Calibration - a simple 'bucket' rainfall-runoff model

Contents

1 Introduction

Calibrating (aka *fitting*, or sometimes *tuning*) a model refers to the process of finding the parameter values that get the model's output to achieve some criterion, usually a measure of how much the model's output differs from some data. Calibration therefore depends on 1) the data, and 2) the definition of "differs". We'll examine both of these in more detail.

A simple example is the model of how vapor pressure changes with temperature (this is the *e0* function used in some of your PET functions). The equation (i.e. model) is

$$e_s = a * exp\left(\frac{bT}{c+T}\right)$$

It's tempting to think that a clean-looking equation like this one is more of a "law" than a model, but it's definitely a model. It's quite (but not perfectly) accurate within about -40 and +50 degrees C, and it approximates a more general relationship called the Clausius Clapeyron equation. Even the latter, though derived from thermodynamic first principles, is itself a simplified representation of extremely complicated modelcular-level processes. Many other models for this relationship have been proposed, but this one has emerged as the winner.

The model's functional form says that air can hold exponentially more water vapor as T increases, but the exact nature of that relationship is defined by the parameters a, b, and c. To get these, we need data. The values of $a = .6108, b = 17.27, c = 237.3$ have been determined to provide a very good fit to a large amount of real-world observtaions, but they are not universal physical constants (like, say, the speed of light). Rather, they arose from some calibration procedure that tried to make predictions from the proposed model best match observed pressure-temperature data.

1.1 The bucket rainfall-runoff model

In this exercise, you'll investigate some aspects of calibration by using a given time series of P and PET to build and calibrate a simple runoff model. Runoff is the amount of precipitation that does not infiltrate the soil, but instead "runs off" down hillslopes and into river networks. Good rainfall-to-runoff (RR) models are

an important part of understanding the terrestrial and global water cycles, and constitute a huge body of hydrology research.

A basic version is the "bucket" model, which treats the soil as a bucket with a limited capacity to hold water. "Ins" to the bucket include precipitation and irrigation (thought we'll ignore the latter), and the "outs" are evaporation and transpiration, which together are called evapotranspiration. If the ins exceed the outs, the difference is the amount of runoff. See below for a graphical depiction.

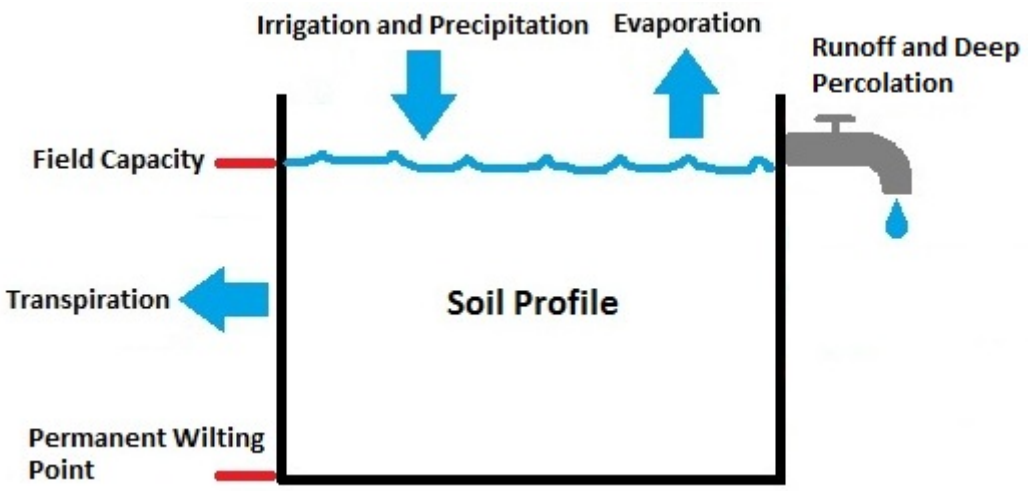


Figure 1: Credit: growingtheconversation.blogspot.com

1.1.1 Upper and lower limits of the bucket

The bucket's maximum capcacity is often defined as the "field capacity", which is the amount of water that the ground can hold and still drain from gravity. The bucket is considered empty at its "wilting point". This doesn't mean there's zero water in the soil, just that whatever little water there is clings so tightly to soil particles that it can't be extracted by plant roots.

1.1.2 Bucket model 2.0

We can make the bucket model slightly more sophisticated by considering two new factors:

1) Even dry soil can only absorb so much water so quickly, which means that not all precipitation goes directly into the bucket. The closer the bucket is to full capacity, the more precipitation will skip the bucket and go straight into runoff.

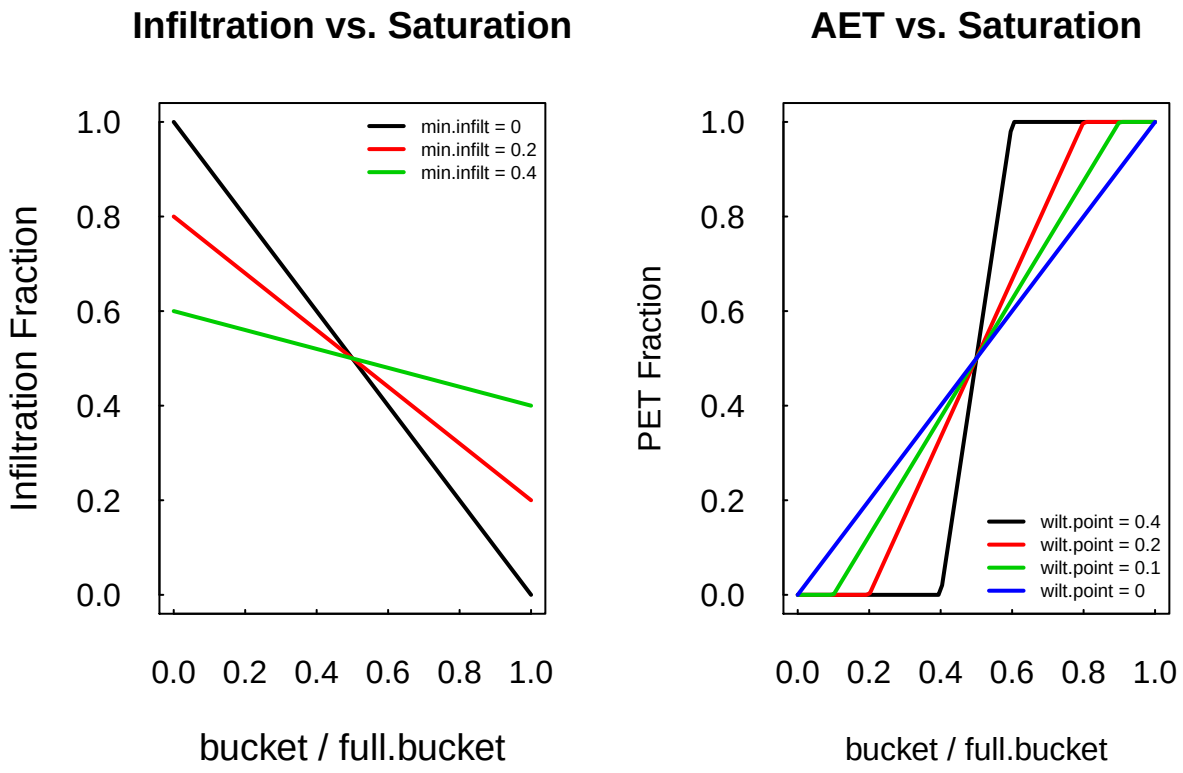
2) There's a difference between *potential* evapotranspiration (PET) and *actual* evapotranspiration (AET). The former only reflects how much moisture the atmosphere is trying to draw out, while the latter should also reflect how much the soil has to give. In completely dry soil, there's no water to lose, so AET should be zero, no matter how high PET is.

We'll incorporate these ideas into a more complex, 3-parameter bucket model with the following assumptions:

- Infiltration fraction decreases linearly with saturation.
- The minimum infiltration fraction ranges between 0 and .5, and maximum infiltration equals (1 - minimum infiltration).

- The fraction of PET that ends up as AET is a piecewise linear function of saturation, equal to 0 at the wilting point and 1 (1 - wilting point).

These assumptions mean that the infiltration and AET fractions each require just one parameter.



1.2 Assignment overview

In the in-class portion of the assignment, you'll take the simpler version of the bucket model and use an optimization routine to find its parameters (aka calibrate it). For the take-home portion, you'll code the more sophisticated bucket model yourself, and look at the impacts of different loss functions and different levels of noise on the calibration results.

1.3 Code for each bucket model

We're providing the following code to define each bucket model, but you should step through it and make sure you understand how it works!

```
# 1-parameter model (bucket capacity)
bucket1 = function(pars,P,PET) {
  full.bucket = pars[1]
  n = length(P)
  runoff = rep(0,n)
  bucket = min(full.bucket,max(P[1]-PET[1],0))
  for (m in 2:n) {
```

```
    infilt = P[m]
    AET = PET[m]
    bucket = bucket + P[m] - PET[m]
    if (bucket > full.bucket) {
      runoff[m] = bucket - full.bucket
      bucket = full.bucket
    } else if (bucket<0) {
      bucket=0
    }
  }
  runoff
}


# 3-parameter model (bucket capacity, varying P scaling factor, varying PET scaling factor)
bucket3 = function(pars,P,PET) {

  full.bucket = pars[1]
  min.infilt = pars[2]
  wilt.point = pars[3]

  bucket = min(full.bucket,max(P[1]-PET[1],0)) # initialize bucket value
  n = length(P)
  AET = runoff = rep(0,n)
  for (m in 2:n) {

    beta = bucket/full.bucket # saturation level
    infilt.frac = (1-min.infilt) + beta*(2*min.infilt-1)
    aet.frac = max(0, min(1,(beta-wilt.point)/(1-2*wilt.point)))

    infilt = P[m]*infilt.frac
    runoff[m] = P[m] - infilt
    AET = PET[m]*aet.frac
    bucket = bucket + infilt - AET

    if (bucket > full.bucket) {
      runoff[m] = runoff[m] + bucket - full.bucket
      bucket = full.bucket
    } else if (bucket<0) {
      bucket=0
    }
  }
  runoff
}
```

2 In-class Exercises:

2.1 Get acquainted with the `bucket()` function

Go through this function line by line and make sure you understand each step. Ask questions! Plot the output of `bucket()` for a few different parameter sets. Use `lines()` with `type="s"` to view several different

time series on the same plot. Does it make sense why changing a given parameter has the effect you see? Does the output seem more sensitive to some parameters than others?

2.2 Using the `optim()` function

2.2.1 Inputs

The `optim()` function needs these arguments:

1) A vector whose values are an initial guess of the values of the parameters we're trying to find (in this case, a vector like `c(150,.5,.5)`, starting guesses for the values of full.bucket, infilt.frac, and aet.frac, respectively). This vector *must* be the first argument of the loss function (see #2 below).
2) The loss function whose value we're trying to minimize. This could be, for example, a function that calculates the output of `bucket(params,P,PET)` for a given set of `params`, then returns the root mean squared error (or some other metric) between the predicted and observed runoff.
3) Any other arguments that the loss function needs. The loss function's first argument, as we've already said, needs to be the vector of parameters. If it has any other arguments (and ours will: P and PET), these come immediately after (1) and (2) in `optim()`'s argument list. Also importantly, their names in the call to `optim()` must be exactly the same as their names in the loss function.
4) Optional arguments specifying algorithm details, such as which optimization routine to use (which you should set as `method="L-BFGS-B"`), and the lower and upper bounds of the parameters.

2.2.2 Outputs

The first two elements of the list that `optim` returns are the ones we care most about. The first (`par`) contains the parameter values that the minimization settled on. The second (`value`) is the final value of the loss function.

How optim() produced them: Basically, what happens is that `optim()` calls the loss function with the initial set of parameters (the first argument) and looks at its resulting value (the quantity we're trying to minimize, also called the "objective value"; for now this will be the RMSE). Then it calls the loss function again, after tweaking the initial parameter values. Then again and again, each time changing the parameters, until it can't find a lower objective value. The secret sauce is in how it decides to change the parameters each time.

Question: Why do we need some fancy algorithm to keep changing the parameters? Why not just loop through all reasonably possible values of all parameters, and take the ones resulting in the lowest loss function value?

2.2.3 Try it yourself

1) Load `runoff.Rdata`, which contains time series of P, PET. Run `bucket()` with these P and PET, and the parameter vector `c(160,.53,.26)`. We'll consider the output of this run the "observed" runoff values.
2) Now write a function that calls `bucket()` with arbitrary parameters and calculates the root mean squared error between `bucket()`'s output and these observed values.
3) Finally, feed all the necessary arguments to `optim()` and see if `optim()` can recover the actual parameter values (`c(160,.53,.26)`).
4) Try a few different vectors of initial parameter guesses. Does `optim()` give the same results every time? What about if you change the loss fuction to calculate the mean of absolute errors rather than squared errors?

3 Take-home Exercises

1) The `runoff.Rdata` file also includes a sample runoff time series. If you plot it, you'll notice that it's similar but not exactly like the runoff time series you produced in the in-class portion. The reason is that it was generated by `bucket3()`, but with different parameters than the ones we used in class. Use `optim()` with a root-mean-squared-error (RMSE) loss function to find the parameters of this "true" model (bucket3). Use the `lower.bound` and `upper.bound` arguments to ensure the parameter search doesn't include values of `min.infilt` or `wilt.point` that are inconsistent with the above functions. Try several different initial parameter vectors and report the final parameters that `optim()` finds for each. What do you think the correct parameters are, and how sensitive is this procedure to the initial parameter guess?

2) Re-calibrate the model with a loss function that takes the mean of absolute (rather than squared) errors. Explain how you expect the absolute loss function to behave differently. Plot the resulting time series of predicted runoff for both loss functions, and discuss whether the differences make sense. Make sure your initial guesses for parameters are the same for both loss functions, so you are comparing the differences due to loss functions and not initial guesses.

3) In reality, we'll never have perfect knowledge of P and PET. We can simulate the effect of measurement error by creating new, noisy versions of P and PET by adding draws from a normal distribution (`rnorm()`) to them. Write a `for` loop in which the standard deviation of the values generated by `rnorm()` grows at each iteration. Start at 10% of the original P and PET's respective standard deviations, and end at 200%. Note that some of the new, noisy values of P and PET might be less than zero (since we might draw a few highly negative values from `rnorm`). Set these values to zero. For each noise multiplier run the optim with corresponding noisy P and PET and save the estimated parameter values. Repeat this procedure 10 times to get 10 time series of parameter values vs. the noise multiplier. Plot them all in a single figure.

4) Now assume instead that the dependent variable, runoff, is measured with noise. Add increasing amounts of noise to the runoff time series, and make a figure analagous to the one in exercise (3). Comparing these two figures, what can you say about the relative sensitivity of each model to noise in the P and PET vs. noise in runoff?