A Method for Computing Internal Rates of Return in ExamView

Jeff Whitworth
University of Houston-Clear Lake

This paper shows how to overcome a difficulty commonly encountered when creating algorithmic problems that ask students to compute the internal rate of return (IRR) of a project. When building an algorithmic question, the instructor generally must enter a formula to calculate the correct answer from the problem's randomly generated input variables. IRR is defined as the discount rate for which the project's cash flows would have a net present value of zero. Unfortunately, no algebraic formula exists that expresses this rate as a function of the cash flows, and the software typically used to create algorithmic problems does not have a built-in IRR function as financial calculators and spreadsheets do. Fortunately, it is possible in ExamView (one of the more popular test generation programs) to obtain a very precise numerical solution for the IRR by using the Newton-Raphson approximation method. For two basic IRR problem types commonly given in corporate finance courses, I show how to create a user-defined function in ExamView that will carry out one iteration of the Newton-Raphson method and how to apply that function iteratively to compute the IRR. Solutions generated using this method are as precise as those obtained from financial calculators and spreadsheets.

INTRODUCTION

ExamView is a versatile software package used by instructors across many disciplines to create and manage banks of questions, and to build exams and other assessments from those questions. One important feature of ExamView – particularly for those who teach more quantitative courses – is that it allows the instructor to create *algorithmic* problems. In an algorithmic problem, one or more of the input variables are randomized so that each time a new version of the question is created, the software generates a new value for each variable within range(s) the instructor has specified. The instructor also enters a formula that ExamView will use to calculate the correct answer to the problem based on the input variables. For multiple choice problems, formulas may also be entered to calculate "distractors" (i.e. incorrect answer choices).

Creating algorithmic problems (as opposed to "static" problems without random variables) usually requires a little more work upfront, but in the long run, having a pool of algorithmic problems can be incredibly useful for quickly and easily creating multiple exam versions so that all students are tested on the same skills and concepts, but any two students are unlikely to be given exactly the same questions. Regardless of the course modality (i.e. in-person, online, or hybrid) or the scheduling of different sections, this makes it much more difficult for dishonest students to cheat by obtaining information from others.

Finance is certainly a "quantitative" business discipline where students are frequently given problems that require calculations. For most problems, it is fairly straightforward to write a formula in ExamView that will compute the answer in terms of the input variables. However, this is not always the case. For example, Whitworth (2019) notes that it is challenging to apply the Black-Scholes option pricing model in algorithmic problems because ExamView lacks a built-in function to calculate standard normal distribution probabilities. Another type of problem where this is not so simple involves the calculation of internal rates of return (IRRs).

In virtually every corporate finance class, students are expected to learn to compute the IRR of a capital budgeting project given its cash flows. Mathematically, IRR is defined as the discount rate that would cause the project to have a net present value (NPV) of zero. It is usually not possible to solve for this rate analytically, so there is no mathematical formula for computing a project's IRR from its cash flows. Because of this, students are normally expected to use their financial calculators, and all of the commonly used business calculators (e.g. TI BA II Plus, HP 10BII+) have built-in IRR functions for this purpose. Microsoft Excel also has an IRR function that can easily be used when students are asked to build a spreadsheet model. Unfortunately, while ExamView does feature over 100 built-in functions to assist in the creation of algorithmic problems, it lacks an IRR function. This is understandable since ExamView was not built specifically for finance, but it nevertheless presents a challenge for finance instructors who want to create algorithmic problems for the quantitative material they teach.

In this paper, I explain and demonstrate a method for overcoming this limitation of ExamView. While an analytic solution for IRR is not possible, a numerical solution is quite feasible. (Indeed, numerical procedures are

exactly what spreadsheets and financial calculators use to compute project IRRs.) In the next section, I explain the Newton-Raphson method, a powerful technique for finding numerical solutions to many equations that cannot be solved analytically, and show how it can be used iteratively to calculate a project's IRR. In the following section, I show how to set up a basic algorithmic IRR problem in ExamView and how to create a user-defined function that can be applied several times in succession to calculate the IRR very precisely. I then show how the algorithm can be modified when the stream of cash inflows is an annuity, as is often true for IRR problems given to students. The final section concludes.

THEORY

Numerical methods often can find accurate solutions to equations which cannot be solved using standard algebraic techniques. One such example in finance involves the calculation of internal rate of return (IRR) for a project with cash flows that occur over multiple years. The IRR is computed by finding the project cost of capital r such that

$$NPV = f(r) = \sum_{t=0}^{n} \frac{CF_t}{(1+r)^t} = 0,$$

where n is the number of years in the project's life and CF_t is the project's cash flow at the end of year t. For projects with only a single cash outflow at $t = 0$ and a single cash inflow at $t = n$, it is straightforward to solve for r. However, most capital budgeting projects that students are asked to evaluate have nonzero cash flows occurring over multiple years, and in those cases, solving for r is not so easy.

The Newton-Raphson method (hereafter simply referred to as Newton's method) is a numerical technique that can be applied to find the roots of most functions that are reasonably well-behaved (as the NPV function above is). The method begins with an initial guess r_0 and at each step uses information about the value and slope of the function to derive successively closer approximations of r. Specifically, given a guess r_i, Newton's method computes the next guess as

$$r_{i+1} = r_i - \frac{f(r_i)}{f'(r_i)},$$

where $f'(r_i)$ is the first derivative of the function at r_i. For the NPV function above, the first derivative is

$$f'(r) = \sum_{t=0}^{n} \frac{-t \cdot CF_t}{(1+r)^{t+1}}.$$

To see how this method can be applied, consider a project with an initial cost of \$100 (i.e. $CF_0 = -100$), and which produces cash inflows of \$30, \$50, and \$60 at the end of years 1, 2, and 3, respectively. Entering these cash flows into a financial calculator, we obtain an IRR of 16.79493614%. (In a problem like this, students are normally asked to give the IRR to two decimal places – e.g. 16.79% – but more digits are retained here to show just how precise the numerical method is.) For this project, suppose we begin with an initial guess of $r_0 = 8\%$ (or 0.08), which is much lower than the actual IRR. At this cost of capital, the project's NPV is

$$f(r_0) = -100 + \frac{30}{1.08} + \frac{50}{1.08^2} + \frac{60}{1.08^3} = 18.27465325,$$

and the slope of the NPV function is

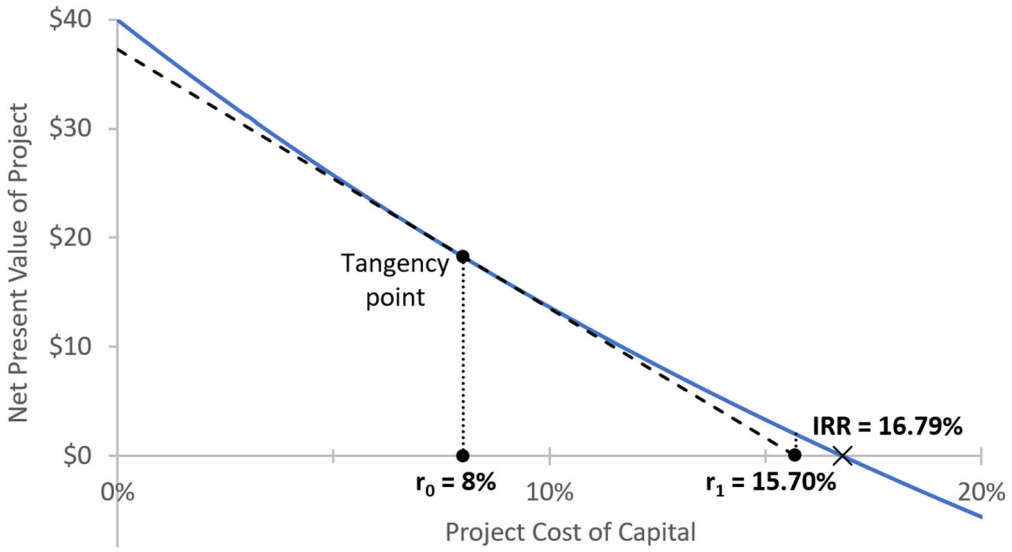
$$f'(r_0) = \frac{-1 \cdot 30}{1.08^{1+1}} + \frac{-2 \cdot 50}{1.08^{2+1}} + \frac{-3 \cdot 60}{1.08^{3+1}} = -237.40876221.$$

Therefore, the first iteration of the procedure produces an estimate of

$$r_1 = r_0 - \frac{f(r_0)}{f'(r_0)} = 0.08 - \frac{18.27465325}{-237.40876221} = 0.1569754793 = 15.69754793\%,$$

which is now only about one percentage point less than the project's IRR. As illustrated in Figure 1 below, one can apply the first step of Newton's method by drawing the tangent line to the NPV function at the initial guess $r_0 = 8\%$, and then finding an improved approximation by calculating where the tangent intersects the horizontal axis – in this case, at $r_1 \approx 15.70\%$:

Figure 1: Graphical Illustration of First Step of Newton's Method to Find IRR



If we were to zoom in on the portion of the graph from 15.70% to 16.79% and then continue this illustration, the next tangent line (at r_1) would be virtually indistinguishable from the NPV function itself since the slope of the graph changes very little between r_1 and the IRR. This means that the tangent line would intersect the horizontal axis at a value r_2 that is extremely close to the solution. Our calculations confirm this, as using r_1 in the second iteration gives us $f(r_1) = 2.024060408$, $f'(r_1) = -187.43712218$, and

$$r_2 = r_1 - \frac{f(r_1)}{f'(r_1)} = 0.1569754793 - \frac{2.024060408}{-187.43712218} = 0.1677740894 = 16.77740894\%.$$

This differs from the true IRR by less than 0.02%.

The next step of the algorithm (starting with r_2) gives us $r_3 = 16.79493166\%$, which is correct to five decimal places (when expressed as a percentage). Continuing this for one more iteration, we obtain $r_4 = 16.79493614\%$, which matches the financial calculator's answer exactly.

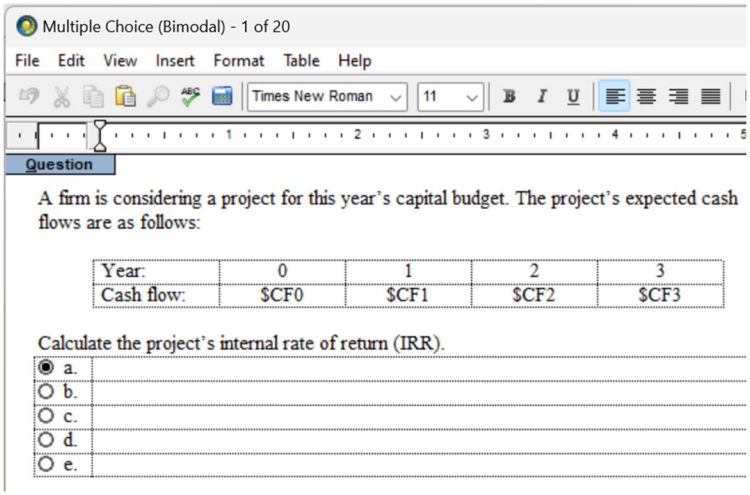
As this example demonstrates, Newton's method is a powerful technique capable of taking a very rough initial guess and zeroing in on an incredibly precise solution in just a few steps. Although Newton's method does not always converge for every function, it does work extremely well here because the NPV function is strictly downward sloping and convex for all $r > 0$. Given any positive value of $r_i <$ IRR, it is guaranteed not only that each successive estimate will be closer to the IRR, but also that the algorithm will converge faster as it gets closer to the solution. As explained in Burden et al. (2016), given certain properties (which the NPV function above satisfies), Newton's method converges at least quadratically, which means that the number of correct decimal places will approximately double with each iteration. This prediction is consistent with the example above.

One critical assumption here is that the project cash flow stream is normal; in other words, the project begins with a negative cash flow and then produces only positive cash flows after that. If the cash flow stream is nonnormal – i.e. containing multiple outflows separated by inflow(s) – the NPV function may no longer have the properties mentioned above, and Newton's method may either not converge at all, or it may converge to different solutions depending on the starting point.

APPLICATION IN EXAMVIEW

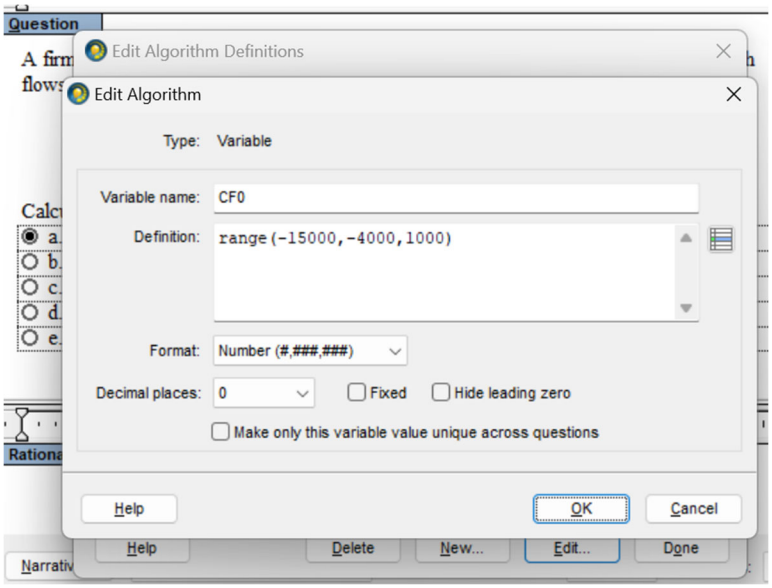
I now demonstrate how to build an algorithmic problem in ExamView where students are asked to find a project's internal rate of return (IRR). In doing so, I show how to program the software to calculate the correct answer to the problem, even though there is no algebraic formula for IRR. Figure 2 below shows the initial problem setup. In this case, students will be presented with a series of uneven annual cash flows for a 3-year capital budgeting project, and they will be asked to compute its IRR.

Figure 2: Initial Setup of an IRR Problem



For a problem to be algorithmic, it must contain one or more input variables whose values are randomly generated each time a new instance of the question is created. For this problem, the input variables are the project cash flows for years 0, 1, 2, and 3, respectively. To specify how these will be generated, we can simply press Control-D, or we can go to "Edit" on the menu shown above and then select "Algorithm Definitions". When the "Edit Algorithm Definitions" window appears, we click the "New" button and enter the variable name `CF0` (corresponding to the initial cash outflow of the project), as shown below:

Figure 3: Defining a Random Input Variable

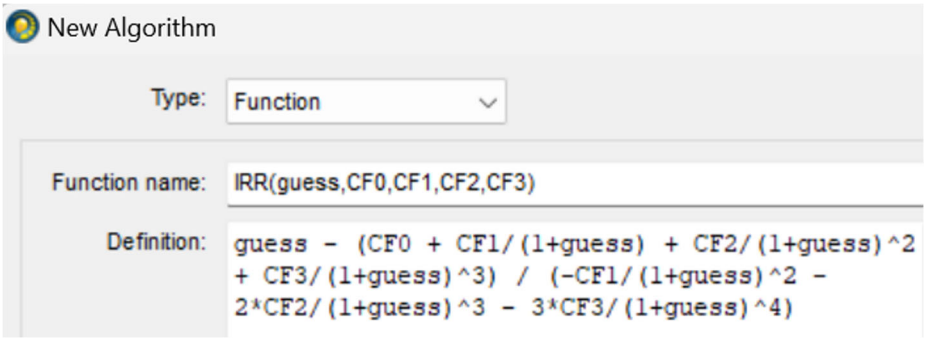


In the "Definition" area, entering `range(-15000,-4000,1000)` will generate a random number from –15000 to –4000 in increments of 1000. There is nothing special about these particular numbers; the lower and upper bounds of the range and the increment size can be changed to suit the instructor's preference. After specifying an appropriate format (in this case, a number with comma separators and no decimal places), we click the "OK" button to complete this variable definition.

Next, we click the "New" button again and enter the variable name `CF1` (the project cash flow at the end of year 1). We define this variable as `round(-CF0*range(.3,.5,.01),100)`, which will generate a first-year cash flow that is anywhere from 30 to 50 percent of the size of the initial investment, rounded to the nearest multiple of 100. After specifying the format and clicking "OK" again, we repeat this process for the variables `CF2` and `CF3`, defining them the exact same way. Note that even if the formulas entered for the three cash inflows are identical, the resulting *values* of these variables will differ because ExamView does an independent random draw each time it encounters the `range` function. As with the initial investment, this is not the only way the cash flows can be defined. However, this method generally results in project IRRs that are plausible – i.e. not extremely high or low.

Now we must tell the software how to compute the IRR from this set of randomly generated cash flows. This, of course, is the challenging part of the process since there is no algebraic formula for IRR. As explained in the previous section, though, we can begin with an initial guess and then apply Newton's method several times to obtain a series of closer and closer approximations. The first step in this process is to create a user-defined function that will execute one iteration of this algorithm. To do this, we click the "New" button in the "Edit Algorithm Definitions" window and then change the "Type" to "Function" via the drop-down list. We then enter `IRR(guess,CF0,CF1,CF2,CF3)` to specify the name of the function and the input variables the function will need to use (i.e. a guess at the solution for IRR, and the cash flows themselves). In accordance with the method explained in the previous section, we enter the function definition as shown below:

Figure 4: Creating a User-Defined Function to Execute One Step of Newton's Method



(Note that the spaces in the function definition above are for readability only and are not required by ExamView.) We then click "OK" to complete this step.

Next, we define the variable `r0`, the initial guess that the algorithm will start with. There are different ways to choose this value. One possible definition for `r0` is `((CF1+CF2+CF3)/(-CF0))^(1/2)-1`, which would be project's return if all of the cash inflows occurred in the second year. It is also possible to define `r0` as some fixed value, such as `0.08` (as was done in the previous section). As long as the initial guess is even remotely reasonable, the algorithm will converge to the correct solution after just a few steps. For the example in the previous section, even with an initial guess of 0% or 30%, the algorithm finds the IRR of 16.79493614% – correct to ten significant digits – after just four iterations. This is especially impressive given that a student answer of 16.79% would normally be accepted as precise enough.

After this, we use our newly defined function to find successively closer approximations of the project's IRR. We begin by defining the variable `r1` as `IRR(r0,CF0,CF1,CF2,CF3)`. This gives the first approximation of the solution after applying one iteration of Newton's method. We then use this result in the next step, defining another variable `r2` as `IRR(r1,CF0,CF1,CF2,CF3)`, which gives us a second, closer approximation. We continue creating variables in this manner until we reach `r5`, defined as `IRR(r4,CF0,CF1,CF2,CF3)`. It is certainly possible to continue creating more variables like this, executing as many iterations of Newton's method as desired,

and there is no harm in doing so. However, under the parameters specified for the cash flows above, five iterations should be more than sufficient.

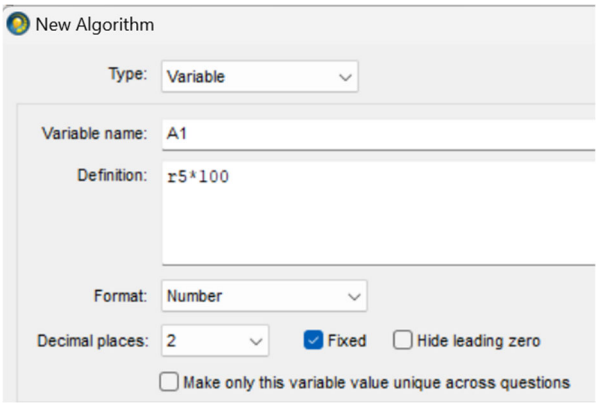
In any case, it may be prudent to include a condition verifying that the value of `r5` really is the project's IRR – that is, the discount rate that gives the project a zero NPV. To do this, click the "New" button again while in the "Edit Algorithm Definitions" window, and change the algorithm's "Type" to "Condition" using the drop-down menu. Then enter

```
abs(CF0 + CF1/(1+r5) + CF2/(1+r5)^2 + CF3/(1+r5)^3) < 0.01
```

to ensure that the project's NPV at a cost of capital of `r5` is effectively zero. It may also be desirable to require that the IRR be within a reasonable range. A second condition stating `r5>0 AND r5<0.30` would accomplish this. All conditions specified in this manner must evaluate to `TRUE`; otherwise, ExamView will generate another set of random variable values until it finds one that does satisfy all the conditions.

Having done this, we now define the five answer choices. In this case, `A1` will be the correct answer and is simply defined as `r5*100`. (It is necessary to multiply `r5` by 100 in order to display the final answer as a percentage.) Since this value will actually be displayed to students in the question, it is important to format it appropriately. For this problem, I would select the "Number" format with 2 decimal places for all answer choices. For consistency, I also select the "Fixed" checkbox to ensure that 2 decimal places will always be displayed, even if the last digit is a zero:

Figure 5: Creating and Formatting the Correct Answer Variable



The remaining variables `A2` through `A5` represent the distractors (i.e. the incorrect answer choices). Sometimes the instructor may want to define one or more distractors based on potential calculation errors a student could make because of an incomplete understanding of the problem. For example, `A2` could be defined as `((CF1+CF2+CF3)/(-CF0)-1)*100`, which incorrectly assumes that all cash inflows occur in the first year. `A3` might be defined as `(((CF1+CF2+CF3)/(-CF0))^(1/3)-1)*100`, which one could obtain by incorrectly treating all three cash inflows as if they occurred in the third year. Another approach is to simply generate random values that are close enough to the correct answer to be plausible. The remaining distractors `A4` and `A5` could be defined in this manner as `A1*range(0.7,1.3,0.01)`.

However the distractors are defined, it is important to be sure that they are sufficiently different from the correct answer so that a student would not obtain those answers because of a small rounding error. For example, if the correct answer is 16.79% as in the example from the previous section, it is probably a bad idea for one of the other choices to be 16.75% or 16.82%. To ensure this does not happen, we can create a new condition defined as follows:

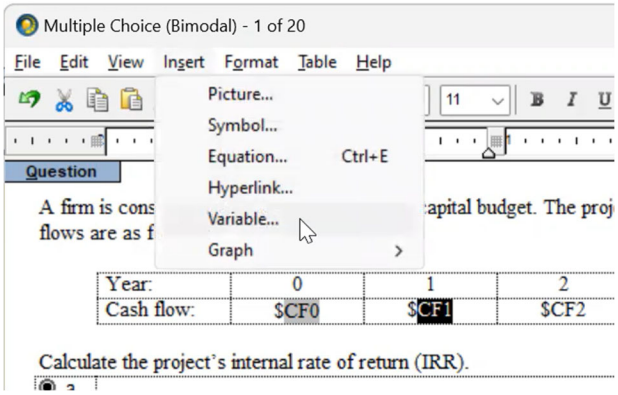
```
min(min(min(min(abs(A1-A2),abs(A1-A3)),min(abs(A1-A4),abs(A1-
A5))),min(min(abs(A2-A3),abs(A2-A4)),min(abs(A2-A5),abs(A3-
A4)))),min(abs(A3-A5),abs(A4-A5)))>0.10
```

This ensures that no two answer choices are within 0.10% of each other. The expression above is admittedly somewhat cumbersome because ExamView's `min` function cannot evaluate more than two arguments at a time. To save time in

the future, it is a good idea to save the text of this condition (except for the value `0.10` at the end) in an easily accessible file so that it can be copied and pasted when creating other algorithmic problems.

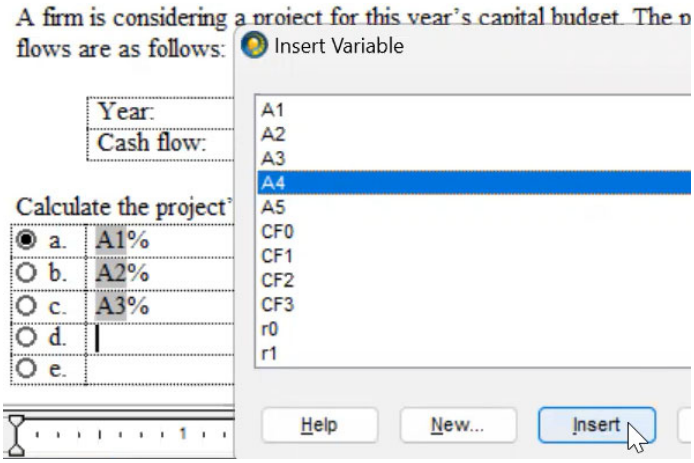
Having defined all of the variables and conditions needed for the problem, we now click "Done" to exit the "Edit Algorithm Definitions" window and return to the main problem window. The final step is to actually place the variables we have created into the problem, where appropriate. Since we have already typed the names of the variables `CF0` through `CF3` into the text of the problem, we only need to highlight each variable and then choose "Insert" and "Variable" from the menu, as illustrated below. As this is done for each cash flow, the variable name becomes highlighted in gray, indicating that the variable's *value* (not its name) will be displayed to students.

Figure 6: Inserting the Input Variables into the Problem



Similarly, we must go to each answer choice "a" through "e" and insert the variables `A1` through `A5` defined earlier. For example, we would first click in the space next to answer choice "a" and, as before, select "Insert" and "Variable" from the menu. In this case, since we did not previously type "A1" (the relevant variable name) into the problem, another window appears and asks us to select one of the defined variables. Of course, we select "A1" from the list and then click the "Insert" button. Again, the variable name that appears is highlighted in gray, indicating that its value, not its name, will be displayed to students. Since ExamView does not have an option to display variables in percentage format, it is necessary to type a percent sign after the variable if the answer is to be displayed in that manner. We continue doing this for the remaining answer choices, as illustrated below:

Figure 7: Inserting the Answer Choices into the Problem



After all five answer choices have been specified, it is necessary to specify the correct answer (in this case, "a") at the bottom of the question window. For the "Scramble" setting, selecting "All" allows the instructor to randomize the order of the answer choices when the question is later placed into a test. This is generally a good idea

unless there is a specific reason why one or more choices should appear in a specific position (e.g. a question where "None of the above" is a choice and naturally should be placed last). Finally, after pressing "Record" at the bottom of the question window, the problem appears in the question bank as shown below with a randomly generated set of cash flows, the correct answer, and several distractors that have been generated according to the formulas:

Figure 8: A Completed Algorithmic IRR Problem as Displayed in the Question Bank

1. A firm is considering a project for this year's capital budget. The project's expected cash flows are as follows:

Year:	0	1	2	3
Cash flow:	$–13,000	$4,400	$6,200	$5,700

Calculate the project's internal rate of return (IRR).
a. 11.69%
b. 25.38%
c. 7.83%
d. 12.51%
e. 10.87%

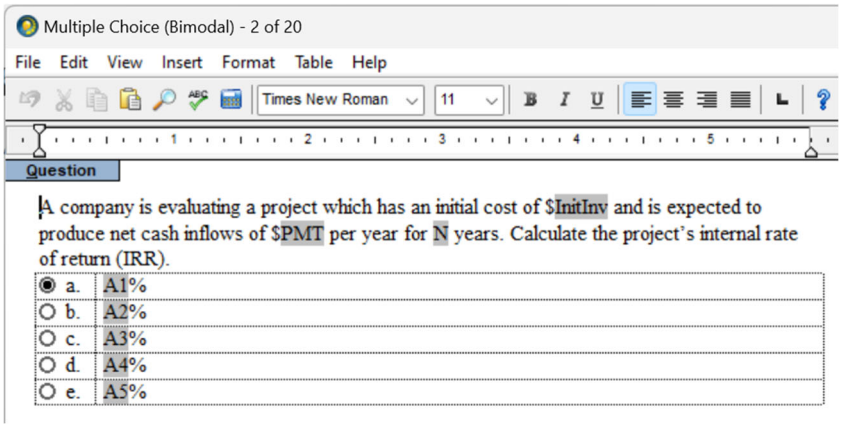
ANS: A

Note that at any time, the instructor can easily generate a new set of random values for the problem by pressing Control-K or by going to the menu bar and selecting "Question" and "Calculate Values", and then specifying either "Entire Question Bank" or "Selected Item". (New values are generated the same way within the Test Builder.)

THE CASE OF EQUAL CASH INFLOWS

Sometimes students are given a capital budgeting problem where (perhaps for simplicity's sake) the project has a stream of *equal* cash inflows for a number of years. An example of this problem type is shown below with the variables already defined and inserted:

Figure 9: An Algorithmic IRR Problem with Equal Cash Inflows



Here `InitInv` has been defined as `range(30000,90000,1000)`, and the number of years `N` in the project's life has been defined as `range(5,10,1)`, but of course these parameters can be modified to suit the instructor's preferences. The recurring cash inflow `PMT` is defined as `round(InitInv/N*range(1.25,2,.01),100)`. Since a yearly cash inflow of `InitInv/N` would result in an IRR of exactly zero, `InitInv/N` is multiplied by a randomly chosen factor from 1.25 to 2.00 to ensure that the IRR will be some positive number in the range typically seen for this type of problem. As used here, the `round` function causes `PMT` to be an exact multiple of 100.

In this case, the stream of project cash inflows is an ordinary annuity, so the NPV function becomes the present value formula for an annuity, minus the project's initial investment:

$$NPV = f(r) = PMT\left[\frac{1-(1+r)^{-n}}{r}\right] - InitInv$$

The derivative of this function is

$$f'(r) = PMT\left[\frac{rn(1+r)^{-n-1} - (1-(1+r)^{-n})}{r^2}\right].$$

We now define the function `IRR(guess,InitInv,PMT,N)` that will execute one iteration of Newton's method given an initial guess. Given the NPV function above and its derivative, the appropriate function definition is as follows:

```
max(0.0001, guess - (PMT*(1-(1+guess)^(-N))/guess - InitInv) /
        (PMT*(guess*N*(1+guess)^(-N-1) - (1-(1+guess)^(-N)))/guess^2))
```

Note that in this case, the `max` function is used to ensure that each estimate of *r* is strictly positive to avoid division by zero. (Requiring positive estimates for *r* is also appropriate here since PMT has been defined so that the project will have a positive IRR.) As in the previous example, the initial guess `r0` must be defined as something reasonable – perhaps 0.08 (but again, any remotely reasonable starting point will do, so long as it is greater than zero). After defining `r0`, we then define `r1` as `IRR(r0,InitInv,PMT,N)` and then continue to define `r2` through `r5` analogously using the `IRR` function. As before, the algorithm finds a very precise solution in only a few steps, but it is never a bad idea to verify that the value of `r5` calculated from this process really is the correct IRR. To do this, one can define the following condition:

```
abs(PMT*(1-(1+r5)^(-N))/r5 - InitInv) < 0.01
```

Next, the five answer choices must be defined. As before, the correct answer `A1` is just `r5*100` (formatted with two decimal places so that it can be displayed as a percentage). The distractors can be programmed in whatever way the instructor wishes, but here, `A2` is defined as `(PMT*N/InitInv-1)*100`, and `A3` is defined as `((PMT*N/InitInv)^(1/N)-1)*100` (incorrectly treating the cash inflows as if they all occurred in the first year or the final year, respectively). `A4` and `A5` are each defined as `A1*range(0.7,1.3,0.01)`, as before. Also, to prevent possible confusion, it is still a good idea to include the same condition from the previous section specifying that no two answer choices may be within 0.10% of each other. After completing this, specifying the correct answer, and clicking the "Record" button in the question window, an example of the newly created problem with randomly chosen values will appear in the main Question Bank Editor window, as shown below:

Figure 10: IRR Problem with Equal Cash Inflows as Displayed in the Question Bank

2. A company is evaluating a project which has an initial cost of $30,000 and is expected to produce net cash inflows of $5,300 per year for 9 years. Calculate the project's internal rate of return (IRR).
   a. 10.44%
   b. 59.00%
   c. 5.29%
   d. 12.21%
   e. 7.93%

   ANS: A

SUMMARY AND CONCLUSION

This paper has demonstrated how to create an algorithmic problem where students are asked to compute the IRR of a capital budgeting project. Algorithmic problems help to reinforce the integrity of course assessments by allowing an instructor to easily generate multiple versions of a problem so that the numbers presented (and therefore the final answer) will vary from student to student. This makes it much more difficult for a student to copy answers from a classmate during the same exam session. When testing is done asynchronously (as is common in online courses and in courses with multiple sections), it also means that any attempt by earlier test-takers to share answers with later test-takers is likely to be ineffective.

Normally, creating an algorithmic problem requires the instructor to enter a formula for the correct answer in terms of the input variables. This is straightforward, for example, when a student is asked to compute the future value of an annuity or to estimate a firm's cost of equity using the capital asset pricing model. However, it seems at first glance impossible in IRR problems because there is no algebraic method to find the zeros of the NPV function, so no single formula can express the IRR as a function of the project cash flows. Despite this difficulty, the NPV function and its first derivative are known, so it is possible to write a formula that will take an initial guess at the solution and apply one step of Newton's method to obtain a much closer estimate. If we are able to write multiple formulas to find the answer, then we can perform several iterations of Newton's method to obtain a very precise numerical solution. ExamView is a software package that gives instructors extensive freedom to define as many variables and/or formulas as needed to carry out the steps of a calculation. This paper illustrates in detail how to do this in ExamView for two types of IRR problems that are commonly seen in corporate finance courses. It is beyond the scope of this paper to review all test generation programs, but Cognero (a web-based resource available to instructors who use Cengage textbooks) has essentially the same capabilities as ExamView, so this method could be implemented in Cognero also with little to no modification. Once the algorithmic problem is created, the instructor can add it to a test in ExamView or Cognero and then create multiple versions of the test. For online assessments, the instructor could use either program to generate a set of many similar IRR problems, and then export that question set to the school's learning management system (e.g. Canvas, Blackboard) so that it can be incorporated into exams or quizzes there.

REFERENCES

Burden, R. L., Faires, J. D., & Burden, A. M. (2016). *Numerical Analysis* (10th ed., pp. 79-82). Cengage Learning.

Whitworth, J. (2019). Using Simpson's Rule to Evaluate the Normal Cumulative Distribution Function in an Algorithmic Option Pricing Problem. *Journal of Instructional Techniques in Finance*, *11*, 10-15.