AE353: Additional Notes on Numerical Integration

A. Borum T. Bretl M. Ornik P. Thangeda

1 Numerical Integration

In order to compute the cost in a linear quadratic optimal control problem, which is

$$x(t_f)^T F x(t_f) + \int_{t_0}^{t_1} \left(x(t)^T Q x(t) + u(t)^T R u(t) \right) \, dt,$$

you need to evaluate an integral. This integral is hard to evaluate symbolically, but it can be evaluated using numerical integration.

In particular, suppose you want to compute

$$\int_a^b g(x) \, dx.$$

"Integral" means "area under the curve." You can approximate this area with many rectangles. Suppose you define $h = (b-a)/n$ for some integer n. Then

$$\int_a^b g(x) \, dx \approx \sum_{k=0}^{n-1} h \; g(a + kh).$$

You can compute this sum in Matlab with a for loop. The accuracy of this method depends on h – smaller h gives better accuracy. Sadly, smaller h also means bigger n, so more terms in the sum, so it takes longer to compute. There are functions in Matlab that do a good job making this tradeoff automatically, so you get both good accuracy and fast computation.

In particular, there is this function:

```
Q = integral(FUN,A,B) approximates the integral of function FUN from A
    to B using global adaptive quadrature and default error tolerances.
```

As an example, suppose

$$g(x) = \cos(x), \qquad a = 0, \qquad b = \pi/2.$$

In this case, the integral is easy to evaluate by hand – the result is 1. Let's see if we get the same thing with Matlab:

```
>> integral(@(x) cos(x),0,pi/2)

ans =

    1.0000
```

It works. The first argument defines $g(x)$. The second and third arguments specify the limits of integration. Consider another example:

$$g(x) = x^2, \qquad a = 0, \qquad b = 1.$$

```
>> integral(@(x) x^2,0,1)
Error using  ^
Inputs must be a scalar and a square matrix.
To compute elementwise POWER, use POWER (.^) instead.

Error in @(x)x^2

Error in integralCalc/iterateScalarValued (line 314)
                fx = FUN(t);

Error in integralCalc/vadapt (line 132)
            [q,errbnd] = iterateScalarValued(u,tinterval,pathlen);

Error in integralCalc (line 75)
        [q,errbnd] = vadapt(@AtoBInvTransform,interval);

Error in integral (line 88)
Q = integralCalc(fun,a,b,opstruct);
```

What happened? We didn't actually read all of "help integral":

```
For scalar-valued problems the function Y = FUN(X) must accept a vector
    argument X and return a vector result Y, the integrand function
    evaluated at each element of X.
```

What this means is that the function I pass to integral must evaluate $g(x)$ at many different x values, not just one. This is easy to fix. Let's create an m-file:

```
function testIntegration
integral(@g,0,1)
end

function y = g(x)
for i=1:length(x)
    y(i) = x(i)^2;
end
end
```

Then call:

```
>> testIntegration

ans =

    0.3333
```

Bingo. But wait, suppose I wanted to integate

$$g(x) = cx^2, \qquad a = 0, \qquad b = 1.$$

where c is a parameter? No problem!

```
function testIntegration
c = 5;
integral(@(x) g(x,c),0,1)
end

function y = g(x,c)
for i=1:length(x)
    y(i) = c*x(i)^2;
end
end
```

Result:

```
>> testIntegration

ans =

    1.6667
```

This is all standard – there are tons of examples both in Matlab (doc integral or doc quad) and online.

One last note. If you get an error message that says "inner matrix dimensions do not agree," this means that you are trying to multiply two matrices that have incompatible sizes (e.g., 3×4 multiplied by 1×2 – you can't do it). This means that something is wrong with your understanding, not with Matlab – that's a great opportunity!! Take the time to figure it out - don't just tinker until it comes out right.

2 Solving ODEs numerically

The methods for numerically solving ordinary differential equations (ODEs) are similar to those in the previous section for numerical integration. Suppose we want to solve the ODE

$$\dot{x}(t) = f(x(t)), \qquad x(a) = x_a \tag{1}$$

on the time interval $t \in [a, b]$, i.e., we want to find the value of $x(t)$ for each t in the interval $[a, b]$. We can approximate the left-hand side of the ODE with a finite difference. In particular, let $h = (b - a)/n$ for some integer n. Then for each $k = 0, 1, \ldots, n - 1$, we have

$$\dot{x}(a + kh) \approx \frac{x(a + (k + 1)h) - x(a + kh)}{h}.$$

Since $\dot{x}(a + kh) = f(x(a + kh))$, we can rewrite the ODE (1) approximately as

$$x(a + (k + 1)h) = x(a + kh) + hf(x(a + kh)).$$

This gives us a recursive way of finding $x(t)$ at each $t = a, a + h, a + 2h, \ldots a + nh$, i.e.,

$$\begin{aligned} x(a) &= x_a \\ x(a + h) &= x(a) + hf(x(a + h)) \\ x(a + 2h) &= x(a + h) + hf(x(a + h)) \\ &\vdots \\ x(a + nh) &= x(a + (n - 1)h) + hf(x(a + (n - 1)h)). \end{aligned}$$

As was the case with numerical integration, decreasing h, and therefore increasing n, improves the accuracy of our approximation but also requires more computation time.

Just as the Matlab function `integral` does a good job making this tradeoff automatically, there are similar functions for solving ODEs. We'll use `ode45`. Let's solve the ODE

$$\dot{x}(t) = x(t) - x(t)^3, \qquad x(0) = 0.5$$

on the time interval $t \in [0, 1]$. In Matlab, we can solve this ODE as follows:

```
1 >> [t,x] = ode45(@(t,x) x-x^3,[0 1],0.5);
```

Let's look at the arguments and outputs of `ode45` more closely. The first argument in `ode45` defines $f(x)$. The `@(t,x)` tells Matlab that what follows is a function of both time `t` and the variable `x`. In some problems, our differential equation may depend explicitly on time, e.g., consider the differential equation $\dot{x}(t) = tx(t)$. The first argument of `ode45` would then be `@(t,x) t*x`. Since our ODE does not depend explicitly on time, we have no `t` terms in the first argument. However, we must still include the `@(t,x)`. Otherwise, Matlab will give us an error.

The second argument is the time interval on which we want to solve the problem, in this case `[0 1]`. The third argument is the initial value for x, in this case `0.5`. The first output `t` is a column vector of times, and the second output `x` is a column vector containing the values of x at the times in `t`. Note that, in this case, the first element of `t` is 0 and the last element of `t` is 1. Similarly, the first element of `x` is $x(0) = 0.5$ and the last element of `x` is $x(1)$.

What if we want to solve an ODE, but specify the value of x at the final time instead of the initial time? In particular, suppose we want to solve the ODE

$$\dot{x}(t) = x(t) - x(t)^3, \qquad x(1) = 0.5$$

on the time interval $t \in [0, 1]$. All we must do is switch the order of the time interval in `ode45`:

```
>> [t,x] = ode45(@(t,x) x-x^3,[1 0],0.5);
```

Note that now, the first element of the column vector `t` is 1 and the last element of `t` is 0. Similarly, the first element of `x` is $x(1) = 0.5$ and the last element of `x` is $x(0)$.

We often need to solve higher-order ODEs or systems of ODEs. As an example, consider the second-order ODE

$$\ddot{x}(t) = x(t) - x(t)^3, \qquad x(0) = 0.5, \qquad \dot{x}(0) = 0$$

on the time interval $t \in [0, 1]$. In order to use `ode45`, we must write this ODE as a first-order system:

$$\begin{aligned} \dot{x}_1(t) &= x_2(t) & x_1(0) &= 0.5 \\ \dot{x}_2(t) &= x_1(t) - x_1(t)^3 & x_2(0) &= 0. \end{aligned}$$

To implement this ODE in Matlab, we'll write an m-file:

```
function [t,x] = testODE

[t,x] = ode45(@f,[0 1],[0.5; 0]);
end

function dx = f(t,x)
dx1 = x(2);
dx2 = x(1) - x(1)^3;
dx=[dx1; dx2];
end
```

When we define the function `f`, note that we must include the arguments `t` and `x`, even though we don't use `t`. This is done for the same reason as discussed earlier.

The output of our function `testODE` will be a column vector of times `t` and a matrix `x` containing the values of x at each time in `t`. The first column of `x` contains the values of x_1, and the second column of `x` contains the values of x_2. Therefore, if we want to extract each component of x, we would use:

```
>> [t,x] = testODE;
>> x1 = x(:,1);
>> x2 = x(:,2);
```

t