Report on the Bisection Method

Benjamin W. Ong

January 26, 2015

Problem Description

The problem we were presented with, was finding the monthly payment, m (in dollars) to pay off a loan amount L (in dollars) at an annual interest rate of $r\%$, for a period of y years. If m is too little, we will still owe money at the end of the loan period. If m is too large, we will have paid too much. We will use the bisection method, a root-finding method based on bracketing, to find the required monthly payment.

Relevant Mathematics

First, we need to state the problem in a way that allows us to apply a root finding method. Given the parameters L, r and y, we can compute the balance b at the end of the loan for any m. Hence, the goal is to find the monthly payment m so that the end balance b is zero. Here is the algorithm to compute b.

```
function [ balance ] = get_balance(loan_amount,rate,year,monthly_payment)
% function [ balance ] = get_balance(loan_amount,rate,year,monthly_payment)
% loan_amount = initial loan amount (in dollars)
% rate = annual interest rate (in percent)
% year = length of loan (in years)
% monthly_payment = monthly payment (in dollars)

% Set the balance initially to be the initial loan amount
balance = loan_amount;

% for each month of the loan
```

```
for k = 1 : year*12
    balance = balance * (1 + rate/100/12) - monthly_payment;
end
```

For each month of the loan, the balance increases based on the interest rate (annual interest rate divided by twelve to get the monthly rate, divide by a hundred to convert from percentage to decimal), and decreases based on monthly amount paid.

The root finding method we want to implement requires two guess, m_0, m_1 which will result in a negative and positive balance. If we choose $m_0 = 0$ (i.e., we don't make any monthly payments, than the balance is going to be positive, $b(m_0) > 0$, provided $L, r > 0$. If we choose $m_1 = L$ (i.e. we pay a monthly amount equivalent to the initial loan every month), then we will certainly have paid too much, $b(m_1) < 0$. The code for the bisection method is :

```
function [monthly_payment] = bisection(m_0,m_1)
%function [monthly_payment] = bisection(m_0,m_1)
% m_0 = guess for monthly payment
% m_1 = guess for monthly payment

% initialize appropriate variables, for example
y = 25; % 25 year loan
r = 4.3; % 4.3% interest rate
L = 100000; % initial loan of $100,000

b_0 = get_balance(L,r,y,m_0);
if (abs(b_0) < 0.01) % if the balance either way is less than one cent
  monthly_payment = m_0;
  return;  % return the monthly payment without evaluating the rest of the function
end

b_1 = get_balance(L,r,y,m_1);
if (abs(b_1) < 0.01) % if the balance either way is less than one cent
  monthly_payment = m_1;
  return;  % return the monthly payment without evaluating the rest of the function
end

% check to make sure initial guesses are reasonable, otherwise return error
if (b_0*b_1 > 0)
  error(['initial guess must bracket the root, '...
         'i.e. one negative balance, one positive balance'])
end

% actual bisection loop
```

```
iterate = 1;
fprintf('iterate, interval, uncertainty\n')
fprintf('------------------------------\n')
fprintf('0, [%.2f %.2f], %.2f \n',m_0,m_1,abs(m_1-m_0)/2);

% while we don't have an estimate to the nearest cent
while ( abs(m_1-m_0) > 0.01 )
  % calculate midpoint of interval
  m_mid = (m_0 + m_1) / 2;

  % evaluate balance using m_mid
  b_mid = get_balance(L,r,y,m_mid);

  % if b_mid is less than one cent return b_mid and exit
  if ( abs(b_mid) < 0.01)
    monthly_payment = m_mid;
    return;
  elseif ( (b_mid * b_0) > 0 )
    % if b_mid and b_0 are same sign, replace m_0 with m_mid
    b_0 = b_mid;
    m_0 = m_mid;
  else
    % replace m_1;
    b_1 = b_mid;
    m_1 = m_mid;
  end
  fprintf('%d, [%.2f %.2f], %.2f \n',iterate,m_0,m_1,abs(m_1-m_0)/2);
  iterate = iterate + 1;
end

% when loop completes, return the average of m_0 and m_1
monthly_payment = (m_0 + m_1)/2;

fprintf('monthly payment amount should be $%.2f \n',monthly_payment);
```

Discussion

The bisection is a bracketing method, that is, every iteration (of the while loop) decreases the "bracket" which contains the root. Mathematically, we are increasing the "precision" of our estimate at every iteration. Here is a table showing the iterate in the first column, the interval containing the root in the second column, the "precision" (or uncertainty) that we know the monthly payment amount to in the third column. Given the initial guess of

$m_0 = 0$, $m_1 = L$, this code took 24 iterations for $L = 100000, r = 4.3\%, y = 25$. The computed monthly payment (to the nearest cent) is \$544.54.

iterate	interval	precision (uncertainty)
0	[\$0.00 \$100000.00]	\$50000.00
1	[\$0.00 \$50000.00]	\$25000.00
2	[\$0.00 \$25000.00]	\$12500.00
3	[\$0.00 \$12500.00]	\$6250.00
4	[\$0.00 \$6250.00]	\$3125.00
5	[\$0.00 \$3125.00]	\$1562.50
6	[\$0.00 \$1562.50]	\$781.25
7	[\$0.00 \$781.25]	\$390.62
8	[\$390.62 \$781.25]	\$195.31
9	[\$390.62 \$585.94]	\$97.66
10	[\$488.28 \$585.94]	\$48.83
11	[\$537.11 \$585.94]	\$24.41
12	[\$537.11 \$561.52]	\$12.21
13	[\$537.11 \$549.32]	\$6.10
14	[\$543.21 \$549.32]	\$3.05
15	[\$543.21 \$546.26]	\$1.53
16	[\$543.21 \$544.74]	\$0.76
17	[\$543.98 \$544.74]	\$0.38
18	[\$544.36 \$544.74]	\$0.19
19	[\$544.36 \$544.55]	\$0.10
20	[\$544.45 \$544.55]	\$0.05
21	[\$544.50 \$544.55]	\$0.02
22	[\$544.52 \$544.55]	\$0.01
23	[\$544.54 \$544.55]	\$0.01
24	[\$544.54 \$544.54]	\$0.00

A couple of interesting observations:

- Evaluating `get_balance(100000,4.3,25,544.54)` gives \$0.86. I guess we still owe 86 cents at the end of the loan. (Unavoidable, since the closest precision that makes sense is in dollars and cents. I'm sure the lender / lendee will not mind our solution!
- The bisection method keeps giving us more accurate/precise numbers. This is a very useful feature! Not all algorithms behave this nicely.
- With our initial guess, the bisection algorithm takes a few iterations before giving us reasonably accurate solutions that we would be happy with.
- We made our stopping criterion $|m_1 - m_0|$ to ensure that we have an answer to the closest cent, but also allowed for the off chance that b_0, b_1 or b_{mid} was less than one cent.