Calculation of first and second derivatives of log-likelihood using *automatic differentiation*.

Robert Davies, Statistics Research Associates

This note follows up a comment I made at the first HMMCS workshop and also at Walter Zucchini's course on HMMs in Sydney.

Suppose you can write a program to calculate an exact log-likelihood. Then an *automatic differentiation* program enables you to convert this to a program for calculating the first and second derivatives of the log-likelihood with respect to your unknown parameters.

The first and second derivatives are useful for

- finding the variances of the maximum likelihood estimates of the unknown parameters;
- for use by a program maximising the likelihood;
- possibly, as part of the derivation of a test for testing, say, two switching levels versus three.

The principle of automatic differentiation is to define objects that consist of a value that depends on the unknown parameters, together with its derivatives. Then define the elementary mathematical operations on these objects following the rules of differentiation so that, for example,

$$\left\{X, \frac{\partial X}{\partial \theta}\right\} + \left\{Y, \frac{\partial Y}{\partial \theta}\right\} \text{ is calculated as } \left\{X + Y, \frac{\partial X}{\partial \theta} + \frac{\partial Y}{\partial \theta}\right\}$$

$$\left\{X, \frac{\partial X}{\partial \theta}\right\} \times \left\{Y, \frac{\partial Y}{\partial \theta}\right\} \text{ is calculated as } \left\{XY, \frac{\partial X}{\partial \theta} Y + X \frac{\partial Y}{\partial \theta}\right\}$$

$$\exp\left\{X, \frac{\partial X}{\partial \theta}\right\} \text{ is calculated as } \left\{\exp(X), \frac{\partial X}{\partial \theta} \exp(X)\right\} .$$

If you want second derivatives, you do the same kind of thing but also include the second derivatives.

One then writes one program to carry out the log-likelihood calculation but applies it to these objects rather than ordinary real numbers. Then, as well as getting the likelihood, you get the derivatives. And, if your likelihood calculation is correct, your derivatives are also likely to be correct.

The basic reference for automatic differentiation is

Griewank, Andreas (2000) *Evaluating derivatives: principles and techniques of algorithmic differentiation.* SIAM, Philadelphia.

A number of programs exist for carrying out automatic differentiation. I have written one in C++ and it is available on my website http://www.robertnz.com. C++ has the necessary structure for defining the objects and the operations on them.

This is what the code looks like for the calculation of the log-likelihood and its derivatives of a hidden Markov model where we have normally distributed data with the mean and variance depending on the state of the Markov process.

```
// log likelihood calculator

template <class T>
T log_likelihood0(
   const Array<Real>& X,          // the data
   const Array<T>& mu,            // the means
   const Array<T>& sigma,         // the sds
   const Array<T>& delta,         // the initial distribution
   const Array2<T>& gamma)        // the transition matrix
{
   // get dimensions
   int x_first = X.first(), x_last = X.last();
   int m_first = mu.first(), m_last = mu.last();
   // ... stuff for checking array starting and finishing
   // points agree has been deleted

   // initialise
   Array<T> phi = delta;
   T llk(0);

   // now do it
   for (int t = x_first; t <= x_last; ++t)
   {
      // temporary stuff
      Array<T> phi1; Array<T> P(m_first, m_last);

      // make P diagonal matrix
      for (int i = m_first; i <= m_last; ++i)
      {
         T d = (X(t) - mu(i)) / sigma(i);
         P(i) = exp(-0.5 * d * d) / sigma(i);
      }
      if (t == x_first) phi1 = phi;
      else matrix_multiply(phi, gamma, phi1);
      diagonal_matrix_multiply(phi1, P, phi);
      T sum_phi = phi.Sum();
      llk += log(sum_phi);
      phi /= sum_phi;
   }
   // return it
   return llk;
}
```

It uses my *automatic differentiation* package which is on my website and an *array* package which is not yet on my website. `Array<T>` defines a one dimensional array of elements of type `T` and `Array2<T>` defines a two dimensional array. You can run the program either with element type `T` as ordinary real variables or with values and derivatives or with values and first and second derivatives depending on whether you want just the value or the value with first or second derivatives.

There is some additional code to set up the objects with first or second derivatives. This is slightly complicated since we need to use parameters that are not subject to the linear equality constraints and I want to avoid the inverse you need to find the stationary probabilities. But it is all quite straightforward. And the automatic differentiation package can work out the derivatives needed to convert the variances of these transformed parameters to the variances of the parameters we are actually interested in.

The calculation with derivatives runs a lot slower than the simple calculation of the value alone. But it is fast enough for practical use. Here are some sample times for a single likelihood calculation on a 3 GHz Pentium 4 computer. I am using a Markov process with three states and have 10,000 observations.

Operation	Time (seconds)
Value only	0.02
Value and first derivatives	0.58
Value and first derivatives – reverse algorithm	0.14
Value and first and second derivatives	3.53

The reverse algorithm for first derivatives is a slightly more efficient method for first derivatives at the expense of being slightly more awkward to set up and having more memory requirements.