Adams Methods for Ordinary Differential Equations

In these notes a Java program for approximating solutions of systems of ordinary differential equations (ODEs) is discussed. The particular mathematical method used is an Adams method; it is simple to understand and code, but it is efficient and robust enough for real work.

1 Initial Value Problems

Models of many situations result in rules that specify how a value changes when we know the value. For example the amount that is still owed on a mortgage can be described by an ODE of the for,

\[ b'(t) = rb(t) - p, \]

where \( t \) is time, \( b(t) \) is the balance owed at a given time, \( r \) is the interest rate, \( w \) is the payment rate. Of course real mortgages involve discrete events, but a continuous model is often easier to deal with. If we have \( r, p, \) and \( b(0) \) we have defined \( b \) for all \( t > 0 \).

A weight on a spring can be modeled by an ODE of the form

\[ x''(t) = -kx(t), \]

where \( x \) is the displacement from the equilibrium position and \( k \) is the spring constant divided by the mass of the weight. This ODE is called second-order, since it involves the second derivative of the function that is the solution. It can be converted into a system of two first order ODEs as follows. Let \( y(t) = (x(t), x'(t))^T \). The \( y \) satisfies

\[ y'(t) = f(y(t)), \]

where \( f = (f_0, f_1)^T \) and \( f_0(w_0, w_1) = w_1 \) and \( f_1(w_0, w_1) = -kw_0 \). To define \( y \) we need to know \( f \) and we need to know the value of \( y \) at some time \( t_0 \). The process of reducing higher-order ODEs to first order ones is quite general.

Here we will be looking at differential equations of the form

\[ y'(t) = f(t, y(t)). \]

In general we will be trying to find \( y \) on some interval \( J = [t_{initial}, t_{final}] \), where we are given the value of \( y \) at \( t_{initial} \). Such problems are called initial value problems (IVPs).
2 Second-Order Adams Method

In trying to approximate the solutions of IVPs we will be looking for approximations, \( Y_j \) of \( y(t_j) \) for some times \( t_j \) where \( t_{\text{initial}} = t_0 < t_1 < t_2 < \ldots < t_N = t_{\text{final}} \). For now you can think of the \( t_j \)'s as being given, but later we will talk about ways to define them adaptively, based on the behavior of the approximate solution. In describing methods we are usually interested in saying how we get to \( t_{j+1} \) if we know the approximate solution at \( t_j \) and earlier. Here we will call \( t_j \) \( t_{\text{now}} \) and will call \( t_{j+1} \) \( t_{\text{new}} = t_{\text{now}} + dt \). In addition we will call \( t_{j-1} \) \( t_{\text{old}} = t_{\text{now}} - dt_{\text{old}} \). The values of \( Y \) at these times will be called \( Y_{\text{now}}, Y_{\text{new}}, \) and \( Y_{\text{old}} \).

A relation that can be used to build approximation schemes is
\[
y(t_{\text{new}}) = y(t_{\text{now}}) + \int_{t_{\text{now}}}^{t_{\text{new}}} f(t, y(t))dt.
\] (2)

For example, if we know \( f_{\text{now}} = f(t_{\text{now}}, Y_{\text{now}}) \) and \( f_{\text{old}} = f(t_{\text{old}}, Y_{\text{old}}) \) we can approximate the integral by integrating the linear function of \( t \) that goes through the values \( f_{\text{old}} \) and \( f_{\text{now}} \) at the corresponding times. That gives
\[
Y_{\text{new}} = Y_{\text{now}} + dt (f_{\text{now}} + (dt/2)(f_{\text{now}} - f_{\text{old}})/dt_{\text{old}}).
\] (3)

This is a second-order Adams method.

Since this method uses two known values to advance, and the original problem only gives us one, we need to do something else to get started. A reasonable startup process is discussed below.

3 Third-Order Adams Method

If we knew the value of \( f_{\text{new}} \) in addition to \( f_{\text{now}} \) and \( f_{\text{old}} \) we could use a quadratic polynomial through the three values in the integral. Since we don’t know \( f_{\text{new}} \) we will make a prediction of its value by evaluating \( f_p = f(t_{\text{new}}, W) \), where \( W \) is a guess at \( Y_{\text{new}} \). We use the second-order Adams method above to get this guess.

\[
W = Y_{\text{now}} + dt (f_{\text{now}} + (dt/2)(f_{\text{now}} - f_{\text{old}})/dt_{\text{old}}),
\] (4)

and then define \( Y_{\text{new}} \) by
\[
Y_{\text{new}} = Y_{\text{now}} + \int_{t_{\text{now}}}^{t_{\text{new}}} p(t)dt,
\]
where \( p \) is the quadratic that has the values \( f_{\text{old}}, f_{\text{now}}, \text{and} \ fp \) at the times \( t_{\text{old}}, t_{\text{now}}, \text{and} \ t_{\text{new}} \), respectively.

We can get a simple formula for \( Y_{\text{new}} \) as follows. Let \( l(t) \) be the linear polynomial based on \( f_{\text{old}} \) and \( f_{\text{now}} \), the one used to get \( W \). Then \( p(t) = l(t) + (fp - l(t_{\text{new}}))q(t) \), where \( q \) is the quadratic which has roots at \( t_{\text{old}} \) and \( t_{\text{now}} \) and has value 1 at \( t_{\text{new}} \). Let \( c \) denote the integral on \([t_{\text{now}}, t_{\text{new}}]\) of \( q \), then

\[
Y_{\text{new}} = W + c(fp - l(t_{\text{new}})).
\]

The value of \( c \) is the same as the integral from 0 to \( dt \) of \( r(t) = t(t + dt_{\text{old}})/(dt(dt + dt_{\text{old}})) \), since \( q(t) = r(t + t_{\text{now}}) \). It is simple calculation to see that \( c \) is given by

\[
c = dt \frac{dt/3 + dt_{\text{old}}/2}{dt + dt_{\text{old}}}.
\]

### 4 Data for Third-Order Adams

In order to take a step of the method just described we need to know \( Y_{\text{now}}, f_{\text{now}}, f_{\text{old}}, t_{\text{now}}, dt, \text{and} \ dt_{\text{old}} \). It is easy to keep \( Y_{\text{old}} \) as well, and if we do then we have an Hermite cubic representation for \( Y \) between \( t_{\text{old}} \) and \( t_{\text{now}} \). In addition to these it is convenient to have a spot to store \( Y_{\text{new}} \), before it is accepted as \( Y_{\text{now}} \) and time is advanced. It is reasonable to have these as instance variables of a solver class. It is also useful to know if this is the first step, since something special will need to be done on that step. Another, thing that one needs to consider is output; for our purposes, this can be an output file to contain the states we compute. We also need to know how to evaluate the function \( f \).

In my implementation of this method I use a class called AdamsSolver. The vector variables are carried using the Vector class of Sedgwick and Wayne. The function \( f \) is required to implement an interface that I called TnVec2Vec. The interface is defined by

```java
public interface TnVec2Vec{
    Vector value( double t, Vector v); //returns a new Vector
}
```
5 Adaptive Time Step Control

We want our results to be reliable, so we need to have some way of judging the quality of a time step. We do not want to take a step that gives a large error, because there is no way to correct that in the future; it just moves us to a different trajectory. In addition to rejecting bad steps, we need a way to choose the next time step, for retrying a rejected step or moving ahead after an accepted one.

The basic idea in deciding when to accept a step is that we will compute some number $err$ that is an error indicator. We want $err < tol$, where $tol$ is some specified tolerance. It is not trivial to choose an error indicator or to pick a tolerance. In general we want our error indicator to be such that if it is small enough the answers we compute are accurate enough for our purposes. So we want the error in the solution process to decrease when the tolerance is decreased. Typically, we cannot say with precision how much accuracy we really need, since solving the IVP is only part of the modeling problem we are addressing. There is art that comes from experimentation to determine a reasonable level for the tolerance.

The error indicator that we choose for the tool we are building for third-order Adams solvers is the difference between the prediction $W$ and $Y_{new}$. We will use the Euclidean norm, but there are very good reasons to consider which norm to use carefully. The variables in $y$ typically have different dimensions, so adding them together without adjustment is problematic. (Even if we nondimensionalize the problem, we are really choosing units.) However, it is still the case that if we get the Euclidean norm of the difference between the prediction and the corrected value small enough we expect that the accepted result will be of high quality.

The algorithm recommended for time step adjustment is the following. There are user defined parameters $dt_{min}$ and $dt_{max}$, and the time step we use will (almost) always be in the interval $[dt_{min}, dt_{max}]$, unless a smaller step is needed to get to the final time. Among other things this allows us to run experiments with constant $dt$. A step is good if its error indicator does not exceed $tol$ or $dt$ is not bigger than $dt_{min}$. A step that is not good is bad, and it is not accepted. In this case the time step is cut by a factor of two (or set to $dt_{min}$ if that is larger than half the bad time step) and the step is retried. If the step is good, the error indicator is compared to the tolerance and an adjustment may be made. If the error indicator exceeds 75% of the tolerance, the recommendation for the next time step is the current step.
size multiplied by a factor \( dtshrink \). This recommendation, is compared to \( dt_{\text{min}} \) to select a next step size. If the error indicator is less than 20\% of the tolerance the recommended step is the current step multiplied by a factor \( dt_{\text{grow}} \), provided there have been as least two steps accepted since one was rejected. This recommended step is compared to \( dt_{\text{max}} \) to select the next step size. Reasonable values for \( dtshrink \) and \( dt_{\text{grow}} \) are 1/1.25 and 1.5. Different values for these parameters might impact the 75\% and 20\% above. Finally, there is a mechanism to try to avoid very small steps at the end of a sequence of steps that are required to end at a specified time. If the recommended step does not make it to the final time (within some small tolerance), but two such steps exceeds the final time, then we use a \( dt \) of one half the time to the final time; this might be less than \( dt_{\text{min}} \).

What information do we need to implement the above time step control? The list for my Adams solver is

\[
\begin{align*}
tol; \\
dtgrow; \\
dtshrink; \\
dt_{\text{min}}; \\
dt_{\text{max}}; \\
endTime; \\
err; \\
stepsSinceRejection; \\
stepsRejected; \\
stepsAccepted;
\end{align*}
\]

where the last two items are just for summary information about the adaptive process. I encapsulated this into a class called timeStepControl. Each instance of the Adams solver has one timeStepControl variable.

6 Start Up

To start the third-order Adams method we need some \( Y_{\text{old}} \) and some \( dt_{\text{old}} \). A plausible way to proceed is to have the constructor for an Adams solver set an instance variable to indicate it is not yet been used. I used a boolean, \( first \), for this. I took \( dt_{\text{old}} = 0.1dt_{\text{min}} \), set \( dt \) to \( dt_{\text{min}} \), and put \( W = Y_{\text{now}} - dt_{\text{old}}f_{\text{now}} \).

Next I put \( Y_{\text{old}} = Y_{\text{now}} - \frac{dt_{\text{old}}}{2}(f_{\text{now}} + f(t_{\text{now}} - dt_{\text{old}}, W)) \), and computed \( f_{\text{old}} = f(t_{\text{now}} - dt_{\text{old}}, Y_{\text{old}}) \).
7 Other Places to Read

Here are some places to read about this type of method and many others as well.


Numerical Methods, J. Douglas Faires and Richard L. Burden, 3rd edition. See Chapter 5. There is a discussion of several adams methods and of adaptity.

January, 2009 –tfd