Notes on BVP-ODE

-Bill Green

There are multiple methods for solving systems of ordinary differential equations (ODEs) or differential-algebraic equations (DAEs) posed as boundary value problems (BVPs) of the form:

$g(y''(t),y'(t), y(t), t) = 0$ at all t in the domain

plus B boundary conditions that hold only at specific values of t called tx:

$q(y'(tx),y(tx),tx) = 0$ where usually $tx=t_0$ or $tx=t_f$

Note that g and q are vector-valued functions, and y is usually a vector also.

If g represents a first-order system of differential equations, dim(y) =dim(g)=dim(q). If it is second-order we will need more boundary conditions, dim(q)>dim(g). If it is a DAE we will need fewer. Just because you have the right number of boundary conditions does not guarantee a unique solution exists: the system of equations may have no solution or multiple solutions. Usually this can be fixed by changing the boundary conditions.

Here are some methods for solving BVP-ODE's:

1) Shooting

Recast the ODE-BVP as an ODE-IVP with some unknown initial values, call these Z. Guess those unknown values Z and solve the resulting ODE-IVP. The solution Y will not satisfy all the boundary values $Y_{bc}(t_f)$, i.e. there is a deviation

$$\text{residual}(Z) = Y(t_f,Z) - Y_{bc}(t_f)$$

where $Y(t_f,Z)$ is the computed value of $Y(t_f)$ from the ODE-IVP using the initial values Z. So we can embed that calculation inside a nonlinear equation solver like fsolve, e.g.

```
Zbest=fsolve(@residual,Zguess)
```

The advantage of shooting is that there are only a few unknown initial conditions, so fsolve is only solving a few equations with a small Jacobian. The disadvantage is that an ODE-IVP problem must be solved at each iteration of fsolve, and that can be expensive particularly if an implicit ODE solver is used.

2) Collocation:

Approximate the solution y(t) as a sum of some basis functions:

$$y(t) = \Sigma\, d_n\, \phi_n(t)$$

This converts the problem into computing the N d's. The collocation method writes the N equations required this way: B of the equations come from the boundary conditions. The remaining N-B equations come from choosing N-B points in the domain $\{t_m\}$ and demanding

$$g(y'(t_m;d),y(t_m;d),t_m) = 0 \quad \text{for these particular } \{t_m\}$$

Note that all the equations depend on the unknown vector d, since y is a function of d. So we could rewrite the equations this way:

$$g(d)=0 \text{ and } q(d)=0$$

This is the form for nonlinear equations solvers like fsolve.

Note that the user can choose which basis functions $\{\phi_n(t)\}$ and $\{t_m\}$ which to use; for each choice you'll get a different approximate solution y(t). Usually increasing the number of basis functions and collocation points $\{t_m\}$ increases the accuracy of the approximation. However, fsolve or similar programs typically need to evaluate the Jacobian of f(d)=(g(d);q(d)), which has N^2 elements. If N is very large this can be expensive and for very large N it is likely that the cond(Jacobian) will be large. One can make the Jacobian matrix sparser by making using a local basis (discussed below), that saves CPU time and might improved the conditioning.

Note that a poorly-conditioned Jacobian means that varying some linear combination of the d's does not change the quality of the solution very much. This implies that there is a better choice of basis functions and/or collocation points $\{t_m\}$.

The disadvantage of collocation is that typically you need a lot of basis functions and collocation points to achieve high accuracy, so fsolve will need to solve a large system of equations. The advantage is that not much work is required to compute the residuals (no embedded ODE solves or numerical integrations).

3) Galerkin:

Approximate the solution y(t) as a sum of some basis functions:

$$y(t) = \Sigma\, d_n\, \phi_n(t)$$

This converts the problem into computing the N d's. B of the equations come from boundary conditions. In Galerkin's method, the N-B additional equations needed to determine the d's are of this form:

$$\int \phi_k(t)\, g(y'(t;d),y(t;d),t) = 0$$

If the integrals can all be evaluated analytically, these integral equations become explicit algebraic equations in the unknowns d, can be solved using fsolve. This can also work if the integrals are evaluated numerically, but it may be necessary to re-evaluate numerical integrals inside each iteration of fsolve, so this can be expensive.

Note that in both Collocation and Galerkin method, fsolve and similar programs evaluate the Jacobian of the system of equations, which has N^2 elements. This can be very large if N is large. One can simplify the integral evaluations and make the Jacobian sparser (and so easier to evaluate and store) by using a local basis, discussed below.

4) Finite Differences

A different approach is to approximate all the derivatives by finite-difference expressions. A common simple approach is to choose a an evenly spaced set of collocation points $\{t_m\}$ and use centered differences, e.g. approximate $y'(t_m) = (y(t_{m+1})-y(t_{m-1}))/(2\Delta t)$. The unknowns you are solving for are $\{y_m\}$. Note this approach only gives you a set of points, not the values between (though you could interpolate), and it only works if Δt is small enough that the finite difference closely approximates the derivative, so you need a lot of points. But this usually gives a sparse Jacobian and it is easy to evaluate the residuals. You can increase Δt if you use a high-order finite differencing formula, this add some bands to the Jacobian but it will still be sparse.

Local Basis Functions

It is often beneficial to choose "local" basis functions,

i.e. choose $\phi_n(t)$ so that it is only nonzero is a small range, i.e. $\phi_n(t)=0$ if $|t-t_n|>X$

Use of local basis functions makes g(t') depend only on the small number of basis functions $\phi_n(t)$ with t_n close to t'. This directly makes the Jacobian used in the Collocation method very sparse. Also, in the Galerkin method, if $\phi_k(t)$ is local, then one only needs to integrate over a small range of t's near t_m, and the resulting integral will only depend on a few d's. So use of a local basis also make the Jacobian used in the Galerkin method sparse.

The most popular Local Basis functions are B-splines, in particular the 1^{st}-order B-splines called "tent" functions defined this way:

$\phi_k(t)=(t-t_{k-1})/(t_k-t_{k-1})$ if $t_k>t>t_{k-1}$
$\phi_k(t)=(t_{k+1}-t)/(t_{k+1}-t_k)$ if $t_{k+1}>t>t_k$

For all other t, $\phi_k(t)=0$.

With these basis functions, one is trying to do a piecewise linear approximation t the solution y(t). Note that these basis functions have discontinuous first derivatives. In Galerkin's method this unsmoothness doesn't matter much since one is mostly evaluating integrals. However, it can cause complications if the boundary conditions involve derivatives, and in the Collocation method one would be well-advised to avoid the discontinuities, e.g. by choosing $\{t_m\}$ and $\{t_k\}$ so that $t_m \neq t_k$.

10.34 Numerical Methods Applied to Chemical Engineering

Fall 2015