Chapter 2: Water supply reservoir simulation basics

Manual for R package `reservoir`

In this chapter:

- *Simulate a simple water supply reservoir*
- *Implement a depth-storage-area function and account for evaporation losses*

The `simRes` funcion

Recall from Chapter 1 the simple mass balance relationship used to model reservoirs:

$S_{t+1} = S_t + Q_t - R_t - L_t$

subject to:

$0 \leq R_t \leq \min(S_t + Q_t - L_t, R_{max})$

$0 \leq S_t \leq S_{max}$

where S_t is the storage level, Q_t is the inflow, L_t is the loss (e.g., seepage, evaporation), and R_t is the release (all in volumetric units) and time period t.

It's important to note the distinction between the release R and the "target" R_{target} (or the "demand"). The release is what is actually released, whilst the target is the desired release. For example, if the reservoir supplies water to a city that has a demand of 1000 megalitres per month, then the monthly target is 1000 megalitres. However, it may be that after a prolonged dry period the reservoir can only supply 900 megalitres (if, for example, $Q + S < R_{target}$). So the actual release would be 900 megalitres, and there would be a 100 megalitre curtailment.

For the most basic reservoir study, we can assume a constant release target and neglect losses from the reservoir. All we need then are the inflow time series, the target release, and the storage capacity:

```
library(reservoir)       # load reservoir
inflow <- resX$Q_Mm3     # reservoir inflow time series
target <- 120            # target release = 120 million cubic metres / month
capacity <- 120 * 12     # reservoir capacity = 1440 million cubic metres (1 year of demand)
```

The simulation is executed in `reservoir` using the `simRes` function:

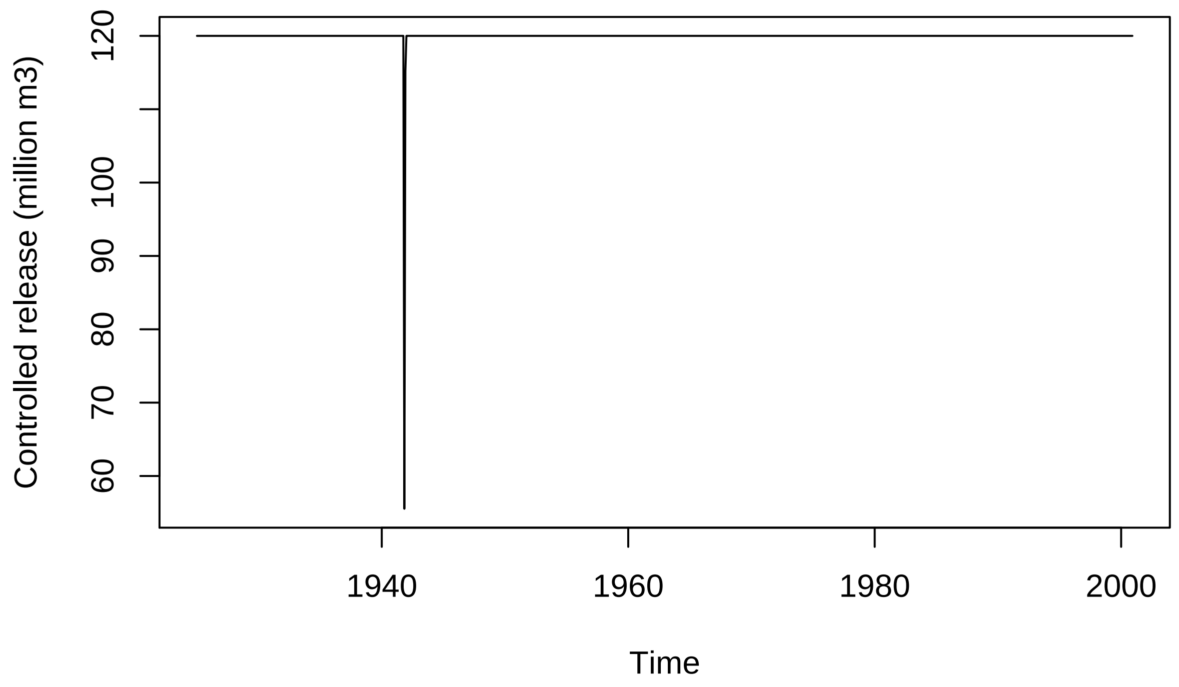
```
# simulate reservoir with output to x, and plotting suppressed
x <- simRes(inflow, target, capacity, plot = F)

# summarise simulation result
summary(x)
```

```
##             Length Class Mode
## storage     912    ts    numeric
## releases    912    ts    numeric
## evaporation 912    ts    numeric
## water_level 912    ts    numeric
## spill       912    ts    numeric
```

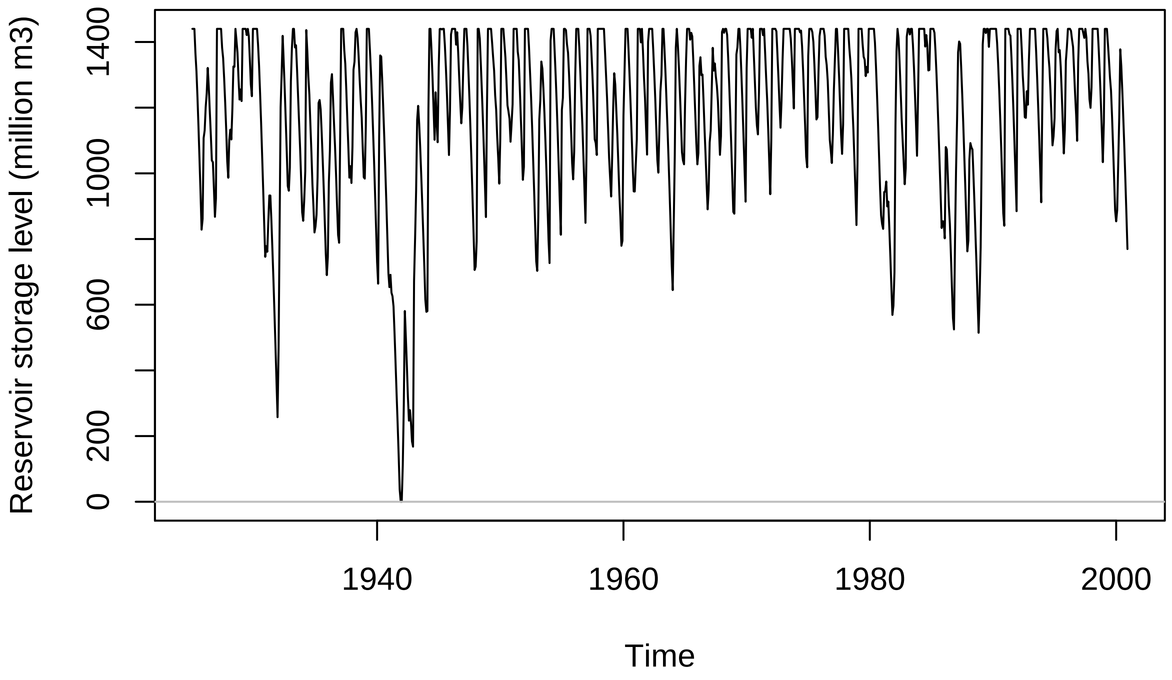
The simulation produces five output time series: `storage` (the reservoir storage behavior), `releases` (the releases from the reservoir), spill (the water spilled from the reservoir when full), `evaporation`, and `water level`. All of these are time series objects with same frequency as `inflow`. Let's intepret the results:

```r
plot(x$releases, ylab = "Controlled release (million m3)")
```



Recall that the `target` is 150 million cubic metres. This means that the reservoir should provide a constant release of 150 Mm3 at all times. But it fails to meet this target in the early 1940s. We can see why this happens if we look at storage:

```r
plot(x$storage, ylab = "Reservoir storage level (million m3)")
abline(h = 0, col = "gray")
```

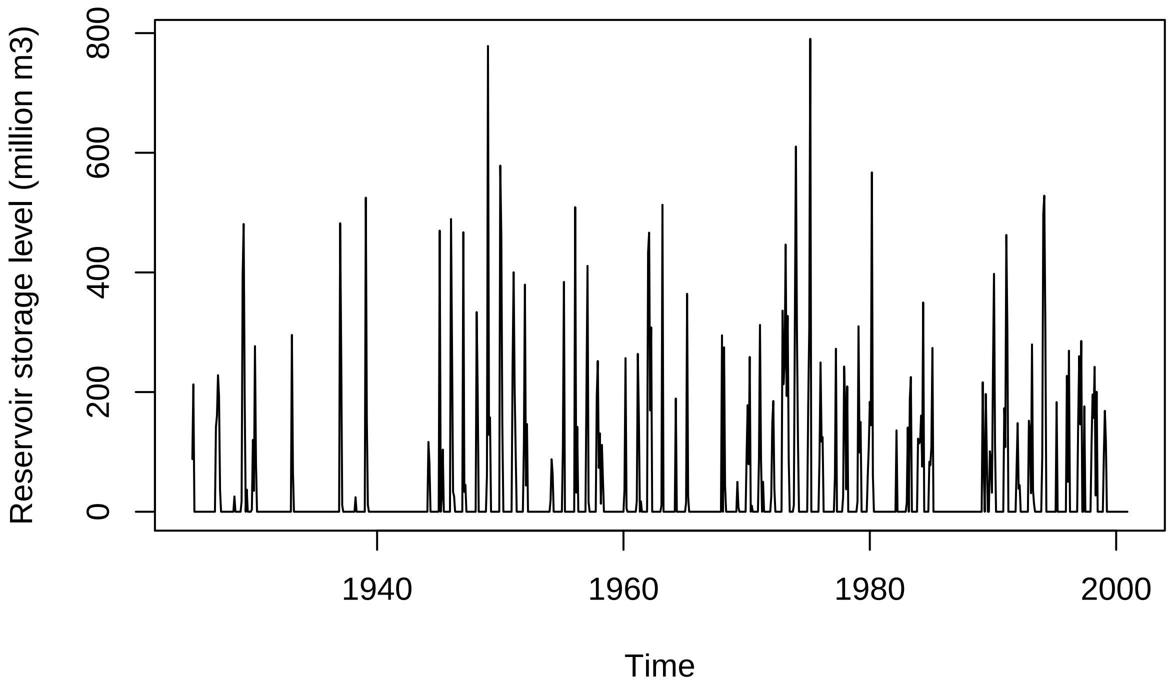


The storage level fluctuates over the simulation period. When the storage becomes empty, there's a risk of curtailment. If the inflow is low and there is no water in storage, then there's simply not enough water to meet the target. In this particular simulation, the operating mode is *Standard Operating Policy (SOP)*—always release to meet the target given the water available in storage and inflow. `resSim` can operate with optimized and user-defined release rules (covered in chapter 5), but unless otherwise specified, the simulation will assume SOP.

> **Standard Operating Policy** is commonly used in reservoir analysis because (1) it's a very simple operating rule, and (2) it's conservative for reservoir design applications (next chapter). `reservoir` assumes SOP as the default for all simulation and analysis functions.

The reservoir spills over available space in storage is insufficient to hold incoming flow:

```
plot(x$spill, ylab = "Reservoir storage level (million m3)")
```



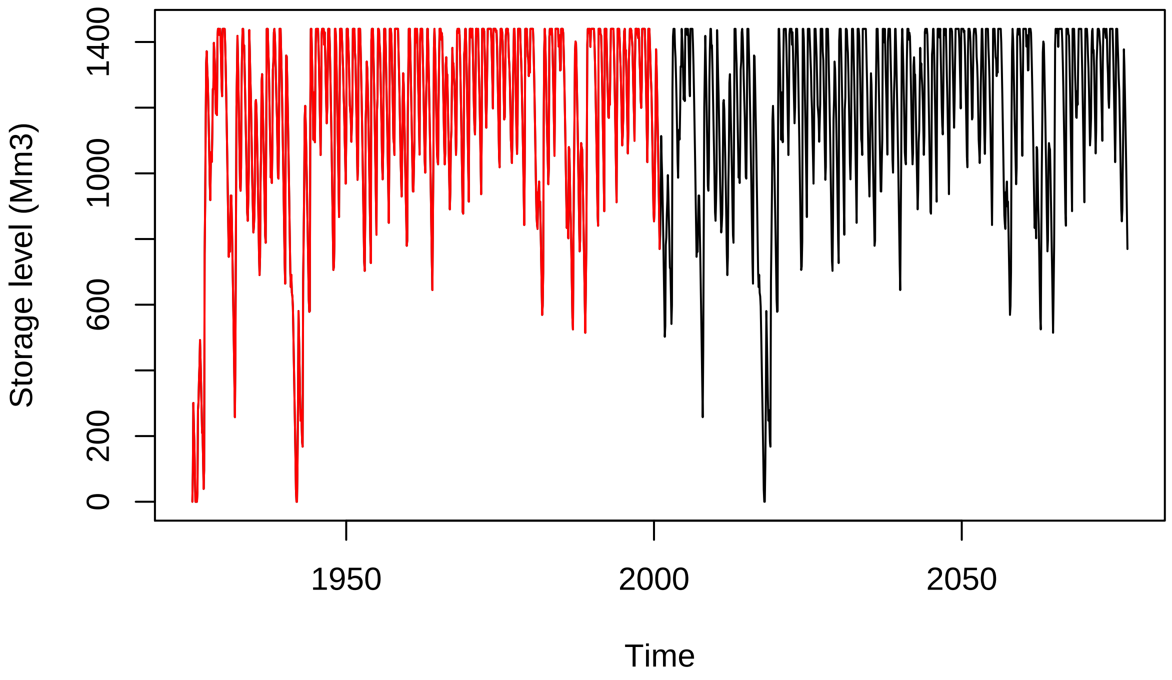
Notice that the reservoir is assumed full at the start of the simulation. This assumption can be problematic when analyzing the performance of the reservoir, because the performance statistics can be sensitive to this arbitrary initial storage level (covered in more detail in Chapter 3). In `reservoir` there are a couple of ways to deal with this problem. First, you can *double cycle* the simulation, which means that the inflow sequence is simply replicated, placed end-to-end, and simulated through the storage. As the simulation approaches the second round of inflows, a new starting storage equal to the end of simulation storage will be read (note that there are other advantages to using a *double cycle*, discussed in Chapter 3). The *double cycle* is initiated by setting the `double_cycle` parameter to `TRUE`. You may also simply change the initial storage assumption by setting the `S_inital` parameter.

```
# run simulation with initial storage of 0
x1 <- simRes(inflow, target, capacity, S_initial = 0, plot = F)

# run same simulation with a double cycle
x2 <- simRes(inflow, target, capacity, S_initial = 0, double_cycle = T, plot = F)



# compare results
plot(x2$storage, ylab = "Storage level (Mm3)")
lines(x1$storage, ylab = "Storage level (Mm3)", col = "red")
```



Depth-storage-area relationships and evaporation losses

The two other outputs of `simRes` are the evaporation and the water level. In the above example the evaporation output is simply zero, whilst the water level is `Inf` (infinity).

```
head(x$evaporation)  # look at the evaporation time series
```

```
## [1] 0 0 0 0 0 0
```

```
head(x$water_level)  # look at water_level output
```

```
## [1] Inf Inf Inf Inf Inf Inf
```

In `reservoir`, potential evaporation is specified as a lenth unit (e.g., metres), which is multiplied by the reservoir surface area to compute actual evaporative water losses from the reservoir (cubic metres). Since a reservoir's surface water area will change depending on storage level (reservoirs are generally non-cuboid), we also need to provide some information on reservoir shape. Generally, when the reservoir is full, the water surface area is at its maximum and so the evaporative potential is at its highest. Reservoir shape can be assigned by specifying surface area only, or by specifying both surface area and maximum depth (the latter will result in a more realistic shape assumption). Currently, `reservoir` does not allow for you to specifiy your own bathymetry, but instead relies on pre-programmed storage-depth-area functions for a typical reservoir shapes (details available by typing `?reservoir`.) In the following example we'll assume that both the maximum surface area `A` and the maximum reservoir depth `y` are available for setting the shape:

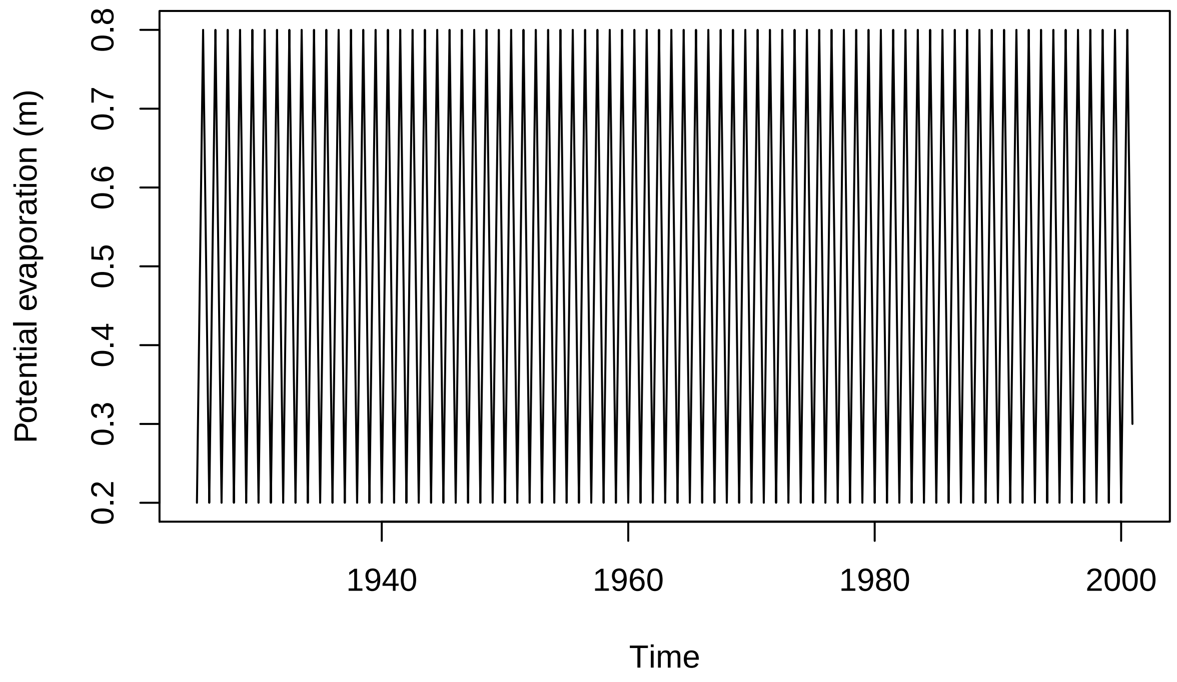
```
maxArea <- 70    # square km (water surface area when full)
maxDepth <- 30   # m (depth when full)
```

There are two ways in which the evaporation can be implemented: (1) as a constant (suitable for equatorial regions where temperature tends not to vary by season); or (2) as a time-series (best option when there is seasonal temperature, or if actual potential evaporation data are available).

Let's assume that we don't have actual potential evaporation data, but that we know the seasonal average change:

```
# set up JAN - DEC evaporation signal (metres)
PE <- c(0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3)

# write out as time series matching inflow
# (note that the ts function will automatically repeat out the seasonal...
# ... signal if we specify the start, end, and frequency)
PE_ts <- ts(PE, start = start(inflow), end = end(inflow), frequency = 12)
plot(PE_ts, ylab = "Potential evaporation (m)")
```

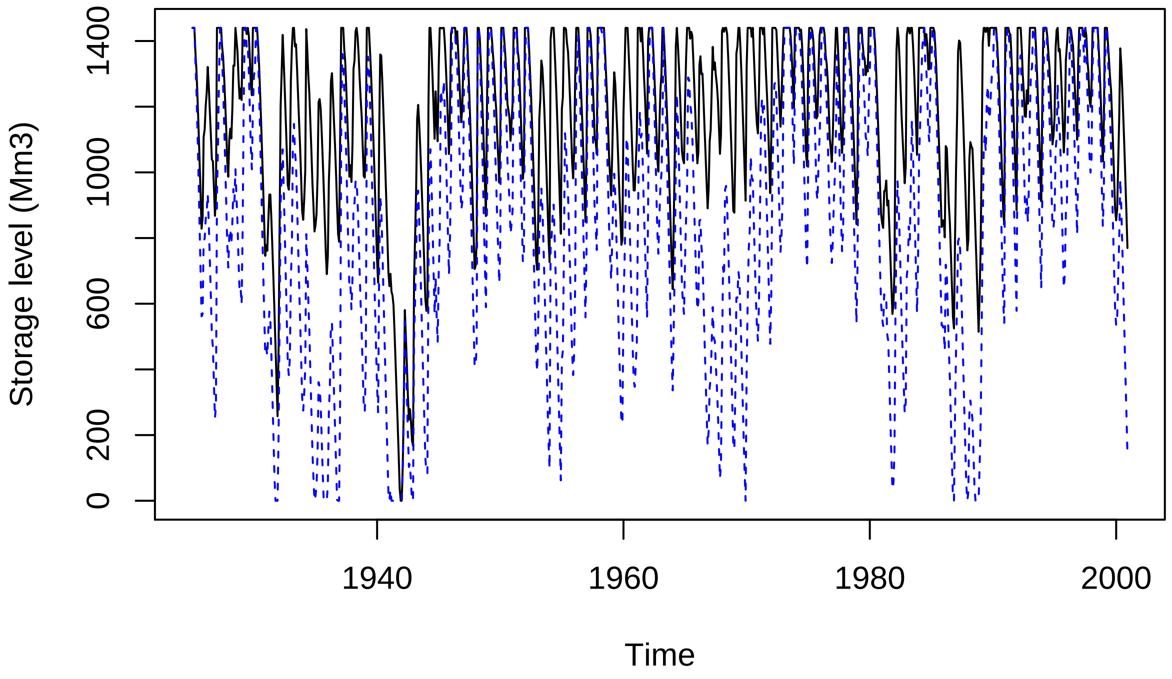


Then to execute the simulation we simply supply the simRes function with the new data:

```
x_incEvap <- simRes(inflow,                    # reservoir inflow
                    target,                    # target release
                    capacity,                  # reservoir capacity
                    surface_area = maxArea,    # water surface area at capacity
                    max_depth = maxDepth,      # maximum depth at capacity
                    evap = PE_ts,              # seasonal evaporation signal
                    plot = F)

# plot the storage time series for the previous simulation (no evap.)
plot(x$storage, ylab = "Storage level (Mm3)")
```

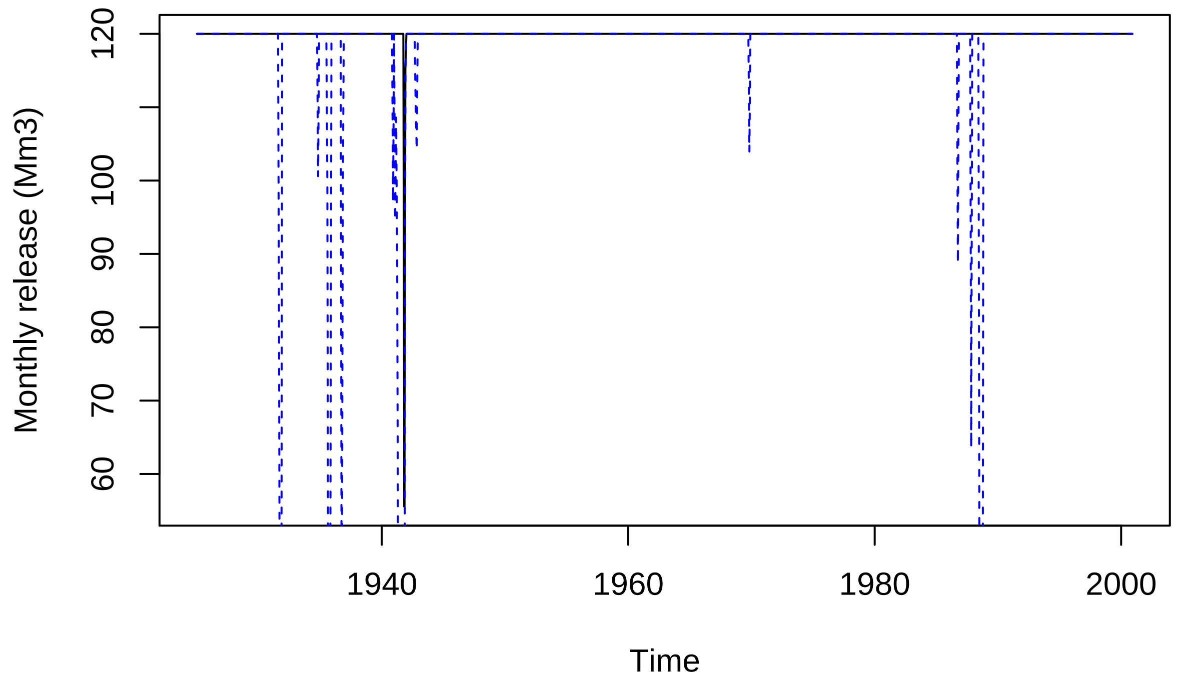
```r
lines(x_incEvap$storage, lty = 2, col = "blue")
```



> **Important**: when using evaporation it is imperative that you use the recommended units given in the `reservoir` documentation (i.e., the units specified in the function help when you enter, e.g., `?simRes`). `reservoir` is set up to use metric units: specifically, volumes (storage, release, inflow) in million m3, surface area in km2, and evaporation in m (or kg/m2 * 10 ˆ 3).

How important is the evaporation? In this case it seems that including evaporation causes the reservoir to fail more frequently. Look at the release time series to see how often the target is missed:

```r
plot(x$releases, ylab = "Monthly release (Mm3)")
lines(x_incEvap$releases, lty = 2, col = "blue")
```



The frequency, magnitiude, and duration of curtailments in the release are often quantifed to assess reservoir performance. In the next chapter, we'll look at how to use `reservoir` to compute a range of performance metrics.