MAST90045

Systems Modelling and Simulation

Assignment 1

Chris Swan 370502

University of Melbourne

Contents

1 Problem Introduction

This problem offers insight not only into applications of modelling and simulation, but also into methods for finding solutions to real world problems. Being able to accurately anticipate how rainfall will affect the level and volume of a dam is pertinent across the world.

But this is only a snapshot of the extensive applications that iterative modelling opens up. The ability to find and offer solutions to problems that cannot be solved analytically is a valuable resource in business, industry and the wider community. We therefore consider the exploration below as a reconnoitring of how iterative modelling can be used in these fields.

In this scenario we examine how change in volume affects the level of water in a dam. Whilst we begin by examining generic cases, we go on to examine more specific cases, with the goal of eventually being able to generalise for any scenario where certain information about a dam is known. In our specific case, our program developed below allows one to be able to extrapolate a change in the water level of a dam over time if the rainfall in the catchment area is known.

2 Volume

For any generic dam, we are given the function $A(h)$ to describe the cross-sectional area of the dam at height h. We can then easily deduce that the volume of water contained within the dam will be the integral of $A(h)$ with the limits (it being a 3-dimensional physical space) being from 0 to h.

We therefore have the volume function:

$$V(h) = \int_0^h A(u)\ du$$

As we want to be able to calculate $V(h)$ for any generic area function $A(h)$, we need some numerical algorithm for calculating the integral of $A(h)$ should it not be feasible (or possible) to solve $\int A(h)$ analytically.

Therefore, we elect to use Simpson's rule as our numerical algorithm for approximating $V(h)$. There are disadvantages in using approximation algorithms, as a precise answer cannot necessarily be computed, and Simpson's rule is no exception. However, in giving due consideration to the error that might result, Simspon's rule will offer accuracy to a degree that any resulting error will be negligible.

Regarding the accuracy, through consideration of both the potential impact of approximation error as well as computation feasibility, we eventually elected to allow for an error tolerance of $1 \cdot 10^{-10}$ as balance between these two aspects. Given that the units of area and volume are unknown, were the volume measured in Megalitres for example, a volume accuracy of $1\text{e} - 10$ would give an accurate volume to 0.1 millilitres.

Whilst this might seem excessive, accuracy to a ten-thousand-millionth of a megalitre allows for some flexibility should other models be required. It was also computationally feasible for our model, and it should be noted that increasing the error tolerance increased the computation time of our root-finding algorithm significantly. Given our discussion above, any further increase in accuracy was deemed unnecessary.

A further outline of the volume function and Simpson's rule can be found in Appendix B.

3 Height

Given a current level of the water in the dam h, we want to be able to calculate what the new dam level (or "height") will be for some change v in the volume in the dam. The new "height" will therefore be

$$u = H(h, v)$$

where u satisfies

$$V(u) = V(h) + v$$

where $0 \leq V(u) \leq V(h_{\text{max}})$.

If $V(u) < 0$, then $u = 0$ and if $V(u) > V(h_{\text{max}})$, then $u = h_{\text{max}}$.

Consequently, we need to define a new function that links these different constraints together and allows us to calculate values of u. To do this, we need to transform our function so that when u is inputted, it returns a value of 0. This will allow us to make use of a root finding algorithm to compute u, which will be the value of the new height of the water in the dam after the change in volume.

For a root finding algorithm, we employ the Newton-Raphson method of using the derivative of a function to iteratively step towards its root. The concern with the Newton-Raphson method is that a function must be continuous and have a derivative $\in \mathbb{R}$ as the method involves dividing by the derivative. However, given that this derivative will be our area function, we can be assured that the derivate both exists and is well defined for $V(u)$

In setting the tolerance and determining the number of iterations to run for the Newton-Raphson function, we explored a similar process to that used for Simpson's rule. In attempting to marry accuracy and efficiency, we found that unless the tolerance for the Simpson's function was excessively small, then convergence occurred relatively quickly with the Newton-Raphson tolerance set at a similar level (10^{-10}).

Setting the tolerance for the Newton-Raphson function smaller led to longer convergence, such that more than 100 iterations were needed, and making it even smaller ($\leq 10^{-14}$) saw the algorithm failing to converge in 10^4 iterations. In consequence, we came to a similar deduction as we did with Simpson's Rule, that a tolerance of 10^{-10} adequately covered variance in units bases for "height" without sacrificing computation time.

4 Test Case

The test case offered a good opportunity to troubleshoot and debug the program written, as well as refine our code to improve readability of output as well as the code itself. Augmenting additional functions proved useful for reuse later, as well as simplifying redundant or inefficient segments of code.

The testing of the functions, the output of which is shown in Figure 1, verified that our functions were acting correctly for values outside of the specified range, in this case returning NA for h values outside of $0 \leq h \leq h_{\text{max}}$ and limiting the values of the new height, "dam_level", at these values when they are returned outside of the specified range. Figure 2 gives some data visualisation of the values returned from the test inputs.

Test Cases	Height Inputs	Volume Inputs	Height Outputs
1	-1.0	1	NA
2	0.5	1	1.5000000000
3	0.5	-1	0.0000000000
4	3.5	1	3.7749172176
5	3.5	2	4.0000000000
6	3.5	-1	3.2015621187
7	5.0	1	NA

Figure 1: Table of test case and results

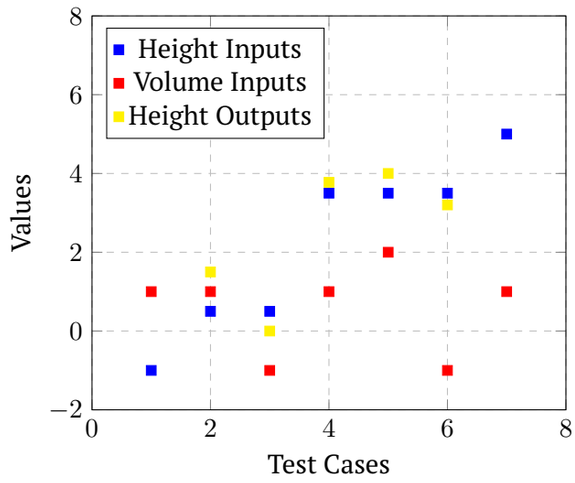

Figure 2: Results of test cases

5 Tracking height over time

We come to the implementation of what we have developed above. We have developed a method for calculating what the dam's height will be, based on the volume of rain falling in the catchment area, the volume of water used from the dam and the volume of water evaporated from the dam. The new level of water in the dam is thus given by

$$h(t+1) = H(h(t), v(t) - \alpha - \beta A(h(t)))$$

The area of the dam is defined by

$$A(h) = \begin{cases} 100h^2 & 0 \leq h \leq 2 \\ 400(h-1) & 2 \leq h \leq 3 \end{cases}$$

From this information, we can iteratively calculate the level of the dam after the volume of catchment water has been added.

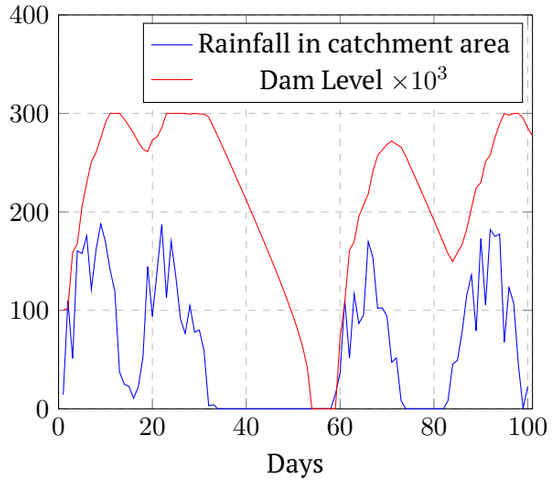

Figure 3: Volume of rain falling in catchment area and resulting dam level$\times 10^3$

6 Summary

Creating a plot of the output of this (see Figure 3), we can clearly see the relationship that the rainfall has upon the level of water in the dam, as we expected. It is evident that there is some delay between a lack of rainfall and a drop in water level, and this itself provides opportunity for action were this model used to advise a body such a local or state government in a time of drought.

It is noteworthy that parts of this project can seem quite abstract when written in mathematical notation but have more clarity when explored iteratively. So too some aspects that are challenging to code are aided by exploring mathematical notation.

7 Apendices

7.1 Appendix A: Code

```
rm(list=ls())

#spuRs alogorithms

# Simpson's Rule, taken from spuRs:
# program spuRs/resources/scripts/simpson.r

simpson <- function(ftn, a, b, tol = 1e-8, verbose = FALSE) {
  # numerical integral of ftn from a to b
  # using Simpson's rule with tolerance tol
  #
  # ftn is a function of a single variable and a < b
  # if verbose is TRUE then n is printed to the screen

  # initialise
  n <- 4
  h <- (b - a)/4
  fx <- sapply(seq(a, b, by = h), ftn)
  S <- sum(fx*c(1, 4, 2, 4, 1))*h/3
  S.diff <- tol + 1  # ensures we loop at least once

  # increase n until S changes by less than tol
  while (S.diff > tol) {
    # cat('n =', n, 'S =', S, '\n')  # diagnostic
    S.old <- S
    n <- 2*n
    h <- h/2
    fx[seq(1, n+1, by = 2)] <- fx  # reuse old ftn values
    fx[seq(2, n, by = 2)] <- sapply(seq(a+h, b-h, by = 2*h), ftn)
    S <- h/3*(fx[1] + fx[n+1] + 4*sum(fx[seq(2, n, by = 2)]) +
                2*sum(fx[seq(3, n-1, by = 2)]))
    S.diff <- abs(S - S.old)
  }
  if (verbose) cat('partition size', n, '\n')
  return(S)
}


# program spuRs/resources/scripts/newtonraphson.r
# loadable spuRs function

newtonraphson <- function(ftn, x0, tol = 1e-9, max.iter = 100) {
  # Newton_Raphson algorithm for solving ftn(x)[1] == 0
  # we assume that ftn is a function of a single variable that returns
  # the function value and the first derivative as a vector of length 2
  #
  # x0 is the initial guess at the root
  # the algorithm terminates when the function value is within distance
  # tol of 0, or the number of iterations exceeds max.iter

  # initialise
  x <- x0
  fx <- ftn(x)
  iter <-  0

  # continue iterating until stopping conditions are met
  while ((abs(fx[1]) > tol) && (iter < max.iter)) {
    x <- x - fx[1]/fx[2]
    fx <- ftn(x)
    iter <- iter + 1
```

```
    cat("At iteration", iter, "value of x is:", x, "\n")
  }

  # output depends on success of algorithm
  if (abs(fx[1]) > tol) {
    cat("Algorithm failed to converge\n")
    return(NULL)
  } else {
    cat("Algorithm converged\n")
    return(x)
  }
}


##########################################################################################################

#volume


volume = function(h,hmax,ftn) {
  #we have function 'ftn' as the cross-sectional area of at height 'h',
  #with 'hmax' as the max height
  if (h < 0) {
    return(NA)
  } else if (h > hmax) {
    return(NA)
  } else {
    V = simpson(ftn,0,h, tol = 1e-10)
    return(V)
  }
}

#height

height = function(h,hmax,v,ftn) {
  #our function for computing the new dam level from a given current height 'h',
  #the maximum dam level 'hmax',
  #the current volume 'v' and the area function 'ftn'
  if (h < 0) {
    return(NA)
  } else if (h > hmax) {
    return(NA)
  } else {
    new_volume = as.numeric(volume(h, hmax, ftn) + v)
    max_volume = volume(hmax,hmax,ftn)
    if (new_volume > max_volume) {
      dam_level = hmax
      print(cat('The dam has reached the maximum level of', dam_level, '\n'))
    } else if (new_volume < 0) {
      dam_level = 0
      print(cat('The dam has reached the minimum level of', dam_level, '\n'))
    } else {
      newt_input = function(x) {
        N = volume(x,hmax,ftn) - new_volume
        return(c(N,ftn(x)))
      }
    dam_level = newtonraphson(newt_input,hmax, tol = 1e-10, max.iter = 1000)
    }
  return(dam_level)
  }
}
```

```
#translating our variables to the assignment:

# dam_level = u = H(h,v)
# volume (function) = V(h)
# new_volume = V(u)
# ftn (function) = A(h)

#########################################################################################################

#test case




heights = c(-1,0.5,0.5,3.5,3.5,3.5,5)
volumes = c(1,1,-1,1,2,-1,1)

# Let's evaluate our test criteria:

A = function(h) return(h)
V_h = function(k) return(k^2/2)
H_h_v = function(i,j) return(sqrt(i^2 + 2*j))

for (i in seq(1,7)) {
      print(cat("The Area of A", i," is ", A(heights[i]), "\n",
                'The Volume of V', i," is ", V_h(heights[i]), "\n",
                'The Dam Level of H',i, " is ", H_h_v(heights[i],volumes[i]), "\n",
                'The New Height of h', i, " is ", height(heights[i],
 hmax = 4, volumes[i], ftn = A), "\n"))
}

#########################################################################################################


#tracking height over time

catchment = scan(file = "/School/Mast of Sci/MAST90045/A1/catchment_b-1.txt")

dam_calc = function(volume_data, initial_height = 1, hmax = 3, alpha = 2, beta = 0.1) {
  #our function for calculating the new levels of a dam from a vector of catchment data of
  #volume of rainfall, with 'volume_data' being a vector of catchment volume,
  #'initial_height' being the starting dam level, 'hmax' being the max dam level,
  #'alpha' being the volume of water taken from the dam for use per day and
  #'beta' being the volume of water lost due to evaporation during a day
  h_vector = c(initial_height)
  len = length(catchment)
  Area = function(h) {
    if (0 <= h & h <= 2) {
      return(100*h^2)
    } else if (2 <= h & h <= 3){
      return(400*(h - 1))
    } else (
      return(NA)
    )
  }
  for (i in seq(1,len)) {
    if (h_vector[i] < 0) {
      h_vector[i] = 0
    } else if (h_vector[i] > hmax) {
      h_vector[i] = hmax
    }
    vol = volume_data[i] - alpha - beta*Area(h_vector[i])
    next_h = height(h_vector[i], hmax, vol, Area)
```

```
    h_vector = append(h_vector, next_h)

  }
  return(h_vector)
}

rain_volumes = append(catchment, NA)
dam_heights = dam_calc(catchment)
dam_heights = formatC(dam_heights, digits = 10, format = 'f')
Results = cbind(rain_volumes,dam_heights)
write.csv(Results, file = "/School/Mast of Sci/MAST90045/A1/results.csv")

days = 1:101
plot(days,dam_heights, type="l")
```

7.2 Appendix B: Functions & Algorithms

7.2.1 Volume function and Simpson's rule

$$S = \frac{y}{3}\left(f(x_0) + 4\sum_{i=1}^{n-1} f(x_{2i}) + 2\sum_{i=2}^{n-2} f(x_{2i}) + f(x_n)\right)$$

where y is the difference between the limits of our integral, with a being the lower limit and b being the upper limit. This is then divided by the number of partitions n.

$$\therefore y = \frac{b-a}{n}$$

NB y is commonly notated as h in Simpson's rule, but we call it y to avoid confusion with our "height" variable "h".

The sequence $[1, 4, 2, 4, \cdot, 4, 1]$ that can be observed is a method for saving on computational cost, enabling the algorithms to reuse previously computed values of f.

The number of partitions n will increase until S changes by less than the specified tolerance.

Our volume function calls Simpson's rule to evaluate the integral of our area function.

$$V(h) = \frac{\frac{h-0}{n}}{3}\left(A(x_0) + 4\sum_{i=1}^{n-1} A(x_{2i}) + 2\sum_{i=2}^{n-2} A(x_{2i}) + A(x_n)\right)$$

7.2.2 Height function and the Newton-Raphson method

The Newton-Raphson method for iteratively finding a root of a function $f(x)$ is defined as

$$x_{n=1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

where $f'(x_n)$ is defined and not equal to 0.

From definition of volumes of the new dam level, we can see the following:

$$\begin{aligned} V(u) &= V(h) + v \\ 0 &= V(u) - V(h) - v \\ &\frac{d}{du}\Big(V(u) - V(h) - v\Big) \\ &= V'(u) \\ &= f(u) \end{aligned}$$

Therefore, we have all components necessary for computing roots of V(u).