1.00/1.001 Introduction to Computers and Engineering Problem Solving

Fall 2002

Problem Set 7

Due: Day 24

Problem 1. YTM (50%)

BeaverBank.com starts offering a new student loan program to 1.00 students that works as follows. Once you sign the loan, you receive the loan amount *L* immediately. (Well, you don't actually get to touch the money yourself, but it goes into your account at the Bursar's office.)

To repay the loan, you are obligated to pay a fixed amount *M* each month, starting immediately, for *N* months. For the last month, you must pay a lump sum *S*.

For example, in package A, listed below, BeaverBank loans you $10,000 and requires you to pay $50 a month for 180 months. Your first payment begins immediately (at time *t=0*). In the final month, you are required to pay a lump sum fee. In package A, the lump sum fee is $12,000. (Note that you do *not* pay the $50 fee in the final month.)

Upon reading BeaverBank's brochure, you are angered and confused by the dizzying assortment of loan packages that are offered (Table 1). You want to know which loan will give you the best deal.

Package	Loan amount	Number of months	Monthly payment	Lump Sum due at end
A	$10,000.00	180	$50.00	$12,000.00
B	$20,000.00	36	$250.00	$14,000.00
C	$30,000.00	120	$125.00	$30,000.00
D	$30,000.00	60	$200.00	$25,000.00
E	$30,000.00	120	$325.00	$0.00
F	$30,000.00	120	$0.00	$50,000.00

Table 1: BeaverBank’s Educational Loan Packages

You consult with a finance professor at MIT Sloan who tells you not to worry. You can make a fair comparison of the different packages by solving for the "yield-to-maturity", *r*, given in Equation 1:

Equation (1) $L = \frac{S}{(1+r)^N} + \sum_{t=0}^{N-1} \frac{M}{(1+r)^t}$

The option with the lowest yield-to-maturity (expressed as a percentage) will cost you the least. The bank will make the most money out of the option that has the highest yield-to-maturity.

Assignment

Write a program in Java® to do the following:

(a) Read in the following amounts from the user using a pop-up window:

> Amount of loan in dollars (L)
> Number of months until payoff (N)
> Monthly fee in dollars (M)
> Lump sum payment at end in dollars (S)
> # of decimal places of precision (P)

You don't have to build a custom GUI to get this information from the user. You can simply use a `JOptionPane` dialog to input each value in succession.

(b) Calculate *r* and print it as a percentage. Your algorithm may assume that *r* is less than 100% and greater than 0%. For example, if *r* turns out to be 0.0725, your program should print *r* as 7.25%. (Represent *r* internally as a value between 0 and 1.)

Your program must calculate *r* numerically, using the bisection method. Since we already know r has a value between 0 and 1, that means, a root can be found between 0 and 1 if the function is $f(r) = \frac{S}{(1+r)^N} + \sum_{t=0}^{N-1} \frac{M}{(1+r)^t} - L$. Because the value for *r* might not be a round number, the user will specify the number of decimal digits of precision to use when solving equation (1). We define this value P as the number of digits after the decimal point in the "7.25%" representation of *r*.

Thus, if the true value of r were 3.1415926%, then your program should calculate the value of *r* to P digits, and print it out as shown below.

P	*R*
0	`3%`
1	`3.1%`
2	`3.14%`
3	`3.141% (or 3.142%)`
…	

You should only print out the significant digits, but don't obsess about this too much. Because floating-point numbers may not be able to represent all decimal values, a value like 3.14099999999999 might be produced when you expected 3.141.

(c) Note that the r given above is a monthly rate, not an annual rate. In addition to the monthly rate given above, calculate and print out the annualized YTM as a percentage, according to equation (2). Print out only P digits of the annualized r, with the same caveat as in (b).

$$\text{Equation (2)} \quad r_{\text{annualized}} = (1+r)^{12} - 1$$

(d) Run your program for each package in table 1 with at least three digits of precision. Summarize your results, including r and the annualized r in a table in the comments at the end of your code. Which is the best package for you? For BeaverBank.com?

Your output for loan package A with 3 digits of precision should look something like:

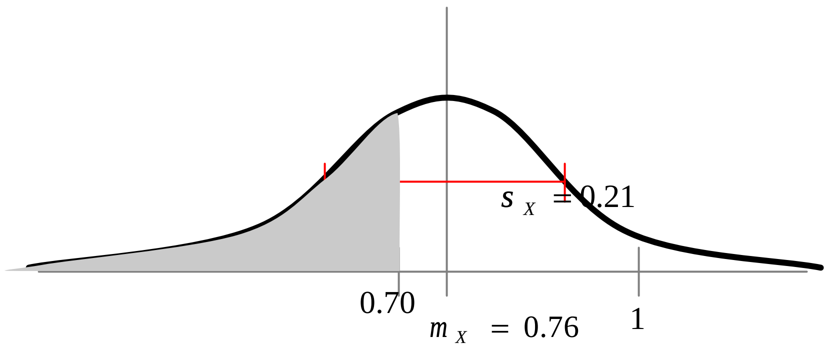
```
yield to maturity is 0.5660000000000001%
annualized YTM is 7.007%
```

Problem 2. Gaussian Distribution (50%)

When many independent random factors act in an additive manner to create variability, data will follow a bell-shaped distribution called the Gaussian distribution. This distribution is also called a Normal distribution (don't confuse this use of the word "normal" with its usual meaning). The Gaussian distribution has some special mathematical properties that form the basis of many statistical tests. Although no data follows that mathematical ideal, many kinds of data follow a distribution that is approximately Gaussian.

The normal distributions are a very important *class* of statistical distributions. All normal distributions are symmetric and have bell-shaped density curves with a single peak. To speak specifically of any normal distribution, two quantities have to be specified: the mean m (pronounced mu), where the peak of the density occurs, and the standard deviation s (pronounced sigma), which indicates the spread or girth of the bell curve. Different values of m and s yield different normal density curves and hence different normal distributions

Let's take a look at a familiar example: the distribution of grades on a 1.00 mid-term exam. Our assumption is that the grades follow a normal distribution. (This cannot be true in reality, because nobody can get a negative grade or a grade greater than 100. But for the range 0 to 100, a normal distribution is probably a reasonable assumption to make.)



The diagram above shows a normal distribution. The mean grade is 0.76 (76%) and the standard deviation is 0.21 (21%). The shaded portion (up to 0.7) is the integral of the normal distribution and is given by:

$$F(t) = \int_0^t \frac{1}{\sqrt{2p}\, s} e^{\frac{(t-m)^2}{-2s^2}} dt$$

where t=0.7. This is called the *cumulative distribution function* (CDF). Its value at t=0.7 is the probability that a student, picked at random, will have a test score below 70%.

Some courses (in some schools, but not MIT) make rules to assign letter grades like:

Rule: if a student's score is:	letter grade
greater than mean + one standard deviation	A
greater than mean AND less than mean + standard dev	B
less than mean AND greather than mean – standard dev	C
less than mean – std dev AND greather than mean-2*std dev	D
otherwise	F

In such cases, you can use the cumulative distribution function to find out, for example, the probability that a student picked at random gets an A on a test. Or, you can find the probably that a student got a score of 85% or higher, etc. if you know the mean and standard deviation.

Normally, the CDF is listed from a table in a book, but here we are going to use numerical methods to calculate the CDF.

Using the same example (mean = 0.76 and standard deviation = 0.21), calculate the percentage of people who got a score within one standard deviation from the mean. Use Simpson's method to calculate the integral. You can find out how this method works on Lecture 20, pg 7. Simpson's method divides a region into equally sized pieces, each of length *h*. As you increase the number of pieces created, you increase the accuracy of the approximation. Keep increasing the number of pieces until the difference between one iteration and the next is less than 10^{-6}. Print out the number of iterations used and the percentage you calculated.

Be sure to write a `MathFunction` interface that your Gaussian class will `implement`.

Example output (with values) is shown below:

```
Number of iterations: 696
% of students +/- 1 std dev of mean: 68.3383811292024%
```

Turnin

Turnin Requirements

- Hardcopy and electronic copy of ALL source code (all .java files). Remember to comment your code, points will be taken off for insufficient comments.
- Place a comment with your name, username, section, TA's name, assignment number, and list of people with whom you have discussed the problem set on ALL files you submit.
- Do NOT turn in electronic or hardcopies of compiled byte code (.class files).

Electronic Turnin

Use *SecureFX* (or another secure ftp or secure shell program) to upload your problem set to your 1.00 homework locker.

Detailed instructions of how to upload are on the course website.

Since your problem set is due at the beginning of lecture, your uploaded problem should have a timestamp of no later than morning on the due date.

Penalties

- Missing Hardcopy: -10% off problem score if missing hardcopy.
- Missing Electronic Copy: -30% off problem score if missing electronic copy.
- Late Turnin: -30% off problem score if 1 day late. More than 1 day late = NO CREDIT.

If your problem set is late, or if a professor has granted you an extension in advance, do not submit a printed copy of your problem set.