Exercise: IRRs for Power Plant Options

Summary

This exercise examines the returns to an electric power producer for constructing a new power plant when a carbon tax may be imposed in the future. It builds on the NPV functions from the previous assignment and also adds an internal rate of return (IRR) calculation. In addition, it shows how modules are constructed in Python.

Input Data

There are four input files, and each gives the cash flows associated with building a particular type of power plant under a particular assumption about a carbon tax. Each file is in comma separated variable (CSV) format with a construction cost in year 0 and then profits in years 1-40. File **std_notax.csv** shows the cash flows for a standard natural gas power plant when no carbon tax is imposed. File **std_tax.csv** shows the same power plant when a carbon tax is imposed beginning in year 6: years 0-5 are identical to the first file but profits in years 6-40 are lower due to the tax. File **std_ev.csv** is the same plant but years 6-40 show the expected profit assuming the firm believes there is a 50% chance of the tax being imposed. Finally, file **ccs.csv** shows the cost and profit associated with building an advanced gas plant with carbon capture and sequestration (CCS) technology. CCS eliminates carbon emissions so the plant's profits are the same whether or not there is a carbon tax, and hence there is only one file.

Deliverables

Please submit two scripts, **npvtools.py** and **analyze.py**, and an updated copy of the Markdown file **results.md**. Each is described below.

Instructions

A. Script npvtools.py

1. This script will be a modified version of `npv.py` from the previous assignment. Please do the following to create it:

    1. Copy `npv.py` from your previous repository to this one and rename it `npvtools.py`.

    2. Trim the script so that it just contains the two functions, `read_cashflow()` and `npv()`, and any appropriate comments. Take out the part of the script that actually used the functions.

    3. Go to the definition of `read_cashflow()` and add a second, optional, argument called `splitter` that has a default value of `None`. `None` is a special keyword in Python that indicates a variable has no value.

    4. Now add type hints to the arguments. The hints for both `filename` and `splitter` should be `str`. For optional arguments, the type hint goes just after the argument name and before the equals sign. Finally, add a type hint for the return value to indicate that it's a `list`.

    5. Modify the call to `split()` later in the function to read `split(splitter)`. The function will now split on whatever character string is given as the `splitter` argument in the call. However, if `splitter` isn't given, it will use `split(None)`, which is exactly the same as `split()`.

    6. Add type hints to the `npv` function as well. The hint for `r` should be `float`, the hint for `cashflow` should be `list`, and the hint for the return value should be `float`.

    7. The result will be a Python module that can be used in other scripts via an `import` statement.

B. Script analyze.py

1. Now create a new script called `analyze.py` as follows:

    1. Include the line `import npvtools as nt` to import the module you built in the previous step. The import statement allows you to call the two functions adding the prefix `nt` followed by a dot and the name of the function: `nt.read_cashflow()` or `nt.npv()`.

    2. Include the line `import scipy.optimize as opt`.

    3. Define a new function called `analyze` that accepts one input parameter, `filename` and use `str` as the type hint. Set the type hint for the function's return value to `None`. Then create a function that does the following:

        1. Call `nt.read_cashflow()` to read the cash flow in `filename` into a variable called `cashflow`. It should use the new argument `splitter=","` to indicate that the file should be split on commas rather than spaces since these are CSV files.

        2. Call `nt.npv()` to compute the NPV of the cash flow at a 5% interest rate. The value will be in dollars. Divide it by 1e6 to convert it to millions of dollars. Store the result in a variable called `npv`.

        3. Compute the IRR using Newton's method via a call to `opt.newton()`. Store the result in a variable called `irr`. The first argument to `opt.newton()` should be `nt.npv` (your NPV function but without any parentheses or arguments), which `opt.newton` will try to set to 0, and the second should be 0.05, a starting guess of the IRR interest rate. You'll also need to pass two more arguments: `maxiter=20` to set the maximum number of iterations to 20 and `args=[cashflow]` to pass `cashflow` through to the NPV function (be sure not to overlook the square brackets around `cashflow`). The `opt.newton()` call will iterate over interest rates until the value calculated by `nt.npv()` is zero. It will return the interest rate it found, which will be the IRR.

        4. Multiply `irr` by 100 to convert it to a percentage.

        5. Print the name of the file, the NPV rounded to 1 digit after the decimal point, and the IRR, also rounded to 1 digit.

    4. In the main body of the script, after the end of the function definition, create a list called `files` consisting of four strings with the names of the input files.

    5. Add a `for` loop that loops over `files` using `filename` as the loop variable. The body of the loop should consist of a single line that calls `analyze()` on `filename`. You can ignore the value returned by `analyze()` since for the current assignment the printed information is all that's needed.

C. Markdown file results.md

1. Edit the Markdown file `results.md` and replace the `TBD` placeholders appropriate answers. Please note that this assignment focuses on the plants from the perspective of their owners or investors: you don't need to discuss externalities or social costs.

Submitting

Once you're happy with everything and have committed all of the changes to your local repository, please push the changes to GitHub. At that point, you're done: you have submitted your answer.

Tips

- Modules are an excellent way to reuse code and are the basis for a lot of Python's more advanced capabilities.
- Whenever you use an iterative numerical procedure like Newton's method, always include a maximum iteration count. Otherwise, if something is wrong with your function and it has no solution, the algorithm could iterate for a long time.