Moving Averages of Solutions of ODEs

L.F. Shampine
Mathematics Department
Southern Methodist University
Dallas, TX 75275
lshampin@mail.smu.edu

S. Thompson
Department of Mathematics & Statistics
Radford University
Radford, VA 24142
thompson@radford.edu

March 6, 2006

Abstract Moving averages of the solution of an initial value problem for a system of ordinary differential equations are used to extract the general behavior of the solution without following it in detail. They can be computed directly by solving delay differential equations. Because they vary much less rapidly and are smoother, they are easier to compute.

1 Introduction

We are interested in the general behavior of a solution of an initial value problem (IVP) for a first order system of ordinary differential equations (ODEs),

\[ y' = f(t, y), \quad t_0 \leq t \leq t_f \]  

(1)

with initial values

\[ y(t_0) = y_0 \]  

(2)

In the first instance we have in mind a solution that can be viewed as a slowly varying function with a rapid oscillation of small magnitude superimposed. A natural approach to following the slow change of such a solution is to compute a moving average \( s(t) \). We study how to compute the moving average directly and in particular, how to do this using step sizes appropriate to \( s(t) \) instead of the rapidly varying \( y(t) \). We are also interested in solutions that oscillate rapidly with a slowly varying amplitude. We study how to assess the amplitude of a solution by computing directly an RMS moving average \( R(t) \). With either kind
of average, we shall see that averaging a second time can provide a solution with much slower variation at little additional cost. Though \( y(t) \) is defined by a system of ODEs, we compute the averages by solving systems of delay differential equations (DDEs).

## 2 Moving Average

For \( \Delta > 0 \), the moving average

\[
s(t) = \frac{1}{\Delta} \int_{t-\Delta}^{t} y(\tau) d\tau, \quad t_{0} + \Delta \leq t
\]  

(3)

is a natural way of forming an averaged approximation to \( y(t - \Delta/2) \). Obviously \( s(t) \) is smoother in the sense that it has one more continuous derivative than \( y(t - \Delta/2) \). It is also smoother in the sense that the moving average suppresses changes in \( y(t - \Delta/2) \) that occur on a time scale smaller than \( \Delta \). A couple of examples will clarify the effects of averaging.

First we consider how the moving average affects the oscillatory function \( y(t) = \exp(i\omega t) \). A little calculation shows that

\[
s(t) = \left( \frac{\sin(\omega \Delta/2)}{\omega \Delta/2} \right) y(t - \Delta/2) = \text{sinc}(\omega \Delta/2) y(t - \Delta/2)
\]

We see that as \( \omega \Delta \rightarrow 0 \), the moving average \( s(t) \rightarrow y(t - \Delta/2) \), hence low frequency oscillations are not affected much by forming a moving average. On the other hand, \( \text{sinc}(\omega \Delta/2) \) is \( O(1/(\omega \Delta)) \) as \( \omega \Delta \rightarrow +\infty \), so high frequency oscillations are damped and roughly speaking, the higher the frequency, the more the oscillation is damped. The upper curve of Figure 1 shows precisely how the damping factor \( |\text{sinc}(\omega \Delta/2)| \) depends on \( \omega \Delta \). To see how averaging affects discontinuous functions, consider the Heaviside function

\[
H(t) = \begin{cases} 
0 & \text{when } t \leq 0, \\
1 & \text{when } 0 < t.
\end{cases}
\]

A little calculation shows that the moving average with interval \( \Delta \) is

\[
s(t) = \begin{cases} 
0 & \text{when } t \leq 0, \\
t & \text{when } 0 < t \leq \Delta, \\
1 & \text{when } \Delta < t.
\end{cases}
\]

That is, the jump discontinuity is made continuous and stretched out over an interval of length \( \Delta \).

Differentiation of the definition (3) shows that the moving average satisfies the delay differential equation

\[
s'(t) = \frac{1}{\Delta} [y(t) - y(t - \Delta)], \quad t_{0} + \Delta \leq t
\]  

(4)
Our plan is to use a program for solving DDEs to compute the moving average for \( t_0 + \Delta \leq t \) by solving (1) and (4) simultaneously. This can be done efficiently by controlling the error with respect to \( s(t) \) rather than \( y(t) \).

It is possible to compute the moving average by solving a system of ODEs. For this we introduce a new variable \( w(t) = y(t - \Delta) \) that we can compute by integrating \( w'(t) = f(t - \Delta, w(t)) \). Substituting \( w(t) \) into (4), we obtain a system of ODEs for \( y(t), s(t), w(t) \). It is illuminating that the system of DDEs we use to compute the moving average is equivalent to a system of ODEs, but when we implemented the approach, we found it to be quite unsatisfactory. The difficulty is that in principle \( y(t) \equiv w(t + \Delta) \), but the values computed are different because the step sizes are different in the two integrations.

3 Getting Started

A convenient way to start the integration of (1), (4) at \( t_0 + \Delta \) is to define an averaged solution \( s(t) \) for the initial interval \([t_0, t_0 + \Delta]\) by

\[
s(t) = \frac{1}{t - t_0} \int_{t_0}^{t} y(\tau) \, d\tau, \quad t_0 \leq t \leq t_0 + \Delta
\]  

(5)

This is an average of the solution in the initial interval and clearly results in the value we want at \( t = t_0 + \Delta \). Differentiation of \((t - t_0)s(t)\) shows that \( s(t) \) satisfies the ODE

\[
s'(t) = \frac{1}{t - t_0} [y(t) - s(t)]
\]  

(6)

It is easy enough to solve this ODE along with equation (1), but we must be careful at the initial point where the differential equation is indeterminate as
written. Taking limits we find that

\[ s(t_0) = y(t_0) = y_0 \quad (7) \]
\[ s'(t_0) = \frac{1}{2} y'(t_0) = \frac{1}{2} f(t_0, y_0) \quad (8) \]

This gives us the initial value we need for \( s(t) \) and the value of the first derivative to be used when evaluating equation (6) at \( t_0 \).

4 If averaging once is good, . . .

It is natural to consider forming the moving average of \( s(t) \),

\[ \bar{s}(t) = \frac{1}{\Delta} \int_{t-\Delta}^{t} s(\tau) \, d\tau, \quad t_0 + \Delta \leq t \quad (9) \]

With this definition, \( \bar{s}(t) \approx y(t - \Delta) \) for \( t_0 + 1.5\Delta \leq t \). Differentiation shows that \( \bar{s}(t) \) satisfies the DDE

\[ \bar{s}'(t) = \frac{1}{\Delta} [s(t) - s(t - \Delta)], \quad t_0 + \Delta \leq t \quad (10) \]

In the first part of the interval we define

\[ \bar{s}(t) = \frac{1}{t - t_0} \int_{t_0}^{t} s(\tau) \, d\tau, \quad t_0 \leq t \leq t_0 + \Delta \quad (11) \]

which we can differentiate to obtain the ODE

\[ \bar{s}'(t) = \frac{1}{t - t_0} [s(t) - \bar{s}(t)] \quad (12) \]

As with the first average, taking limits shows that

\[ \bar{s}(t_0) = y(t_0) = y_0 \quad (13) \]
\[ \bar{s}'(t_0) = \frac{1}{4} y'(t_0) = \frac{1}{4} f(t_0, y_0) \quad (14) \]

This gives us the initial value we need for \( \bar{s}(t) \) and the value of the first derivative to be used when evaluating equation (12) at \( t_0 \).

We see that it is little trouble to augment the differential equations for \( y(t) \) and \( s(t) \) so as to compute \( \bar{s}(t) \). For \( t_0 + 1.5\Delta \leq t \), the effect of averaging \( y(t) = \exp(i\omega t) \) twice is easily found to be \( \bar{s}(t) = \text{sinc}^2(\omega\Delta/2) y(t - \Delta) \). The lower curve of Figure 1 shows that a second averaging provides a much stronger damping of high frequency oscillations. This function is also smoother, making it easier to integrate.
5 Moving RMS Average

In our discussion of the moving average we had in mind a solution that can be viewed as a slowly varying function with small, fast changes superimposed. There is another kind of multiple-scale problem that can be discussed in a similar way. To be concrete, we consider a problem analyzed by Holmes [2] using asymptotic methods. A forced Duffing equation

$$y'' + \epsilon \lambda y' + y + \epsilon \kappa y^3 = \epsilon \cos(1 + \epsilon \omega)t$$  \hspace{1cm} (15)

is solved with initial conditions $y(0) = y'(0) = 0$. Figure 3.5 of [2] presents a numerical solution when $\epsilon = 0.05, \lambda = 2, \kappa = 4, \omega = 3/8$ along with a numerical solution of an ODE for the amplitude function obtained by a multiple-scales approximation. On the interval $[0, 250]$ the solution has many oscillations, but its amplitude increases smoothly from zero to a value that is nearly constant. For such a solution the amplitude is more informative than a moving average.

As a measure of the amplitude of a solution $y(t)$, we use the moving root-mean-square (RMS) average, $R(t) = \sqrt{A(t)}$, with

$$A(t) = \frac{1}{\Delta} \int_{t - \Delta}^{t} y^2(\tau) \, d\tau, \quad t_0 + \Delta \leq t$$  \hspace{1cm} (16)

(For systems the integrand $y^2(t)$ and similar expressions are interpreted componentwise.) It is computationally convenient to compute $A(t)$ and then simply take the square root to get $R(t)$ as needed. We can proceed very much as we did with $s(t)$. First we define

$$A(t) = \frac{1}{t - t_0} \int_{t_0}^{t} y^2(\tau) \, d\tau, \quad t_0 \leq t \leq t_0 + \Delta$$  \hspace{1cm} (17)

and compute it on this interval by solving the IVP

$$A'(t) = \frac{1}{t - t_0} \left[ y^2(t) - A(t) \right]$$  \hspace{1cm} (18)

with initial value

$$A(t_0) = y_0^2$$  \hspace{1cm} (19)

along with the IVP for $y(t)$. At the initial point the indeterminate form (18) has the value

$$A'(t_0) = y_0 f(t_0, y_0)$$  \hspace{1cm} (20)

Thereafter, differentiation of (16) shows that

$$A'(t) = \frac{1}{\Delta} \left[ y^2(t) - y^2(t - \Delta) \right], \quad t_0 + \Delta \leq t$$  \hspace{1cm} (21)

Clearly it is as easy to compute $A(t)$ as $s(t)$ and similarly we can compute a second average that we denote as $\bar{A}(t)$. 


6 Computational Issues

We have written a solver, ODEAVG, in Fortran 90 (F90) based upon the DDE solver DDE_SOLVER [5]. We have also written a solver, odeavg, in MATLAB based upon the DDE solver dde23 [4]. The user interfaces are quite similar, but there are differences in detail and in the languages that led us to proceed somewhat differently for the two solvers.

We compute $s(t)$ by solving a pair of DDEs with one delay that is constant, namely $\Delta$. Some programs for solving DDEs numerically limit the step size to the shortest delay, but for the present application it is crucial that the solver use step sizes bigger than $\Delta$ whenever the smoothness of the solution warrants it. Both dde23 and DDE_SOLVER have this capability. On $[t_0, t_0 + \Delta]$ the equations are (1) and (6). There is no solution history, rather initial values (2) and (7), because the equations are ODEs on this interval. If we code equation (6) so as to use the proper value (8) at $t = t_0$, integration of the system is straightforward. For $t > t_0 + \Delta$, the equations are (1) and (4). Clearly we need to handle carefully the change in definition and the lack of smoothness in the solution at $t_0 + \Delta$. This is accomplished easily by means of the capability dde23 has for restarting an integration. DDE_SOLVER has an equivalent capability, but it is realized in a quite different way.

A key issue is to control the error according to the averaged solution rather than the solution of the ODE itself. This raises an issue of software design because neither of the DDE solvers allows a vector relative error tolerance, though they do allow a vector absolute error tolerance. We added this capability to the solvers, but before discussing how we did that, we explain why we needed the capability. Because the averaged solutions approximate $y(t)$, it is reasonable to apply tolerances specified for $y(t)$ to the desired moving average. Of course we use relaxed tolerances for the other quantities. We tried controlling the error in the desired average only, but we quickly found that it is necessary to insist on some accuracy in $y(t)$. After some experimentation we settled on the following: The user specifies a scalar relative error tolerance $rtol$ or accepts the default value. This tolerance is imposed on the desired average and the minimum of 0.05 and 10$rtol$ is imposed on the other components of the system. The user also specifies an absolute error tolerance $atol$ which can be a vector, though the default is a scalar. Again this tolerance is imposed on the desired average and 100$atol$ is imposed on the other components.

Though DDE_SOLVER does not allow a vector relative error control, a lower level subroutine does, so modifying the solver was very easy. it was also easy to modify the error control of dde23 to obtain a function vrdde23 that allows a vector of relative error tolerances. The situation with dde23 is complicated by the fact that it is embedded in the MATLAB problem solving environment (PSE). dde23 uses auxiliary functions ntrp3h and odefinalize that are located in a private subdirectory. Naturally the modified program must also have access to these functions. This can be accomplished by placing copies of the functions in the directory where vrdde23 is located or placing vrdde23 in the directory where dde23 is located. Fortunately, it was not necessary to modify any of the
auxiliary functions that accompany dde23. The moving average is computed with a solver odeavg that should resemble the IVP solvers of the PSE, not the DDE solvers. Error tolerances are set for the IVP solvers with the auxiliary function odeset. The corresponding function for the DDE solvers is ddeset. It was not necessary to alter odeset for use with odeavg. And, the option structure formed by this function is compatible with ddeset, so it was easy to alter the options inside odeavg so as to deal with tolerances for the larger system of DDEs and call vrdde23.

Both DDE_SOLVER and dde23 return solutions in the form of a structure and values of the solution are obtained with an auxiliary function. That is an option for the IVP solvers of MATLAB, but the only mode of output from its DDE and boundary value problem solvers. In MATLAB it is typical that users receive the output of an IVP solver in the form of a mesh and values of the solution on the mesh. Certainly it would be possible to write odeavg so as to return output in this form, but it seemed more natural simply to return a solution structure. The auxiliary functions used to evaluate a solution returned as a structure required no alteration to apply to the output of odeavg (ODEAVG).

A somewhat awkward issue for the design of software is that the moving average \( s(t) \) is defined in the usual way only for \( t_0 + \Delta \leq t \). Though we extended the definition to \( [t_0, t_0 + \Delta] \), returning values for this interval muddies interpretation of the numerical solution. It is also awkward that we want an averaged approximation to \( y(\tau) \), but the moving average is delayed, so the approximation to this quantity is \( s(\tau + \Delta/2) \). Our solvers compute, say, \( s(t) \) as described earlier, but the approximation on \( [t_0, t_0 + \Delta] \) is dropped and the independent variable is shifted so that if \( \text{sol} \) is the solution structure returned by odeavg, the value computed by \( \text{deval}(\text{sol}, \tau) \) approximates \( y(\tau) \) and similarly for ODEAVG. This seems natural, but the very definition of the moving average means that an averaged solution is not defined in the first and last parts of \( [t_0, tf] \). Averaging twice is handled similarly.

It was convenient to use programs that solve rather general DDEs, but there is a price for this. For instance, these solvers provide for delays that reach all the way back to the initial point. The special DDEs that arise in computing moving averages do not require the ability to evaluate a solution at times further back than \( \Delta \). Accordingly, it would be possible to write a program to compute moving averages that required much less storage than our solvers.

The call lists for the F90 subroutine ODEAVG and the MATLAB function odeavg are very similar. Indeed, they differ only in that they respect the underlying solvers and languages. A typical call to the F90 subroutine has the form

```fortran
SOL = ODEAVG(AVG_OPT, DELTA, NVAR, ODES, Y0, TSPAN, OPTIONS=OPTS)
```

and a typical call to the MATLAB function has the form

```matlab
sol = odeavg(avg_opt, delta, ode, tspan, y0, options)
```
The only arguments that require comment is DELTA (delta) which is the interval $\Delta$ and AVG_OPT (avg_opt). The latter is a vector of two components. The first indicates the number of times the solution is to be averaged, either 1 or 2. The second indicates the kind of average with 1 indicating a moving average and 2 indicating an RMS average. It is to be appreciated that in the case of an RMS average, the solvers compute and return the square of the average, say $A(t)$, so the user must take the square root to get $R(t)$.

7 Numerical Experiments

The F90 subroutine ODEAVG and the MATLAB function odeavg are available from the website http://www.radford.edu/~thompson/ffddes/averages. Available also from the website are test and demonstration programs for each of the two codes. To study the effect of averaging for an important case, we solved systems corresponding to a linear oscillator. Though complicated, it is straightforward to work out analytical expressions for the various averages when solving

$$y''(t) = -\omega_0^2 y(t) + a \cos(\omega t), \quad y(0) = \alpha, \quad y'(0) = \beta$$

We used these expressions to verify that our solvers were performing in a satisfactory way.

To illustrate the effects that high frequency oscillations of small magnitude can have on the cost of an integration, we solved the system

$$y_1'(t) = 5y_2(t), \quad y_2'(t) = -5y_1(t)$$
with initial conditions $y_1(0) = 0, y_2(0) = 1$ on $[0, 25]$ and the same IVP with the first equation changed to $y_1'(t) = 5y_2(t) + 10^{-6}\sin(1000t)$. With absolute and relative error tolerances of $10^{-8}$, the F90 solver required 28,883 derivative evaluations to solve the first IVP and 33,948 to solve the second. With $\Delta = 5$, only 5904 evaluations were required to compute the once averaged solution for the second IVP and 5274 evaluations to compute the twice averaged solution. The averaged solutions exhibited the correct long term behavior.

Borelli and Coleman [1] discuss a chemical reaction that exhibits unexpected autocatalytic oscillations. The defining equations are

$$x'(t) = e^{-0.002t} - 0.08x - xy^2$$

$$y'(t) = 0.08x - y + xy^2$$
(22)

which are to be solved on $[0, 1000]$ with initial conditions $x(0) = y(0) = 0$. Both solution components have the same qualitative behavior, so we show only $x(t)$ in Figure 2. Near $t = 200$ there is a burst of oscillations and then $x(t)$ settles back down near $t = 600$. The plot shows the twice averaged solution $\bar{s}(t)$ computed with $\Delta = 25$. Averaging once is considerably less oscillatory than $x(t)$, but averaging twice is smoother and captures well the long-term behavior of the solution. In our experiments we found that for all error tolerances, averaging twice required fewer derivative evaluations than averaging once which in turn required fewer derivative evaluations than computing the solution directly.
In §5 we discussed an example of Holmes and here display a plot of the solution $y(t)$ and twice averaged RMS amplitude $\bar{R}(t)$ as Figure 3. He approximates the peak amplitude $A(\epsilon t)$ and plots it along with $y(t)$ in his Figure 3.5. Our figure shows the same qualitative behavior with $\bar{R}(t)$ being roughly equal to $A(\epsilon t)/\sqrt{2}$, as would be the case for a pure sinusoidal solution. Holmes approximates the peak amplitude by deriving an IVP for an asymptotic approximation and then solving the IVP numerically. Our approach has the virtue of being completely straightforward. The amplitude of the oscillation is used to study resonance of the solution as the forcing frequency $\omega$ is varied, but Figure 4 shows that this can also be done with the RMS average. Specifically we have done the computation of Figure 3 for a range of $\omega$ and plotted the last value of $\bar{R}(t)$.

References


Figure 3: Solution and twice averaged RMS solution of (15) with $\Delta = 35$.

Figure 4: Final RMS amplitude for (15) as a function of $\omega$