Even organisations and people that use a statistical package for hard-core data and econometric analysis might use spreadsheets for financial and economic scenario and evaluation models. There’s something about the ad hoc nature of financial models in particular that seems to tempt people towards Excel not only for prototyping but even for end products. This is a shame because it means missing out on the advantages of a programming language, such as ease of quality control, team-based development, and scaling up to running multiple simulations with different parameter choices.

In particular, economic evaluative analysis such as cost-benefit analysis and cost effectiveness analysis is commonly undertaken in spreadsheets. While there is software out there – see the book [Bayesian Cost-Effectiveness Analysis with the R package BCEA](https://www.springer.com/gp/book/9783319557168), for example – that can provide more analytic grunt, particularly when adding probabilistic elements and estimates of uncertainty to such analyses, I think it is fair to say that Excel still rules in this world..

In this post I’ll put forward a simple method of doing the number-crunching part of [cost-benefit analysis](https://en.wikipedia.org/wiki/Cost%E2%80%93benefit_analysis)(CBA) in R. The *hard* part of CBA is data collection and the valuing of costs and benefits, but I’m not going to touch on that here. I’m just looking at turning the data you’ve generated into estimates of the standard outputs of CBA, with appropriate sensitivity and uncertainty bounds that are awkward to produce in Excel.

Note that CBA is not the same as the cost-effectiveness analysis delivered in the BCEA R package mentioned above. Simply put, CBA requires the analyst to measure both benefits and costs on the same metric (monetary value). Cost-effectiveness analysis EITHER allows the benefits to be taken as given and compares the costs of methods of achieving those benefits OR, if allowing different levels of benefit to come into the equation, measures benefits with a different metric to the costs. For example, if costs are in dollars, benefits in cost-effectiveness analysis might be measured in Quality-Adjusted Life Years; it’s still necessary in this case to put all the benefits on a common measurement basis, it just doesn’t have to be the same metric as costs.

So the distinctive feature of CBA is that both costs and benefits are on the same monetary scale. This allows the two common key metrics in CBA:

* Net Present Value – the real value of benefits in today’s dollars minus the real value of costs (where the “real” value means future values are discounted by some arbitrarily chosen discount rate – the choice of which makes a big difference to how much the analysis values either the short term or the long term)
* Internal Rate of Return – the discount rate at which benefits break even with costs

A nice feature of the Internal Rate of Return as a measure is that it avoids the omnipresent [arguments about which discount rate to choose](https://grattan.edu.au/wp-content/uploads/2018/02/900-unfreezing-discount-rates.pdf). You calculate the discount rate at which you would break even, and if that is higher than your cost of capital, and higher than the rate of return of other uses of capital, then it is worth investing in. However, there’s no closed solution to calculate the Internal Rate of Return, which makes it “harder” to calculate and perhaps to explain, so it is perhaps less commonly presented than Net Present Value (casual observation only, I haven’t collected data on this). You can in fact use Excel’s solver functionality to estimate Internal Rate of Return, but it’s a step up in sophistication from general spreadsheet work, liable to human error, and (in my opinion) is going against the grain of how spreadsheets work, by reducing the instant flexibility of calculations you usually get.

My approach to implementing CBA is to build in uncertainty from the ground up; allowing each parameter to be treated as a random variable. All the key analytical tasks will be separated into functions, which is good for maintainability, scaling up complexity, and efficient adaption to other cases.

**Calculating net present value**

Let’s start with a generic function to calculate the net present value of a stream of costs and benefits, given an arbitrary discount rate. Discount rates are normally described in percentage terms. 7% is a commonly chosen one, required by the [Australian Department of Prime Minister and Cabinet](https://www.pmc.gov.au/sites/default/files/publications/cosst-benefit-analysis.docx) for regulatory interventions for instance – although why this should be the case given years of inflation and interest rates well below that level I’m not sure. Or for another example, see this [guidance from the New Zealand Treasury](https://treasury.govt.nz/information-and-services/state-sector-leadership/guidance/financial-reporting-policies-and-guidance/discount-rates), specifying 6.0% as the default, 4.0% to be used for office and accommodation buildings, and 7.0% for ICT-related investments.

So here’s an R function for estimating the net present value of an investment, given two equally-lengthed vectors of annual costs and benefits (in current prices) and a discount rate:

#' Net present value

#'

#' @param discount Discount rate, as a percentage (eg "7" means 7%)

#' @param cost Vector of annual costs

#' @param benefit Vector of annual benefits

#' @param return\_abs Whether or not to return the absolute value of the net present value (only likely to be

#' used if you are using this function as part of an internal rate of return calculation)

#' @details for a given stream of costs, benefits, number of years and discount rate, return the net

#' present value (discounted benefits minus discounted costs)

net\_present\_value <- function(discount, cost, benefit, return\_abs = FALSE){

if (length(discount) != 1 | class(discount) != "numeric"){

stop("discount should be a single number")

}

if(length(benefit) != length(cost)){

stop("cost and benefit should be equally-lengthed numeric vectors")

}

# convert discount percentage into a multiplier

discount\_m <- 1 - discount / 100

# number of years, other than year zero, which get discounted

m <- length(benefit) - 1

npv <- sum(benefit \* discount\_m ^ (0:m)) - sum(cost \* discount\_m ^ (0:m))

if(return\_abs){

npv <- abs(npv)

}

return(npv)

}

In action, here we can use it to calculate the net present value of an investment that costs $1,000,000 in its first year and $100 per year subsequently, and returns a benefit stream valued at $70,000 per year. I’ve done this with two different discount rates:

> costs <- c(1000000, rep(100, 29))

> benefits <- rep(70000, 30)

> net\_present\_value(discount = 7, cost = costs, benefit = benefits)

[1] -114534.1

> net\_present\_value(discount = 4, cost = costs, benefit = benefits)

[1] 234083.8

So we see that depending on the discount rate, this same set of costs and benefits could be either minus $115k or positive $234k. If we followed New Zealand Treasury guidance, this investment would be worthwhile if it for an office building (discount rate of 4%) but not if it were in ICT (discount rate of 7%).

One of the key reasons for using R (and in particular, functions) for this sort of analysis is how easy it makes it to explore assumptions systematically and at scale. In contrast, it is common for “sensitivity analysis” that uses Excel to laboriously run the CBA for a mere handful different scenarios. This is as tedious and error prone as you can imagine, and difficult to generalise to work for thousands of scenarios rather than just one or two.

As an example, with R, instead of saying that we’re sure the ongoing costs will be $100 per year and the benefits $70,000 per year, we can easily make them both uncertain estimates with Gamma distributions (hence guaranteed to be positive) with $100 and $70,000 as the expected values. Maybe the benefits will be as low as $10,000 per year, maybe as high as $200,000; and similar for the costs around their central value of $100 per year.

So here’s a little simulation, still only using our one function so far:

#----------with uncertain assumptions----------

nsim <- 10000

set.seed(123)

sims <- data.frame(

benefit\_rand = rgamma(nsim, 7, 0.0001),

cost\_rand = rgamma(nsim, 10, 0.1)

)

for(i in 1:nsim){

sims[i, "npv"] <- net\_present\_value(discount = 4,

cost = c(1000000, rep(sims[i, "cost\_rand"], 29)),

benefit = rep(sims[i, "benefit\_rand"], 30))

}

sims %>%

ggplot(aes(x = npv)) +

geom\_density(fill = "steelblue", alpha = 0.1) +

scale\_x\_continuous(label = dollar) +

labs(x = "Net present value",

title = "Impact of randomness on net present value",

subtitle = "Investment with initial cost of $1m, mean annual costs of around $100,\nand mean annual benefits of around $70,000\nAnnual costs and benefits are Gamma-distributed random variables")

The 80% credibility interval for the net present value in this case is that it is somewhere between minus $320k and positive $825k, but probably above zero. This is a much bigger range of uncertainty than we would like of course, but it captures the generous uncertainty of our assumptions.

**Relationship of discount rate to internal rate of return**

The functional approach to R programming also usefully facilitates visualising the relationship between discount rate and internal rate of return. Let’s go back to a fixed (non-random) value for the costs and benefits, and calculate the net present value for a whole range of discount rates:

You pays your money, and you takes your choice. In this case we can see that the net present value line hits the negative area with a discount value of around 6%. If your discount rate or your cost of capital is higher than that, that’s a “no” for the project; if it’s lower, then it makes sense to undertake the project.

Code for that chart:

discounts <- seq(from = 0, to = 10, length.out = 100)

npvs <- sapply(discounts, net\_present\_value, cost = costs, benefit = benefits)

tibble(npv = npvs,

discount = discounts) %>%

ggplot(aes(x = discount, y = npvs)) +

annotate(geom = "rect", xmin = -Inf, xmax = Inf, ymax = 0, ymin = -Inf, fill = "red", alpha = 0.2) +

geom\_line() +

scale\_y\_continuous(label = dollar) +

labs(x = "Discount rate",

y = "Net Present Value",

title = "Impact of discount rate on net present value",

subtitle = "Investment with initial cost of $1m, annual costs of $100, and annual benefits of $70,000")

We can use the R optimise() function just like the Excel “solver” to calculate the actual value at which the net present value hits zero. For convenience (remember, this is all about why R is easier than Excel for this sort of thing), I’m going to wrap this in my own internal\_rate\_return() function:

internal\_rate\_return <- function(cost, benefit, interval = c(-25, 25)){

opt <- optimise(net\_present\_value,

interval = interval,

cost = cost,

benefit = benefit,

return\_abs = TRUE)

return(opt$minimum)

}

internal\_rate\_return(costs, benefits)

So our internal rate of return turns out to be 5.84. Because we’ve made a function out of this internal rate of return calculation, it’s just as easy to calculate this for our 10,000 simulations as it was the NPV. Try doing that in Excel (no, don’t really – I’m sure it can be done, my point is just that it would be fiddly).

Produced with this code:

#-------------IRR with random costs and benefits--------

for(i in 1:nsim){

sims[i, "irr"] <- internal\_rate\_return(cost = c(1000000, rep(sims[i, "cost\_rand"], 29)),

benefit = rep(sims[i, "benefit\_rand"], 30))

}

sims %>%

ggplot(aes(x = irr / 100)) +

geom\_density(fill = "steelblue", alpha = 0.1) +

scale\_x\_continuous(label = percent) +

labs(x = "Internal rate of return",

title = "Impact of randomness on internal rate of return",

subtitle = "Investment with initial cost of $1m, mean annual costs of around $100, and mean annual benefits of around $70,000\nAnnual costs and benefits are Gamma-distributed random variables")

**Functionalising the whole process**

So far we’ve worked with a very basic set of costs and benefits and I’ve examined uncertainty or scenarios in only an ad hoc way. In practice, any real-life example will be more complex than this. If we leave our costs and benefits as a bunch of ad hoc vectors and data frames lying around we might as well be working in Excel.

To handle more complex scenarios and still be able to simulate many results with uncertainty built in, we will need to abstract the generation of our simulation costs and benefits into its own function. This will take a set of user-provided parameters and calculate the net present value and internal rate of return for just on simulation with those parameters, including whatever randomness is dictated by them. We’ll then be able to wrap that function in another, which will call the single-analysis function multiple times and collect the results for analysis.

In the next chunk of code I’ll introduce these functions, and a little more complexity in the actual model. Key additions include:

* the introduction of ad hoc costs, which happen with a given probability (10% chance in any one year in the example below) in any year other than year zero, and have a random cost when that happens. This is the sort of uncertainty that is handled particularly badly by the more manual models but which is commonplace in real life.
* benefits are a product of the number of new customers and their spend. The customers goes up in year zero by a level shift, and then grows steadily from that point on; the average spend is a random variable.

Obviously this is just scratching the surface, and in practice any useful CBA model will be more complex than this. But the approach is easy to extend, and importantly is easy to document and quality control. The end result we are aiming for is charts like these:

#' Cost benefit analysis of a single state of the world

cba\_single <- function(

initial\_cost,

initial\_cost\_se = initial\_cost \* generic\_uncertainty,

ongoing\_cost,

ongoing\_cost\_se = ongoing\_cost \* generic\_uncertainty,

prob\_ad\_hoc\_cost,

ad\_hoc\_cost,

ad\_hoc\_cost\_se = ad\_hoc\_cost \* generic\_uncertainty,

customer\_level\_shift,

customer\_level\_shift\_se = customer\_level\_shift \* generic\_uncertainty,

customer\_growth,

customer\_growth\_se = customer\_growth \* generic\_uncertainty,

customer\_spend,

customer\_spend\_se = customer\_spend \* generic\_uncertainty,

end\_year = 30,

discount = 7,

generic\_uncertainty = 0.1

) {

# randomness

initial\_cost\_this <- rnorm(1, initial\_cost, initial\_cost\_se)

ongoing\_cost\_this <- rnorm(end\_year, ongoing\_cost, ongoing\_cost\_se)

ad\_hoc\_cost\_this <- rbinom(end\_year, 1, prob = prob\_ad\_hoc\_cost) \*

rnorm(end\_year, ad\_hoc\_cost, ad\_hoc\_cost\_se)

customer\_level\_shift\_this = rnorm(1, customer\_level\_shift, customer\_level\_shift\_se)

customer\_growth\_this = rnorm(1, customer\_growth, customer\_growth\_se)

customer\_spend\_this = rnorm(1, customer\_spend, customer\_spend\_se)

costs\_current <- c(initial\_cost\_this, ongoing\_cost\_this + ad\_hoc\_cost\_this)

customers <- customer\_level\_shift\_this \* (1 + customer\_growth\_this) ^ (0:end\_year)

benefits\_current <- customers \* customer\_spend\_this

npv <- net\_present\_value(discount, cost = costs\_current, benefit = benefits\_current)

irr <- internal\_rate\_return(cost = costs\_current, benefit = benefits\_current)

dm <- 1 - discount / 100

costs\_discounted <- costs\_current \* dm ^ (0:end\_year)

benefits\_discounted <- benefits\_current \* dm ^ (0:end\_year)

time\_series\_output <- tibble(

'Costs in current prices' = costs\_current,

'Benefits in current prices' = benefits\_current,

'Discounted costs' = costs\_discounted,

'Discounted benefits' = benefits\_discounted,

'Cumulative discounted costs' = cumsum(costs\_discounted),

'Cumulative discounted benefits' = cumsum(benefits\_discounted),

'Cumulative net value' = cumsum(benefits\_discounted - costs\_discounted),

year = 0:end\_year

)

return(list(

'Net Present Value' = npv,

'Internal Rate of Return' = irr,

time\_series\_output = time\_series\_output

))

}

# Example usage of the single CBA analysis; performs one run (with random results) for a given

# set of parameters

set.seed(321)

cba\_single(

initial\_cost = 1000000,

ongoing\_cost = 10000,

prob\_ad\_hoc\_cost = 0.1,

ad\_hoc\_cost = 100000,

customer\_level\_shift = 1000,

customer\_growth = 0.02,

customer\_spend = 100

)

#' Cost benefit analysis of many simulations with a single set of parameters

#'

#' @param nsim number of simulations to run

#' @param seed random seed to fix so results are reproducible

#' @param ... other parameters passed to \code{cba\_single}

cba\_multi <- function(nsim = 1000, seed = 123, ...){

output\_simple <- tibble('Net Present Value' = numeric(nsim),

'Internal Rate of Return' = numeric(nsim))

output\_complex\_l <- list()

set.seed(seed)

for(i in 1:nsim){

analysis <- cba\_single(...)

output\_simple[i, ] <- c(analysis$'Net Present Value', analysis$'Internal Rate of Return')

tmp <- analysis$time\_series\_output

tmp$id <- i

output\_complex\_l[[i]] <- tmp

}

output\_complex <- do.call(rbind, output\_complex\_l) %>%

gather(variable, value, -id, -year) %>%

group\_by(variable, year) %>%

summarise(mid = mean(value),

lower = quantile(value, 0.1),

upper = quantile(value, 0.9))

return(list(

output\_simple = output\_simple,

output\_complex = output\_complex)

)

}

# Example run

results <- cba\_multi(

initial\_cost = 1000000,

ongoing\_cost = 10000,

prob\_ad\_hoc\_cost = 0.1,

ad\_hoc\_cost = 100000,

customer\_level\_shift = 1000,

customer\_growth = 0.02,

customer\_spend = 100

)

# Summary plot

results$output\_simple %>%

gather(variable, value) %>%

ggplot(aes(x = value)) +

facet\_wrap(~variable, scales = "free") +

geom\_density(fill = "steelblue", alpha = 0.1) +

scale\_x\_continuous(label= comma) +

labs(title = "Key summary results from CBA of a hypothetical project, with uncertainty built in")

# Costs and benefits plot

results$output\_complex %>%

filter(!grepl("^Cumulative", variable)) %>%

ggplot(aes(x = year)) +

facet\_wrap(~variable, scales = "free\_y") +

geom\_ribbon(aes(ymin = lower, ymax = upper), alpha = 0.2) +

geom\_line(aes(y = mid)) +

scale\_y\_continuous(label = dollar) +

labs(x = "Year",

y = "Value") +

labs(title = "Key time series results from CBA of a hypothetical project, with uncertainty built in",

subtitle = "80% credibility intervals shown")

# Cumulative plot

results$output\_complex %>%

filter(grepl("^Cumulative", variable)) %>%

ggplot(aes(x = year)) +

facet\_wrap(~variable, scales = "fixed") +

geom\_ribbon(aes(ymin = lower, ymax = upper), alpha = 0.2) +

geom\_line(aes(y = mid)) +

scale\_y\_continuous(label = dollar) +

labs(x = "Year",

y = "Value") +

labs(title = "Cumulative costs and benefits from CBA of a hypothetical project, with uncertainty built in",

subtitle = "80% credibility intervals shown")

Close readers will observe that I’ve taken some shortcuts here, particularly by using a “generic uncertainty” parameter which by default gives each of my random parameters a standard deviation that is 10% of its expected value; and that all the random variables are Normally distributed. Obviously these are parts of the model that can be made more specific if better information is available, for example the the standard errors of estimates of willingness-to-pay from a survey could be used as the actual standard deviation of the relevant parameter in an appropriate model.

**Conclusion**

I’ve tried to at least begin to show:

1. how easy it is to do cost-benefit analysis calculations in R
2. how using R makes it much easier to simulate uncertainty by performing many runs with a given set of parameters and defined randomness
3. how building your model in an R function like cba\_single() makes it easier to quality control and see at a glance exactly what the model is doing.

The analysis is done with four functions:

* net\_present\_value() and internal\_rate\_return() apply to any analysis
* cba\_single() is the guts of the model, and estimates a single set of costs and benefits, with defined areas of randomness, for a given set of parameters. It will vary greatly depending on the task at hand.
* cba\_multi() is a wrapper to run cba\_single() many times, with the same set of parameters, and collect the results for presentation and discussion.

Happy cost-benefit analysing!