# Assignment 3: Variable Time steps

September, 2010 Guanqing Ou and Arash Ushani

## Problem Statement

Successfully implement three root-finding methods, bisection, Newton’s method, and the secant method, to solve a test problem to determine the depth at which a duck of a given volume and density floats in water. Evaluate these methods based on efficiency and stability.

Combine a previously tested ODE solver with a chosen root-finding algorithm to solve an example boundary value problem:

using the shooting method.

### Computational Approach

Root-finding Methods

1. Bisection

The simplest of the three root-finding methods to be discussed, the bisection method begins with a user selected interval , which is assumed to contain a root for the system in question. The algorithm allows us to find which half of the interval—either or —contains the root and continues this process using the new interval until the desired tolerance is achieved.

1. Newton’s Method

Newton’s method requires a single initial guess instead of an interval. The derivative of the function in question is evaluated at this point and used to find the x-intercept of the line passing through the initial guess, which is taken as the new guess. This process is repeated until the desired tolerance is achieved.

1. Secant Method

The secant method involves picking two initial guesses, a and b, evaluating these values, and finding the x-intercept of the line that passes through and (b, . This new point,, replaces , and the process is repeated until convergence is achieved. This method is very similar to Newton’s method, except rather than computing a derivative at one point, it uses the secant between two points to make a linear approximation for the function in question.

Shooting Method

In boundary value problems, an initial and end value are given for some function . In the case of our example, we know and . The shooting method tries to start with the known initial value and “end up” at the other boundary value by guessing the derivative of the function at the first boundary value, and then estimating the intermediate values using a method such as Euler’s or Modified Euler’s Method. The shooting method can be used in conjunction with a root-finding method to find an accurate guess for, in this case, This can be done by establishing a new function , which yields , which is normally calculated with Euler’s or Modified Euler’s method. Finding the root of means finding the value of that gives the correct result for Euler’s method can then be used to solve the ODE.

## Implementation and evaluation

### The duck problem

To validate successful implementation before proceeding with attempting to solve the given ODE, each of these methods was tested on a simple problem:

The density of a duck, ρ, is (0.3 times the density of water). The volume of a sphere with radius is . If a sphere with radius is submerged in water to a depth , the volume of the sphere below the water line is

Volume =

An object ﬂoats at the level where the weight of the displaced water equals the total weight of the object. Assuming that a duck is a sphere with radius 10 cm, at what depth does a duck ﬂoat?

This problem needs to be restated such that its solution is the answer provided by a root-finding method. Essentially, we want to find such that the difference between the weight of the duck and the volume of displaced water times the density of water is zero:

This definition gives us a function for which to root-find.

Each of the root-finding methods described above were implemented in MATLAB, and associated code is shown below:

Bisection:

function newguess = bisect (functn, guess)

range = guess (:,end) - guess (:,1);

beg = guess(:,1);

half = range/2;

intrvlend = guess (:,end);

left = [beg, beg+half];

right = [beg+half, intrvlend];

if (functn(beg)>0) == (functn(beg+half)>0)

newguess = right;

else

newguess = left;

end

res = newguess;

end

Newton’s method:

function res = newtons(functn, initial, step)

if nargin < 3

step = [.001 .001];

end

if nargin < 2

initial = [3.1 1];

end

if nargin < 1

functn = @shooting;

end

slope = (functn(initial + step)-functn(initial))./step;

newpt = initial - functn (initial)./slope;

res = newpt;

end

Secant method:

function res = secant(functn, initial)

if nargin < 2

initial = [2 2.01];

end

if nargin < 1

functn = @blah2;

end

olderpt = initial (:,1);

oldpt = initial (:,2);

newpt = oldpt - functn(oldpt) \*(oldpt- olderpt)/(functn(oldpt)-functn(olderpt));

res = [oldpt newpt];

end

Each of these functions advances each method for one step. They can then called in a general rootfind function, in which it is possible to pick the method by which a root is found:

function root=rootfind (functn, method, guess, tolerance1)

answer(1,:) = guess;

for i = 1:1000

answer (i+1,:) = method (functn, answer(i,:),.001);

temp1 = answer(i,:);

temp2 = answer(i+1,:);

left = temp1(:,1);

right = temp2(:,1);

if abs(left - right)< tolerance1

break

else

continue

end

end

root = answer (end);

res = root;

end

All three methods give the same answer for the duck problem (), which strongly suggests that all three were implemented correctly.

While these methods all gave the right answer for the duck problem, they are not equal in terms of dependability and efficiency. In general, one would expect the bisection method to be extremely slow, as it is limited by the fact that each step leads to a 50% reduction of the interval size. Furthermore, it is highly dependent on a good initial guess. We can see in Figure 1 that in the case of the duck problem, the bisection method took the highest number of steps. In a more complex problem, where the range of possible answers is greater, we can expect this difference to be more significant. Despite its relative inefficiency, the bisection method is very dependable. Given that the initial interval contains a root, the method will always converge on an answer. This is not true for the secant and Newton’s method. While they converged on a root for the duck problem, it is entirely conceivable that the secant or tangent lines could actually lead one farther from the root, instead of converging on the root. Even if convergence does occur, these two methods, unlike bisection, do not necessarily continually narrow in on the root. Instead, the proximity of any guess to the actual root is not correlated to step number, but can fluctuate, as seen in the implementation of the secant method in Figure 1.

C:\Users\gou\Documents\Classes\(4) Numerical Methods\error vs step number root finding duck.tif

Figure 1. The absolute difference between guesses as a function of step number.

### Shooting method

With the validity of our root-finding methods established, we can proceed to using the shooting method to solve the given ODE. As previously stated, the shooting method involves making an initial guess for the value of the derivative at the first boundary value, and using Euler’s or Modified Euler’s method to step the function forward to the other boundary. The calculated value of the function at this boundary is then compared to the given value, and root finding is utilized to find the value of that makes the difference between these two values zero. In MATLAB, this procedure is implemented as follows:

function res = shooting(v\_d)

res = shoot([1 v\_d]);

end

function res = shoot(state, ode, tstep, boundary)

if nargin < 4

boundary = [0 1];

end

if nargin < 3

tstep= .001;

end

if nargin < 2

ode = @tut3;

end

if nargin < 1

state = [1 0];

end

length = abs( boundary (:,1) - boundary (:,2));

temp = euler (state, ode, tstep, length/tstep);

res = temp(:,1);

end

In the function shoot, the function is stepped forward from the first initial value using Euler’s method with constant time step . Euler’s method requires two derivatives: the first and second derivative because it solves for both and The first v’(t) is given by our guess, and the second derivative can be calculated using this value and the ODE . This process can then be repeated for each subsequent step. This function is shown below:

function res = tut3(state, dt, x)

if nargin < 3

x = 1;

end

if nargin < 2

dt = .001;

end

if nargin < 1

disp ('no initial values');

return

end

v = state (:,1);

vprime = state (:,2);

vdoubleprime = x^2 - 2\*x\*vprime + x^2\*v;

vprimenew = vprime + vdoubleprime \* dt;

res = [vprimenew, vdoubleprime];

end

### ODE solving and reflections

The result of root-finding for the shooting method is used to solve the ODE. Because we employed Euler’s method to shoot for the correct we use Euler’s method with constant time step to visualize our solution:

C:\Users\aushani\Desktop\Documents\MATLAB\grace\ode solution graph.tif

Figure 2. Solution for tutorial 3 ODE, found using Euler’s method to implement from the shooting method with Newton’s method of root finding.

As in the duck problem, we compare the number of step necessary to find a root for for each root-finding method. As can be seen in Figure 3, bisection still takes the longest time, whereas Newton and secant both converge on the answer in less than 4 steps. Furthermore, as before, we see that the difference between guesses for the secant method can fluctuate.

C:\Users\gou\Documents\Classes\(4) Numerical Methods\error vs step number root finding.tif

Figure 3. Differences in using the different root-finding methods to find . The difference between subsequent guesses is plotted as a function of step number.

In deciding which root finding method to use to help solve a boundary value problem, it is helpful to consider one’s values. Bisection offers the most reliable method, but is slow. Newton’s method is quick and generally dependable, but requires knowing how to calculate the derivative. Secant method is perhaps the best option, as it has the same benefits as Newton’s method but does not require calculating the derivative. The only consideration one must bear in mind is the fact that it is not necessarily guaranteed that Newton’s method or secant method will converge on an answer. Depending on where the tangent and secant lines intersect the solution curve and where the root of the function is, it is possible to actually move progressively farther away from a root using these methods. Perhaps the best option is to attempt to solve problems using all three approaches, and to reap the benefits of both accuracy and efficiency.