Introduction to computational programming

Chapter 2 Lab Exercise

Cobweb plots and stability

David M. Rosenberg
University of Chicago
Committee on Neurobiology

Version control information:
Last changed date: 2009-11-05 03:39:38 -0600 (Thu, 05 Nov 2009)
Last changes revision:198
Version: Revision 198
Last changed by: David M. Rosenberg

November 5, 2009

Part I

Exercise

1. Find the equilibrium values of populations governed by the following equations, and determine their stability analytically. Is there is a stable nonzero equilibrium (carrying capacity) that the population may approach in the long run?
   (a) $N_{t+1} = 41N_t - 10N_t^2$
   (b) $N_{t+1} = 41N_t + 2N_t^2$

(a)

$$\begin{aligned} N_{t+1} &= 41N_t - 10N_t^2 \\ \text{Let } x &= N_{t+1} = N_t \\ x &= 41x - 10x^2 \\ 0 &= 10x(4 - x) \\ x &\in \{0, 4\} \end{aligned}$$

(b)

$$\begin{aligned} N_{t+1} &= 41N_t - 2N_t^2 \\ \text{Let } x &= N_{t+1} = N_t \\ x &= 41x - 2x^2 \\ 0 &= 2x(20 - x) \\ x &\in \{0, 20\} \end{aligned}$$

3. Generate cobweb plots for the following logistic models. Graphically identify all fixed points, and state whether they are "stable."
   (a) $f(x) = 3x - \frac{3x^2}{4} + 1$
   (b) $f(x) = 100x - 2x^2$
   (c) $f(x) = -100x + \frac{x^2}{2}$
4. Write a *functor* which takes as an argument a function describing a logistic model and generates a cobweb plot.

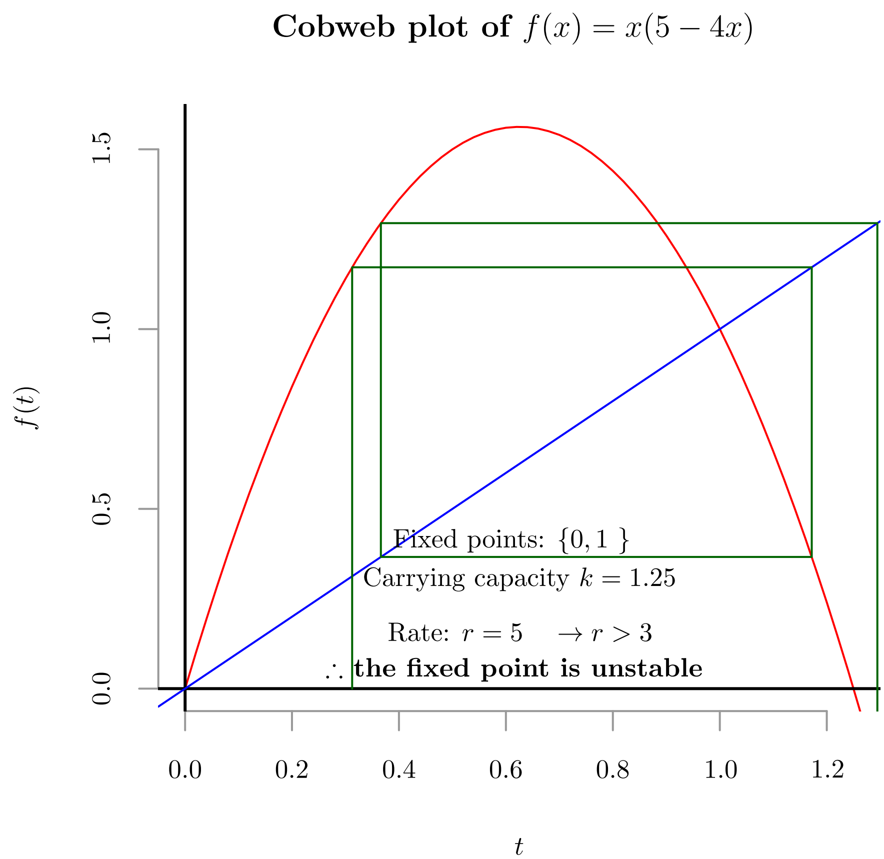
```
f1 <- function (x)
  x * (5 - 4 * x)
f2 <- function (x)
  2 * x * (1 - 3 * x / 2)
f3 <- function (x)
  x / 2 - x^2 / 5
f4 <- function (x)
  x * (5/2 - 7 * x)
```

```
plotCobWeb <- function (f, max_iter=50) {
  ## initialize variables
  f_exp <- deparse(body(f))
  df <- mDeriv(f);
  df_exp <- deparse(body(df))
  fixed_pts <- sort(mSolve.RServe(paste(f_exp, 'x', sep='=')));
  zeros <- sort(mSolve.RServe(paste(f_exp, '0', sep='=')));

  rate <- df(0);
  carrying_cap <- max(zeros);

  init_value <- min(zeros) + diff(zeros) / 4;
  y <- x <- numeric(length=max_iter*2);
  x[1] <- init_value;
  y[1] <- 0;

  ## Loop over iterations

  for (ii in seq(along=1:max_iter)) {
    y[2 * ii + c(0,1)] <- f(x[2 * ii - 1]);
    x[2 * ii + c(0,1)] <- c(x[2 * ii - 1], y[2 * ii + 1]);
  }

  ## Determine plot limits
  if (all(is.finite(x) & is.finite(y))) {
    xlim <- c(min(floor(zeros)), max(ceiling(zeros)));
    ylim <- c(  max(c(0, min(f(c(zeros, fixed_pts, mean(zeros)))))),
                max(f(c(zeros, fixed_pts, mean(zeros)))))
  } else {
    yranges <- xranges <- c(1);
    for (ii in 2:length(x)) {
      xranges <- c(xranges, diff(range(x[1:ii])));
      yranges <- c(yranges, diff(range(y[1:ii])));
    }
    xlim <- c(min(floor(zeros)), max(ceiling(zeros)));
    ymagnitudes <- na.omit(yranges[-1] /  yranges[-length(yranges)])
    first_runaway <- min( (1:length(ymagnitudes) )[ymagnitudes > 100])

    ylim <- c(0, max(c(y[1:(first_runaway - 1)], f(mean(zeros)), f(zeros),
              f(fixed_pts))));
  }

  ## Draw plot elements
  curve(f, from=xlim[1], to=xlim[2], ylim=ylim, xlim=c(min(zeros),
        max(zeros)), fg=gray(0.6), bty='n',col='red', xlab="$t$",
        ylab="$f(t)$");
  abline(h=0, lwd=1.5);
  abline(v=0, lwd=1.5);
  abline(0, 1, col='blue', lwd=1);
  lines(x, y, col='darkgreen', lwd=1);

  ########################################################
  ##  Page break only: Function continues on next page  ##
  ########################################################
}
```

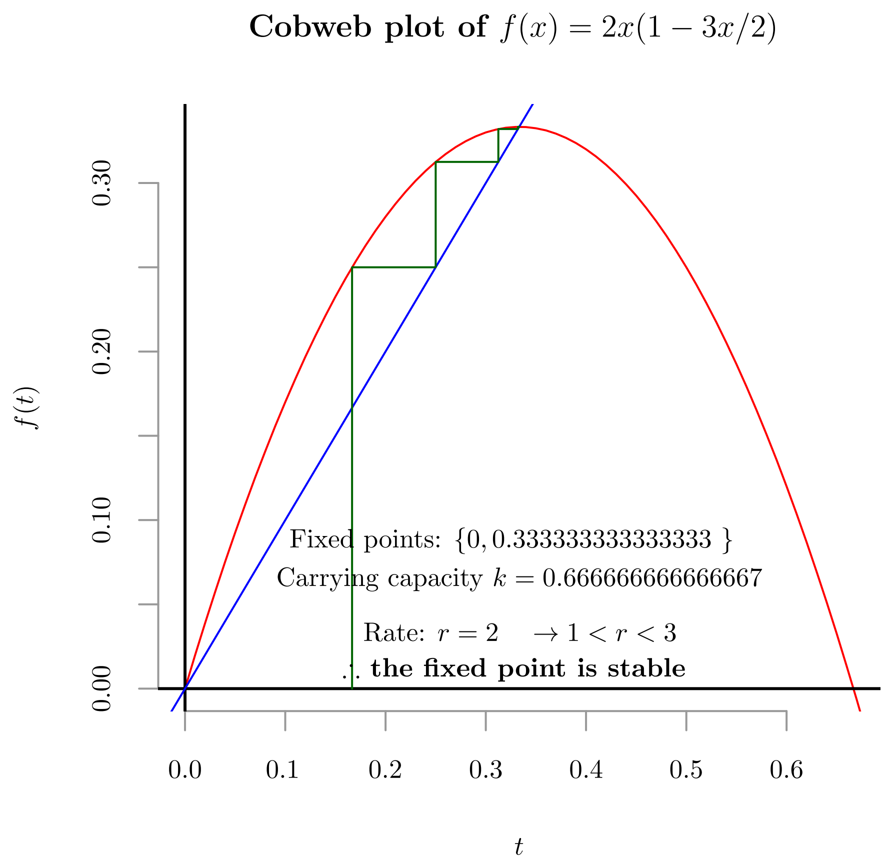
```
{
  ############################################################
  ##  Page break only: Function continues from last page  ##
  ############################################################

  ## Calculate label positions
  typ <- text_y_positions <- mean(ylim) - diff(ylim) / 15 * c(4, 5, 6, 7, 8);
  txp <- mean(zeros)

  ## Add plot description
  title(main=paste("{\\larger\\bf Cobweb plot of $ f(x) = ",
        gsub("\\*", "", f_exp),"$ }"));
  text(   x=mean(zeros), y=typ[1],
          paste(  "Fixed points: $ \\{ ", paste(as.character(fixed_pts),
                collapse=", "), " $ \\} \nCarrying capacity $ k=",
                carrying_cap, " $"));
  if(rate < 3 && rate > 1) {
    text(x=txp, y=typ[3], paste('Rate: $ r= ', rate,
                                "\\quad \\rightarrow 1 < r < 3$"));
    text(x=txp, y=typ[4], "{\\bf $\\therefore $ the fixed point is stable }");

  } else {
    text(x=txp, y=typ[3],
         paste('Rate: $r = ', rate, "\\quad \\rightarrow r > 3$") );
    text(x=txp, y=typ[4],
         "{\\bf $\\therefore $ the fixed point is unstable }");
  }
}
```

□

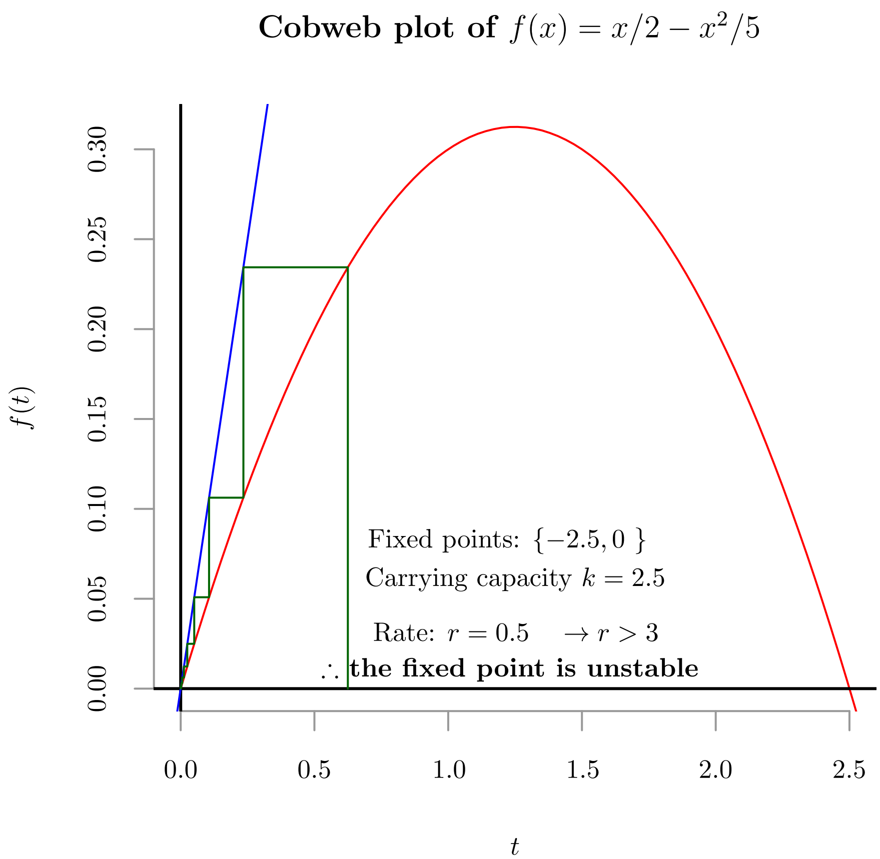
```
f <- f1;
plotCobWeb(f)
```




```
f <- f2
plotCobWeb(f2)
```

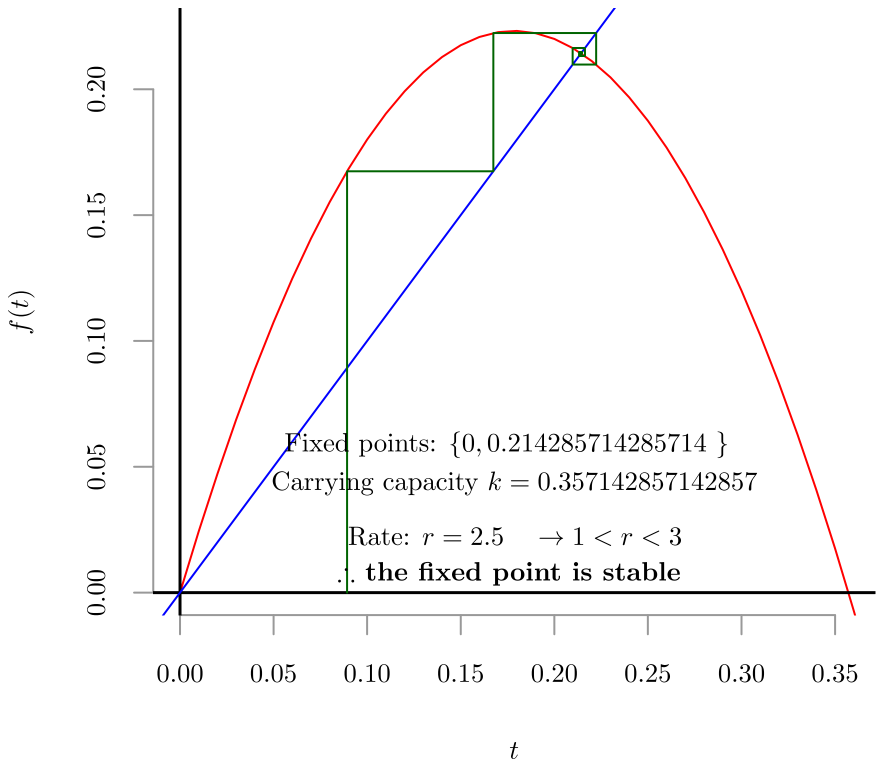



```
f <- f3
plotCobWeb(f)
```




```
f <- f4
plotCobWeb(f)
```

Cobweb plot of $f(x) = x(5/2 - 7x)$



5. In physiology, maintaining a steady level of glucose in the bloodstream is necessary for the proper functioning of all organs. To study this process, define $G(t)$ to be the amount of glucose in the bloodstream of a person at time t. Assume that glucose is absorbed from the bloodstream at a rate proportional to the concentration $G(t)$, with rate parameter k.

   (a) Write down a differential equation to describe this situation. What kind of ODE is it?

   $\dot{G} = kG(t)$, a homogenous linear differential equation.

   (b) Find the analytical solution for this equation, in terms of an initial value G_0 and the rate parameter k.

$$
\begin{aligned}
\frac{dG}{dt} &= k * G(t) \\
\frac{dG}{dt} \cdot \frac{1}{G(t)} &= k \\
\int \left(\frac{dG}{dt} \cdot \frac{1}{G(t)} \right) dt &= \int k \; dt \\
\int \frac{1}{G(t)} dG &= \int k \; dt \\
\log G(t) &= kt + C \\
G(t) &= e^{kt+C}
\end{aligned}
$$

(c) Let the initial glucose concentration be $G_0 = 100mg/dl$, and the glucose removal rate be $k = 0.01/min$. Use R to plot the solution as a graph over a reasonable time interval, with properly labeled axes.

$$
\begin{aligned}
G(t) &= e^{kt+C} \\
G(0) &= e^{C} && = 100\text{mg/dl} \\
G(t) &= e^{0.01 \text{ min}^{-1} \cdot t \text{ min} + 100 \text{ mg/dl}} \\
G(t) &= e^{100 - t/100}
\end{aligned}
$$

(d) What is the equilibrium concentration of blood sugar in this model? Is the equilibrium stable or unstable?

Under the assumption that k is negative (inferred from the text), the equilibrium concentration is 0 mg/dl.

6. Now let us assume that glucose is added to to bloodstream at a constant rate a, independent of glucose concentration.

(a) Write down a differential equation to describe this situation. What kind of ODE is it?

$$\dot{G} = kG(t) + a$$

(b) Find the analytical solution for this equation, in terms of an initial value G_0 and the parameters k and a.

$$\begin{aligned}
\dot{G} &= kG(t) + a \\
\frac{dG}{dt} &= kG(t) + a \\
G'(t) - kG(t) &= a \\
\\
\text{Let } \mu &= e^{\int -kdt} && \text{(integrating factor)} \\
&= e^{-kt} \\
\frac{d\mu}{dt} &= -ke^{-kt} \\
\\
\mu\,(G'(t) - kG(t)) &= \mu\, a \\
= e^{-kt}G'(t) - ke^{-kt}G(t) &= ae^{\int kdt} \\
= \mu G'(t) + \mu' G(t) \\
= (G(t) \cdot \mu)' && && \text{(product rule)} \\
\\
\int (G(t) \cdot \mu)'\, dt &= \int ae^{-kt} dt \\
G(t) \cdot \mu + C &= \int ae^{-kt} dt \\
G(t) \cdot \mu &= -\frac{a}{k}e^{-kt} + C \\
G(t) &= -\frac{1}{e^{-kt}} \left(\frac{a}{k}e^{-kt} + C\right) \\
&= e^{kt}\left(C - \frac{a}{k}\right) \\
G(0) &= C - \frac{a}{k}
\end{aligned}$$

(c) Let the initial glucose concentration be $G_0 = 100mg/dl$, the glucose removal rate be $k = 0.01/min$, and $a = 4mg/dl/min$. Use R to plot the solution as a graph over a reasonable time interval, with properly labeled axes.

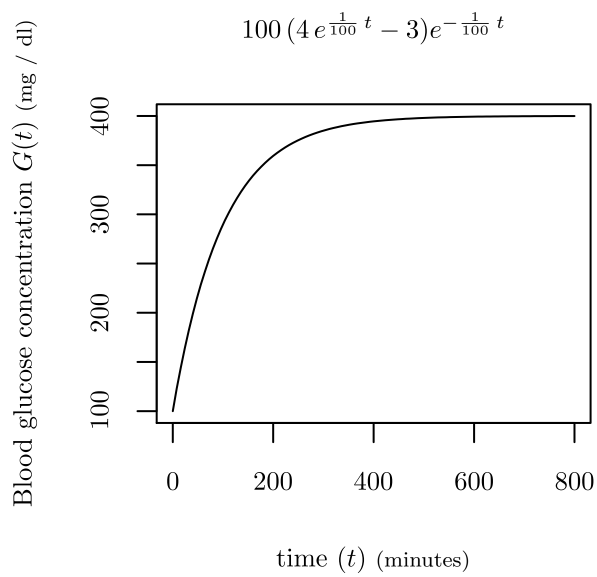
$$G(0) = C - \frac{\mathrm{a}}{k} \tag{1}$$

$$100 = C - \frac{4}{-0.01} \tag{2}$$

$$C = -300 \tag{3}$$

$$100\,(4\,e^{\frac{1}{100}\,t} - 3)e^{-\frac{1}{100}\,t} \tag{4}$$

```
glucose <- function(t) {
  100 *  exp(-1 * t / 100) * (4 * exp(t / 100) - 3)
}
xlab <- "time $(t)$ {\\smaller (minutes)}"
ylab <- "Blood glucose concentration $G(t)$ {\\smaller(mg / dl)}"
main <- "$100 \\, {(4 \\, e^{\\frac{1}{100} \\, t} - 3)} e^{-\\frac{1}{100} \\,
    t}$"
plot(glucose, from=0, to=800, xlab=xlab, ylab=ylab, main=main);
```



(d) What is the equilibrium concentration of blood sugar in the model with glucose infusion? Is the equilibrium stable or unstable?

$$
\begin{aligned}
G(t) &= 400 - 300e^{\frac{-t}{100}} && (5) \\
\frac{dG}{dt} &= 3e^{\frac{-t}{100}} && (6) \\
\lim_{t\to\infty} e^{\frac{-t}{100}} &= 0 && (7) \\
\lim_{t\to\infty} G(t) &= 400 - 300 \cdot 0 && (8) \\
&= 100 && (9) \\
\lim_{t\to\infty} G'(t) &= 0 && (10) \\
& && (11)
\end{aligned}
$$

Thus equilibrium is reached and stable at 400 mg / dl.

7. *Not required. A good exercise which relates to numerical approximation.*

   (a) *Binary Search.* Write a function which, given a sorted vector X of n unique integers, and an integer k, returns the index of k in vector X if $k \in X$ and -1 otherwise. Your solution should perform approximately $\log n$ comparisons in the "worst case" scenario. *Hint:* one example of a "worst case" is when the $X_0 = k$.

```
bSearch1 <- function(intVec, value) {
  nIter <- 0;
  bottom <- 1;
  top <- length(intVec);
  testVal <- intVec[floor(mean(c(top, bottom)))];
  while (testVal != value) {
    nIter <- nIter + 1;    # Just to count steps
  if (testVal < value) {
    bottom <- ceiling(mean(c(top, bottom)));
  } else {
    top <- floor(mean(c(top, bottom)));
  }
  testVal <- intVec[floor(mean(c(top, bottom)))];
  }
  cat(sprintf("Match found in %d steps\n", nIter));
  return(floor(mean(c(top, bottom))));
}
iVec <- sort(sample(1:1000, 10));
for (value in iVec) {
  print(bSearch1(iVec, value));
}

Match found in 3 steps
[1] 1
Match found in 2 steps
[1] 2
Match found in 1 steps
[1] 3
Match found in 2 steps
[1] 4
Match found in 0 steps
[1] 5
Match found in 3 steps
[1] 6
Match found in 2 steps
[1] 7
Match found in 1 steps
[1] 8
Match found in 2 steps
[1] 9
Match found in 4 steps
[1] 10
```

(b) Modify the previous solution so that the requirement of *unique* integers may be relaxed. In the case that multiple solutions exist return a vector of all such solutions.

```
bSearch2 <- function(intVals, value) {
  ## Sanity Check, bail if unsure
  if (!(is.numeric(c(intVals, value)))) {
    stop('Error: parameters must be numeric!\n');
  } else if (length(value) != 1) {
    stop('Error: value must be a vector of length 1!\n');
  } else if (is.unsorted(intVals)) {
    stop('Error: intVals must be sorted!\n');
  } else if (!(all(is.finite(c(intVals, value))))) {
    stop('Error: parameters may not contain infinite or undefined values.\n');
  }

  ## Cheating, a bit - ensure value is in vector
  if (!(value %in% intVals)) {
    return(numeric(length=0));
    #TODO: Remove this hack and code the case loop properly 10/13/2009 DMR
  }

  ## There are two 'breakpoints' - consecutive pairs (a,b) in
  ##   intVals to find such that exactly one of a or b == value

  vRange <- numeric(length=2);

  ## Find top breakpoint
  bottom <- 1; top <- length(intVals);
  idx1 <- floor(mean(c(top, bottom)));
  idx2 <- min(c(idx1 + 1, top));
  while (TRUE) {           ## Infinite; use 'break' to escape
    if(intVals[idx1] < value) {
      bottom <- idx1;
    } else if (intVals[idx1] > value) {
      top <- idx1;
    ## All remaining cases must have intVals[idx1] == value
    } else if (idx2 == top) {    ## Can't go any higher
      vRange[2] <- top;          ##   must have found it!
    ## Note: vRange is referenced by lexical scope
      break;                     ## Breaks the while loop
    } else if (intVals[idx2] == value) {  ## 'in' range but not at top
      bottom <- idx1
    } else {                     ## Found it
      vRange[2] <- idx1;
      break;                     ## Break the while loop
    }
    idx1 <- floor(mean(c(top, bottom)));  ## Refine the interval
    idx2 <- min(c(idx1 + 1, top));
  }

  ## Find bottom breakpoint - similar to above;
  bottom <- 1; top <- length(intVals);
  idx1 <- ceiling(mean(c(top, bottom)));
  idx2 <- max(c(idx1 - 1, 1));
  while (TRUE) {
    if(intVals[idx1] < value) {
      bottom <- idx1;
    } else if (intVals[idx1] > value) {
      top <- idx1;
    } else if (idx2 == 1) {
      vRange[1] <- 1;
      return(vRange[1]:vRange[2])
    } else if (intVals[idx2] == value){
      top <- idx1
    } else {
      vRange[1] <- idx1;
      return(vRange[1]:vRange[2]);
    }
    idx1 <- ceiling(mean(c(top, bottom)));
    idx2 <- max(c(idx1 -1, 1));
  }
}
```

```
iVec2 <- c();
iVec <- sort(sample(unique(floor(runif(n=50, min=-250, max=250))), 25));
for (i in 1:5) {
  bnds <- sort(sample(1:25, 2));
  iVec2 <- c(iVec2, iVec[bnds[1]:bnds[2]])
}
iVec2 <- sort(iVec2)
iVec2

 [1] -215 -210 -199 -187 -187 -184 -184 -158 -158
[10] -158 -153 -153 -153 -148 -148 -148 -148 -136
[19] -136 -136 -103 -103 -103 -103  -70  -70  -70
[28]  -70  -58  -58  -58  -58    8    8    8    8
[37]   67   67   67   73   73   73  102  102  102
[46]  111  111  111  123  123  123  176  176  190
[55]  201

for(i in 1:5) {
  value <- sample(iVec2[-1], 1);
  cat(sprintf('Iteration %d of %d: searching for value %s ...\n',
              i, 5, as.character(value)));
  print(bSearch2(iVec2, value));
}

Iteration 1 of 5: searching for value -153 ...
[1] 11 12 13 14
Iteration 2 of 5: searching for value -153 ...
[1] 11 12 13 14
Iteration 3 of 5: searching for value -103 ...
[1] 21 22 23 24
Iteration 4 of 5: searching for value -70 ...
[1] 25 26 27 28
Iteration 5 of 5: searching for value 123 ...
[1] 49 50 51
```