κ respectively.

Then $y_*(u)$ is continuously differentiable with respect to u, with gradient given by

$$\nabla_u \; y_*(u) = \frac{\partial y_*}{\partial u}(u) = (I - \Phi_y(y_*(u), u))^{-1} \Phi_u(y_*(u), u)$$

Furthermore

$$\|\nabla_u \; y_*(u)\| < \frac{\kappa}{1-\tau} \quad \text{for all } u \text{ in } U,$$

the map $(y, u) \to (I - \Phi_y(y, u))^{-1} \Phi_u(y, u)$ is Lipschitz with constant

$$C \; \frac{1 - \tau + \kappa}{(1-\tau)^2}$$

and the map $u \to \nabla_u \; y_*(u)$ is Lipschitz with constant

$$C \; \frac{(1 - \tau + \kappa)^2}{(1-\tau)^3}$$

Finally, for any function $y_0 : U \to Y$ (whether continuous or not) we have for all u in U

$$\|y_*(u) - y_{n+1}(u)\| \leq \tau \|y_*(u) - y_n(u)\|$$

and hence

$$\|y_*(u) - y_n(u)\| \leq \frac{\|y_n(u) - y_{n+1}(u)\|}{1-\tau} \leq \|y_0(u) - y_1(u)\| \cdot \frac{\tau^n}{1-\tau}$$

where $y_{n+1}(u) = \Phi(y_n(u), u)$

Proof: Since y_* is a continuous function of u it is easy to show that

$$\delta y_* = \Phi_y(y_*(u), u) \delta y_* + \Phi_u(y_*(u), u) \delta u$$

to first order in δu, and hence

$$\frac{\partial y_*}{\partial u}(u) = (I - \Phi_y(y_*(u), u))^{-1} \Phi_u(y_*(u), u)$$

provided the inverse exists. Since we have assumed that $\|\Phi_y(y_*(u), u)\| < \tau < 1$, we can establish the existence of the inverse directly:

$$(I - \Phi_y(y_*(u), u))^{-1} = I + \Phi_y + \Phi_y^2 + \Phi_y^3 + \cdots + \Phi_y^n + \ldots$$

where the RHS must converge to an operator with norm at most $1/(1-\tau)$. Since $\|\Phi_u(y_*(u),u)\| < \kappa$, the bound on $\|\nabla_u\ y_*(u)\|$ follows.

Now suppose u_0, u in U; y_0, y in Y with $\|(y,u)-(y_0,u_0)\| < \epsilon$ and define

$$\Psi = \Phi_y(y,u), \qquad \Psi_0 = \Phi_y(y_0,u_0), \qquad \delta\Psi = \Psi - \Psi_0$$

Since the map $\Phi' : (y,u) \to (\Phi_y(y,u), \Phi_u(y,u))$ is Lipschitz with constant C throughout $Y \times U$, we have $\|\delta\Psi\| \le C\epsilon$. Now clearly

$$\Psi^n - \Psi_0^n = \delta\Psi(\Psi^{n-1} + \Psi^{n-2}\Psi_0 + \cdots + \Psi\Psi_0^{n-2} + \Psi_0^{n-1})$$

so

$$\|\Psi^n - \Psi_0^n\| \le n\tau^{n-1}\|\delta\Psi\| \le nC\epsilon\tau^{n-1}$$

and so

$$\|(I-\Phi_y(y,u))^{-1} - (I-\Phi_y(y_0,u_0))^{-1}\| = \|(I-\Psi)^{-1} - (I-\Psi_0)^{-1}\|$$

$$\le \sum_{n=1}^{\infty} \|\Psi^k - \Psi_0^k\| \le \sum_{n=1}^{\infty} nC\epsilon\tau^{n-1} = \frac{C\epsilon}{(1-\tau)^2}$$

Now define

$$\chi = \Phi_u(y,u), \qquad \chi_0 = \Phi_u(y_0,u_0), \qquad \delta\chi = \chi - \chi_0$$

Since also

$$\|\delta\chi\| = \|\Phi_u(y,u) - \Phi_u(y_0,u_0)\| \le C\epsilon$$

we have that

$$\|(I-\Psi)^{-1}\chi - (I-\Psi_0)^{-1}\chi_0\| \le \|(I-\Psi)^{-1}\| \cdot \|\delta\chi\| + \|(I-\Psi)^{-1} - (I-\Psi_0)^{-1}\| \cdot \|\chi_0\|$$

$$\le \frac{C\epsilon}{1-\tau} + \frac{C\epsilon\kappa}{(1-\tau)^2} = C\epsilon\left(\frac{1-\tau+\kappa}{(1-\tau)^2}\right)$$

A line integral of $\nabla_u\ y_*(u)$ from u_0 to u shows that

$$\|(y_*(u),u) - (y_*(u_0),u_0)\| \le \|y_*(u) - y_*(u_0)\| + \|u-u_0\| \le \left(\frac{\kappa}{1-\tau} + 1\right)\|u-u_0\|$$

$$\le \frac{1+\kappa-\tau}{1-\tau}\|u-u_0\|$$

which with the previous result gives the Lipschitz constant for the map $u \to \nabla_u\ y_*(u)$, while a line integral of $\Phi_y(y,u)$ from $(y_n(u),u)$ to $(y_*(u),u)$ establishes the next part. The final equation follows by induction since

$$\|y_*(u) - y_0(u)\| \le \|y_*(u) - y_1(u)\| + \|y_0(u) - y_1(u)\| \le \tau\|y_*(u) - y_0(u)\| + \|y_0(u) - y_1(u)\|$$

QED.

Theorem 2.3 Suppose Φ a well behaved iterative constructor for y_* on U, with attraction constant τ, and let r be a fixed arbitrary adjoint (row) vector in $\mathcal{L}(R^q \to R) = (R^q)^T$.

Given u and y in U and Y respectively, let $\zeta_*(y,u)$ be the (unique) fixed point of the equation $\zeta = r + \zeta\Phi_y(y,u)$ and set for the given u, y

$$\zeta_0 = 0, \qquad \zeta_{n+1} = r + \zeta_n\Phi_y(y,u)$$

Then

$$\|\zeta_* - \zeta_n\| \leq \|\zeta_n - \zeta_{n+1}\| \cdot \frac{1}{1-\tau}$$

and

$$\|\zeta_* - \zeta_n\| \leq \|r\| \cdot \frac{\tau^n}{1-\tau}$$

uniformly for y and u.

In particular, if $y = y_*(u)$ then the value of $r\ \nabla_u\ (y_*(u))$ is given by $\zeta_*(y_*(u),u)\Phi_u(y_*(u),u)$, so that

$$\|r\ \nabla_u(y_*(u),u) - \zeta_n(y_*(u),u)\Phi_u(y_*(u),u)\| \leq \|\zeta_n(y_*(u),u) - \zeta_{n+1}(y_*(u),u)\| \cdot \frac{\kappa}{1-\tau}$$

Finally, if second order derivatives of Φ exist and are Lipschitz, and Z is the open ball in R^q with centre $\{0\}$ and radius $\|r\|/(1-\tau)$, then the map $Z \times U \to Z$:

$$(\zeta^T, u) \to [r + \zeta\Phi_y(y_*(u),u)]^T$$

is a well behaved iterative constructor for $\zeta_*^T(y_*(u),u)$ on $Z \times U$, and also has attraction constant τ.

Proof: If r is an adjoint (row) vector, then

$$\zeta_* = r + r\Phi_y + r\Phi_y^2 + \ldots = r(I - \Phi_y)^{-1}$$

is the fixed point of the equation $\zeta = r + \zeta\Phi_y$.

If we set $\zeta_0 = 0, \zeta_{n+1} = r + \zeta_n\Phi_y$ then

$$\|\zeta_n - \zeta_*\| \leq \|\zeta_n - \zeta_{n+1}\| + \|\zeta_{n+1} - \zeta_*\|$$

$$\|\zeta_{n+1} - \zeta_*\| = \|(r + \zeta_n\Phi_y) - (r + \zeta_*\Phi_y)\| \leq \|\zeta_n - \zeta_*\|\|\Phi_y\| \leq \tau\|\zeta_n - \zeta_*\|$$

$$(1-\tau)\|\zeta_n - \zeta_*\| \leq \|\zeta_n - \zeta_{n+1}\|$$

whence $\|\zeta_* - \zeta_n\Phi_u\| \leq \|\zeta_n - \zeta_{n+1}\| \cdot 1/(1-\tau)$. Similarly

$$\|\zeta_n - \zeta_{n+1}\| \leq \tau\|\zeta_{n-1} - \zeta_n\| \leq \tau^n\|\zeta_0 - \zeta_1\| = \tau^n\|r\|$$

whence $\|\zeta_* - \zeta_n\| \leq \|r\| \cdot \tau^n/(1-\tau)$.

From Theorem 2.2 we have that

$$r\ \nabla_u\ (y_*(u)) = r(I - \Phi_y(y_*(u),u))^{-1}\Phi_u(y_*(u),u) = \zeta_*(y_*(u),u)\Phi_u(y_*(u),u)$$

The final assertion (which we shall not use in the rest of this paper) is now straightforward.

QED.

This result shows that, under fairly mild assumptions, the (uniform) rate of convergence of the derivative fixed point is the same as the asymptotic rate of (uniform) convergence for y_* itself. This is also true in the particular case of quadratic convergence of y_n to y_*, since in this case $\Phi_y(y_*(u), u) = 0$.

3. Implementation Strategy. We seek to use the technique of reverse accumulation to calculate row vectors of the form $r \nabla_u y_*(u)$ for fixed row (adjoint) vectors r.

We can use the construction in Theorem 2.3 to do this, since a reverse pass through the graph of Φ is a way of forming adjoint vectors of the form $r\Phi_y$ and $r\Phi_u$, and we can expand $r \nabla_u y_*$ as

$$r \cdot \frac{\partial y_*}{\partial u} = r\Phi_u + r\Phi_y\Phi_u + r\Phi_y^2\Phi_u + r\Phi_y^3\Phi_u + \cdots + r\Phi_y^n\Phi_u + \cdots$$

This leads to the following.

Algorithm 3.1 Suppose as before that Φ a well-behaved iterative constructor for y_*, and assume now that the fixed point construction for y_* is part of a larger computation of a dependent variable $z = f(x, y)$, where the x are the independent variables, $y = y_*(u)$, and the u are also functions of the x.

To evaluate $\nabla_x z$ proceed as follows:

Step 1. Build the part of the computational graph corresponding to the calculation of u from the independent variables x.

Step 2. Switch off building the graph and perform the iterations which construct $y_*(u)$ from some starting values y_0 until the convergence criterion is met. (We shall discuss the convergence criterion further below. The starting values y_0 are irrelevant to the further analysis.)

Step 3. Set $y_{initial}$ to the calculated value for $y_*(u)$, and switch on graph construction through one further iteration $y_{final} = \Phi(y_{initial}, u)$, where we treat $y_{initial}$ as additional independent variables. We should have $y_{final} = y_{initial} = y_*(u)$ to the desired level of accuracy (which we quantify in §5).

Step 4. Continue building the rest of the graph for the dependent variable $z = f(x, y_{final})$.

To reverse accumulate the gradient $\nabla_x z$ proceed as follows:

Step 5. Initialize $\bar{z}$ to 1 as usual.

Step 6. Reverse through the graph from z to y_{final}, accumulating the adjoint quantities $\bar{y}_{final} = r$, say.

Step 7. Reverse through the portion of the graph corresponding to Φ, accumulating the adjoint values $\bar{u}, \bar{y}_{initial}$.

Step 8. If $\bar{y}_{initial} + r$ is sufficiently close to $\bar{y}_{final}$ then go to step 10, else go to step 9. (We shall quantify what "sufficiently close" means in §5.)

Step 9. Set $\bar{y}_{final} = r + \bar{y}_{initial}$, set $\bar{u} = \bar{y}_{initial} = 0$ and go to step 7. (This will require us to reverse through the *same* portion of the graph again.)

Step 10. Reverse through the graph from u to x, accumulating the adjoints $\bar{x}$ of the independent variables x.

Pseudocode for this algorithm will be given in §6.

At the start of step 10, $\bar{u}$ will be a good approximation for $r \ \nabla_u \ y_*(u)$. To see this, assume for the moment that $y_{initial} = y_{final} = y_*(u)$, then (in the notation of Theorem 2.3) after the n-th reverse pass through the graph for Φ we have

$$\bar{y}_{final} = \zeta_n, \quad \bar{y}_{initial} = \zeta_n \ \Phi_y, \quad \bar{u} = \zeta_n \ \Phi_u$$

so convergence is established by Theorem 2.3.

Since (as observed in §2) the forward and reverse constructions have the same rate of convergence, we should expect to see roughly the same number of iterations in Step 2 as of Step 7. In the event that the fixed point construction has quadratic convergence to y_*, the reverse accumulation Step 7 should converge after a single pass through Φ since under this assumption $\Phi_y(y_*(u), u) = 0$. If this is not the case, and $\tau \cdot \kappa$ is significantly greater than zero, then we can make an optimization of the algorithm by not bothering to accumulate $\bar{u}$ on any but the final invocation of Step 7 (analogous to throwing away all but the final forward iteration in Step 3.)

Alternatively, we can use the convergence of $\bar{u}$ as the stopping criterion in Step 8, without estimating or interpolating values for τ and κ, but this heuristic is dangerous because it cannot guarantee convergence from an arbitrary start point as the following example shows.

Example 3.2

$$y \in R^3, u \in R, \Phi(y_i, y_j, y_k, u) = \left(\frac{1}{2}y_j, \frac{1}{2}y_k, 16u\right), y_*(u) = 4u$$

$$r = e_i, r\frac{\partial y_*}{\partial u} = 4, \text{ but } r\Phi_u = 0, (r + r\Phi_y)\Phi_u = 0$$

which falsely appears to have converged (although the next iteration gives the correct value.)

QED

It is important to note that the constructor Φ need not be used in Step 2 of Algorithm 3.1 to find the fixed point y_*. The fixed point may be constructed by any means at all, even using non-differentiable operations, so long as a single iteration of a well-behaved constructor is included at the end in Step 3.

There are a number of potential sources of *truncation* error in Algorithm 3.1. In Step 2 we may have truncated iteration of the operator Φ before y has converged to y_*. This will mean that the value of z will be slightly perturbed, which in turn will have an effect upon r and hence upon $\bar{u}$ and $\bar{x}$. Provided we can quantify the error in y, the propagation effects of these perturbations can be analysed in the same way as rounding error.

Also, the values of $\zeta\Phi_y$ and $\zeta\Phi_u$ will be repeatedly evaluated at perturbed y-values in Step 7, and this will also affect $\bar{u}$ and hence $\bar{x}$. However, the crucial point is that (by Theorem 2.3) the adjoint constructor for $\bar{u}$ is Lipschitz in y, and so the error growth is contained.

A second truncation effect arises in Step 8, if we terminate the reverse fixed point iteration before $\bar{u}$ has converged. Again, this will affect the values calculated for $\bar{x}$, in a fashion similar to the propagation of rounding error.

We investigate further the effects of these errors and their effect upon setting an appropriate stopping criterion in the following sections.

We conclude this section by noting that an implementation of an algorithm essentially equivalent to Algorithm 3.1 is described in [5].

4. Error Estimates for Function Values. We begin by investigating the effect of truncation in Step 2 in Algorithm 3.1 upon the calculated value for the target function, and the interaction of this with the propagation of roundoff error.

Lemma 4.1 Let z be a dependent scalar variable produced by a sequence of elementary operations, which includes a subsequence representing a fixed point constructor Φ for the intermediate variables $y = y_*(u)$, just as in Algorithm 3.1. As usual, we assume Φ is a well-behaved iterative constructor for y_* with attraction constant τ.

Then an approximate worst case forward truncation error bound, ie a measure of the maximum impact on the value of z of the fact that $y_{initial} \neq y_*(u)$, is given by

$$\frac{\tau}{1-\tau}\|r\|\|y_{final} - y_{initial}\|$$

where $r = \partial z/\partial y$ is calculated just as in Algorithm 3.1.

Proof: From Theorem 2.2 we have

$$(1-\tau)\|y_{initial} - y_*\| \leq \|y_{final} - y_{initial}\|$$

Now (to first order) the effect on z of a change in y from $y_{initial}$ to y_* is at most

$$\delta_y^z \leq \|(\nabla_y\ f(x, \Phi(y_{initial}, u))(y_{initial} - y_*)\| \leq \|r\|\|\Phi_y(y_{initial}, u)\|\|y_{initial} - y_*\|$$

$$\leq \tau\|r\|\frac{\|y_{final} - y_{initial}\|}{(1-\tau)}$$

QED.

An important application of reverse accumulation is in the analysis of roundoff error. The conventional method of reverse accumulation allows an analysis of the roundoff error in a dependent variable to be made automatically. We summarize these results in the following.

Proposition 4.2 Let z be a scalar dependent variable produced by a sequence of elementary operations, indexed by i. (We assume for convenience that the indexing includes the independent variables at the beginning and z itself at the end.) For each node v_i in the computational graph of the calculation for z, let $\bar{v}_i = \partial z/\partial v_i$ be the corresponding

adjoint quantity calculated by reverse accumulation, and let δ_i be an upper bound on the rounding error in v_i introduced by the elementary operation i which produced v_i.

Then an approximate worst case upper bound on the rounding error for z is given by the quantity $e = \sum_i \delta_i \cdot |\bar{v}_i|$.

Proof: Omitted. For further details see [2, §7], [10] and the references cited there.

We now show how to extend Lemma 4.1 in the style of Proposition 4.2 so as to provide a combined analysis of rounding and forward truncation error where the computation of z involves a fixed point construction.

Lemma 4.3 Under the conditions of Lemma 4.1, assume that we have available an accurate upper bound $\hat{\eta}$ for $\|\hat{y}_{final} - y_{initial}\|$ where $\hat{y}_{final}$ denotes the true value (with no rounding error) of $\Phi(y_{initial}, u)$. Then an approximate worst case error bound for the entire calculation of z, including both rounding and truncation error, is given by $e + e_*$ where e is calculated as in Proposition 4.2, including the graph for Φ but treating the values for $y_{initial}$ as exact, and

$$e_* = \frac{\tau\hat{\eta}}{1-\tau} \, \|r\|$$

Proof: In Lemma 4.1 we ignored rounding error. Replacing y_{final} by $\hat{y}_{final}$ in Lemma 4.1 gives the truncation error bound e_*. This accounts (to first order) for the effect on z of the fact that $y_{initial} \neq y_*(u)$. The rounding errors (including those for Φ on y_{final}) are contained in e by Proposition 4.2.

QED.

If the norm $\|.\|$ is differentiable (for example if we use the Euclidean norm $\|.\|_2$), then we can use Proposition 4.2 recursively to develop an estimate for $\hat{\eta}$ as follows.

Algorithm 4.4 Under the conditions of Lemma 4.3, and assuming that $\|.\|$ is differentiable, proceed as follows:
Step 1. Augment the graph for Φ by adding an explicit calculation of $\eta = \|y_{final} - y_{initial}\|$.
Step 2. Use Proposition 4.2 (recursively) to perform an automatic rounding error analysis for η on the (augmented) graph for Φ. For the purpose of this analysis, regard the values for $y_{initial}$ as exact.
Step 3. Add the error estimate e_η from Step 2 to the calculated value for η from Step 3 to obtain $\hat{\eta} = \eta + e_\eta$.
Now $\hat{\eta}$ is an estimated upper bound for $\|\hat{y}_{final} - y_{initial}\|$.

See also [11, §6].

5. Setting the Stopping Criterion. Once we have a value for $\hat{\eta}$ for use in Lemma 4.3, we can also use it to estimate on the truncation errors introduced by Algorithm 3.1 in

the calculated values $\bar{u}$ for $r\ \nabla_u\ y_*(u)$, and hence to provide a stopping criterion for the forward and reverse passes of Algorithm 3.1.

Lemma 5.1 The *forward truncation* error in the calculated value of $\bar{u}$ due to the difference between $y_{initial}$ and $y_*(u)$ at the end of Step 2 is at most

$$C\frac{1-\tau+\kappa}{(1-\tau)^3}\|r\|\|\hat{y}_{final}-y_{initial}\|$$

where $\hat{y}_{final}$ denotes the true value (with no rounding error) of $\Phi(y_{initial},u)$.

The *reverse truncation* error in $\bar{u}$ due to stopping after a finite number of passes through Step 7 is at most

$$\frac{\kappa}{1-\tau}\|\bar{y}_{initial}+r-\bar{y}_{final}\|$$

Proof: The first part is a direct consequence of Theorem 2.2. The second part follows from Theorem 2.3.

QED.

Now we can determine the optimal stopping criterion in the light of this error analysis. If the desired accuracy is that $\|\nabla_u\ r\cdot y_*(u)-\bar{u}\|<\xi\|r\|$, where $\xi<1$, then it suffices to ensure that

$$\|y_{final}-y_{initial}\|<\frac{\xi(1-\tau)^3}{2C(1-\tau+\kappa)}\qquad\text{and}\qquad\frac{\|\bar{y}_{initial}+r-\bar{y}_{final}\|}{\|r\|}<\frac{\xi(1-\tau)}{2\kappa}$$

where τ,κ respectively are bounds for $\|\Phi_y\|,\|\Phi_u\|$ in a neighbourhood of $(y_*(u),u)$ including $(y_{initial},u)$, and C is the Lipschitz constant for the map $\Phi'=(\Phi_y|\Phi_u)$ in this neighbourhood. Note that after n reverse iterations we are guaranteed to have $\|\bar{y}_{initial}+r-\bar{y}_{final}\|\leq\tau^n\|r\|$.

The criterion and estimates given here should be compared with the convergence characteristics of forward accumulation, for which

$$\|{y'}_k-{y'}_*\|\leq\frac{\|{y'}_k-{y'}_{k+1}\|}{1-\tau}+\frac{\|y_k-y_{k+1}\|}{(1-\tau)^2}C(1+\|{y'}_k\|)$$

For further details the reader is referred to the comprehensive analysis of the forward case given in [9].

In our analysis we have treated r as if it were a constant, and taken no account of the effect of truncation errors in y_{final} on r. (This corresponds to the assumption in the forward case that the values of u' are correct.) These effects can be subjected to essentially the same analysis as for rounding errors in gradient propagation [2] , given appropriate knowledge of the Lipschitz constants for f' (respectively u' in the forward case.)

The criterion given here requires upper bounds to be estimated for the Lipschitz constant C as well as for τ and κ. In many cases, in particular optimal control problems involving the solution of ODE's, techniques such as those described in [8] and [4] combined with suitable preaccumulation strategies [1, §6] allows efficient direct evaluation of the

required constants under a weighted ∞-norm. This is of particular utility in the case of quadratic convergence where second derivatives are used in the construction of the fixed point. We do not pursue this approach further here.

However, it is rare for a function to be evaluated only once in a neighbourhood of interest. Since the computational graph of the target function z is available, an alternative approach is to estimate the constants we require from measured convergence behaviour, and to refine our estimates for convergence parameters as we proceed. We briefly describe this approach in the next section.

6. Self-adapting Code. The development so far can be summarized as follows. A fixed point construction contained in an algorithm is coded (either by the programmer, or by a preprocessing tool, or by a purpose built compile-time environment) as follows:

$y := fix(\Phi, y_0, u, norm, normadj, \xi)$

This is transformed into the following code on the forward pass

$mark\_graph(begin\_fix, \Phi, \xi)$
$declare\ u_1 := make\_copy(u)$
$declare\ y_i := make\_copy(y_0)$
$graph\_build(turn\_off)$
$declare\ y_f := \Phi(y_i, u_1)$
$declare\ \eta := norm(y_f - y_i)$
$mark\_graph(end\_variables, y_i, u_1)$
$while\ 2\eta C(1 - \tau + \kappa) > \xi(1 - \tau)^3\ do$
    $y_i := y_f$
    $y_f := \Phi(y_i, u_1)$
    $\eta := norm(y_f - y_i)$
$end\ do$
$graph\_build(turn\_on)$
$y_f := \Phi(y_i, u)$
$\eta := norm(y_f - y_i)$
$mark\_graph(end\_fix, y_f, \eta)$
$y := make\_copy(y_f)$

In addition, the following *adjoint* code is inserted on the reverse pass:

$in\ case\ graphnode.type = end\_fix:$
    $r := \bar{y}$
    $\bar{y}_f := r$
    $\bar{y}_i := 0$
    $\eta := normadj(\bar{y}_i + r - \bar{y}_f)/normadj(r)$
    $while\ 2\eta\kappa > \xi(1 - \tau)\ do$
        $\bar{y}_f := \bar{y}_i + r$
        $\bar{y}_i := 0$
        $\bar{u}_1 := 0$

$reverse\_accumulate(end\_fix, end\_variables)$
$\eta := normadj(\bar{y}_i + r - \bar{y}_f)/normadj(r)$
end do
$\bar{u} := \bar{u}_1$

In this pseudocode, *mark_graph* is a function which returns a pointer to the (current) end-of-graph position. This pointer points at a special graph node created to record the mark. The *reverse_accumulate* routine accumulates adjoint values for independent variables in a sparse vector (of pointers to graph nodes) associated with the marked node.

To the forward code may be added code to reverse accumulate the rounding error in η (Algorithm 4.4), or to pre-accumulate estimates for τ and κ [1, §6], or even to interpolate estimates for the Lipschitz constant C.

Following Theorem 2.3, the adjoint code could itself be represented in the form of a fixed point construction for ζ using $\bar{\Phi}(\zeta, y, u) = r + \zeta\Phi_y(y, u)$ followed by the step $\bar{u} = \zeta\Phi_u(y, u)$, where the derivative multiplications are performed by nested calls to *reverse_accumulate*. This would allow the techniques of [3] to be applied to obtain values for higher derivatives.

Once the graph containing the fixed point construction has been built, the fixed point can be automatically re-constructed with a finer tolerance ξ. A similar tightening technique to that suggested here is described in detail in [12]. If the function (and gradient) calculation is part of an iterative algorithm, for instance solving an optimization or optimal control problem, then this allows the tolerance to be adaptively tightened (using adjoint values from previous outer iterations) as the algorithm nears a solution, thus avoiding unnecessary inner iterations of the fixed point constructors in the early outer stages. This is especially significant when such constructors are nested. In particular, forward and adjoint values from previous function evaluations at nearby points can be used as starting points for the forward and reverse fixed point iterations, and in order to contain the effects of forward truncation of inner iterations on the value of the target function to a level which is acceptable in the light of the improvement sought.

Once a trial solution has been reached, the calculation can be automatically re-executed to check its own tolerances, by reverse accumulating rounding error estimates to check that the chosen values for ξ are satisfactory and that the fixed point iterations have converged to the limit of machine accuracy for the given constructor. This technique can be combined with exact interval arithmetic (see for example [2]) to validate the estimates for τ, κ, C, and to provide tight interval error bounds on the solution, but this requires more analytical machinery than we have developed here.

7. Conclusion. We have considered the problem of applying reverse accumulation to obtain automatic gradients and error estimates for objective functions which include in their construction the use of iterative fixed point constructors. We suggest an implementation approach which would allow this to be done by a fairly routine extension of existing reverse accumulation code. It is known that, under fairly mild assumptions on the fixed point constructor Φ (namely that, in a neighbourhood of the fixed point, $\Phi_y \times \Phi_u$ is Lipschitz and $\|\Phi_y\| < 1$), if the fixed point construction converges to the correct value, then the reverse gradient construction converges to the correct value also (see [6], [11, §6]). We

have shown here how to re-use the graph for the constructor Φ so as to set explicit stopping criteria for the forward and reverse passes based on the gradient accuracy required. Our construction allows the gradient values to be obtained to the same order of accuracy as the objective function values, which is in general the best we can hope for. Indeed, our construction shows that the forward and the reverse iteration passes converge at the same rate.

The technique described here, applied in conjunction with a run-time stack of pointers to the computational graph, allows several fixed point constructions to be used either serially or hierarchically in any combination when defining the objective function values, as for example when solving numerically a system of differential equations by Picard iteration. The appropriate gradients will be extracted automatically.

References.

[1] Bruce Christianson, 1992, Automatic Hessians by Reverse Accumulation, IMA Journal of Numerical Analysis **12**, 135–150

[2] Bruce Christianson, 1992, Reverse Accumulation and Accurate Rounding Error Estimates for Taylor Series Coefficients, Optimization Methods and Software **1**(1), 81–94

[3] Bruce Christianson, 1993, Reverse Accumulation of Functions containing Gradients, Theory Institute on Combinatorial Challenges in Automatic Differentiation, Argonne National Laboratories, Illinois *(extended abstract)*; *also* Technical Report 278, Numerical Optimisation Centre, University of Hertfordshire, England Europe *(full text)*

[4] Bruce Christianson and Laurence Dixon, 1992 Reverse Accumulation of Jacobians and Optimal Control, Technical Report 267, Numerical Optimisation Center, University of Hertfordshire, England Europe

[5] Ralf Giering, 1993, Adjoint Model Compiler, Manual Version 0.4, Max Planck Institute fur Meteorologie, Hamburg, Germany

[6] Jean Charles Gilbert, 1992, Automatic Differentiation and Iterative Processes, Optimization Methods and Software **1**(1), 13–21

[7] Andreas Griewank, 1989, On Automatic Differentiation, *in* Mathematical Programming 88, Kluwer Academic Publishers, Japan

[8] Andreas Griewank, 1993, Some Bounds on the Complexity of Gradients, Jacobians and Hessians *in* Complexity in Numerical Optimization, *ed* P.M. Pardolos, World Scientific

[9] Andreas Griewank *et al*, 1993, Derivative Convergence for Iterative Equation Solvers, Optimization Methods and Software, *to appear*

[10] Masao Iri, 1991, History of Automatic Differentiation and Rounding Error Estimation, *in* Automatic Differentiation of Algorithms, SIAM, Philadelphia

[11] Gershon Kedem, 1980, Automatic Differentiation of Computer Programs, ACM Transactions on Mathematical Software **6**(2), 150-165

[12] E.A. Musaev, 1992, Wave Computations: A Technique for Optimal Quasi-concurrent Self-validation, Interval Computations **1**(3), 53–60