Johns Hopkins School of Public Health Coding Challenge

Larry D. Lee Jr.

October 18, 2024
REVISED: November 7, 2024

Abstract: This technical note answers the questions presented in Johns Hopkins School of Public Health's (JHSPH) Coding Challenge. In this note, I review the mathematical theory behind combining datasets that use different stratifications and present code that combines these datasets.

Introduction

This technical note answers the questions presented in Johns Hopkins School of Public Health's (JHSPH) Coding Challenge. This coding challenge consisted of two parts. The challenge provided four datasets that described the age distribution of Baltimore City residents and the prevalence rates of Gonorrhea, HIV, and heroin use across different age intervals. The first part of the coding challenge asked the interviewee to answer the following questions using the datasets:

> "Using these four datasets, calculate the following quantities, or if they cannot be calculated exactly, explain why
>
> 1. The ratio of gonorrhea rate to HIV diagnosis rate among people ages 25-44, and among people ages 45 or greater, in 2020
> 2. The ratio of heroin use rate to HIV diagnosis rate among people ages 12 and older in 2020"

The second part asked the interviewee to write a function that:

> "... takes three datasets as inputs (two with rates per population and one with population) and returns the ratio of these rates by the most granular age brackets possible"

In this note, I answer these questions and present R code that implements this function. Additionally, I review the mathematics behind data stratification and associated algorithms.

Mathematical Background

In this section, I briefly review the mathematical background needed to understand how to combine stratified datasets.

General Notation

Let $s : set\ T$ represent a set of entities of type T. In general, T can represent anything, however, in this technical note, T will represent people. We will use the dot notation to reference attributes. For example, $s_i.age : \mathbb{R}^+$ will represent a person's age. Whenever we have a set s, $|s|$ will denote the number of elements within it.

Partitions

A finite **partition** of a set s is a finite set of n sets $p : set\ (set\ T)$ such that $s = \bigcup_{i=1}^{n} p_i$ and $\forall i, j \in \mathbb{N}, i \neq j \implies p_i \cap p_j = \emptyset$. A **partition function** $f_p : T \to \mathbb{N}$ is a function that accepts a value s_i from s and returns the index j of the subset p_j that contains s_i in a partition p.

As a special case we will often consider contiguous clopen interval partitions (**CCIPs**) of the positive real number line $\mathbb{R}^+$.[1] We will represent these partitions as a list of intervals in ascending order and will represent the type of such partitions as $[Interval]$. Note that there are no gaps between these intervals thus:

$$\forall p : [Interval], \forall x \in \mathbb{R}, x \geq p_0.start \ \wedge x < p_n.end \implies \exists p_i \in p, x \in p.$$

We will also consider uniform contiguous clopen interval partitions (**uCCIPs**) of the positive real number line $\mathbb{R}^+$. These are CCIPs where every interval has a uniform width w.

Stratification

A **stratification** is a partition p of some set A in which elements from s are allocated to different sets in p based some property value such as age. For this technical note, A will always be the positive real number line $\mathbb{R}^+$.

In a stratification we take a set of entities of type T, identify a property, such as age, that spans some domain, such as $\mathbb{R}^+$, partition that domain p, and then project entities into those partition sets.

[1] We call intervals that are closed at their lower bound and open at their upper bound **clopen**.

Rates

Let $f : T \to \mathbb{B}$ represent a predicate such as the function that accepts a person and returns true iff the person has a given disease. We define the **frequency** of f over a set s as:

$$freq\ (f, s) := |\{x \in s \mid f\ (x)\}|.$$

The **rate** of a f over a set s is defined as:

$$rate\ (f, s) := \frac{freq\ (f, s)}{|s|}.$$

Let p denote a finite disjoint partition of s into n subsets, then:

$$freq\ (f, s) := \sum_{i=1}^{n} freq\ (f, p_i)$$

and accordingly:

$$rate\ (f, s) = \frac{\sum_{i=1}^{n} freq\ (f, p_i)}{|s|} = \frac{\sum_{i=1}^{n} rate\ (f, p_i)\ |p_i|}{\sum_{i=1}^{n} |p_i|}. \tag{1}$$

We say that a dataset gives the rates of a predicate f over a partition p when the dataset lists the rate of f for every element (subset of s) in p.

Rate Ratios

Consider the case in which we have two different predicates $f, g : T \to \mathbb{B}$. These two functions may represent different diseases such as HIV and Gonorrhea for instance. Often, we want to compare their rates. Their **rate ratio** over a set s is given by:

$$rateRatio(f, g, s) := \frac{rate\ (f, s)}{rate\ (g, s)} = \frac{freq\ (f, s)}{freq\ (g, s)}.$$

Notice that the rate ratio can only be defined when the rates given apply to the same underlying set. We say that two sets are **compatible** if they are equal. We can calculate risk ratios using rates over two compatible sets. Also note that the risk ratio of f and g can be calculated by taking the ratio of their frequencies over s.

Refinement and Coarsening

Let p, q : $set(set\ T)$ represent two partitions of a set s. We say that q is a **refinement** of p iff for every element p_i in p there exist one or more elements in q that collectively represent a partition of p_i. In other words, there are one or more elements in q whose union equals p_i.

$$\forall p_i \in p, \exists r \subseteq q, p_i = \bigcup r.$$

Conversely, we say that p is a **coarsening** of q if q is a refinement of p.

We slightly overload our terminology by saying that r is p_i's **refinement** in q.'

We can apply this notion to contiguous clopen interval partitions of $\mathbb{R}^+$. Let p, q : $[Interval]$ represent two CCIPs of $\mathbb{R}^+$. We say that q **refines** p iff:

$$refines?\ (q, p) := \forall p_i \in p, \exists (j, n \in \mathbb{N}), p_i = \bigcup_{k=j}^{j+n} q_k.$$

Conversely, we say that p **coarsens** q. We say that q is **more granular** than p when q refines p.

Rates over Coarsening

Let p, q : $[Interval]$ be two CCIPs where q coarsens p. Let's assume furthermore that we have rates for some predicate f over p. We can use (1) to calculate the corresponding rates of f over q. To do this, we apply the following algorithm. For every interval q_i in q, identify the sublist of intervals r_i in p that refine q_i. Use (1) to calculate the rate of f over q_i:

$$rate\ (f, q_i) = \frac{\sum_{i=1}^{n} freq\ (f, r_i)}{|q_i|}.$$

If we know the proportion $prop(r_i) := |r_i|/|q_i|$ of entities in the underlying set q_i that are grouped within each interval r_i, we can rewrite this as:

$$rate\ (f, q_i) = \sum_{i=1}^{n} rate\ (f, r_i)\ prop\ (r_i).$$

Rate Ratios over Coarsening

We can calculate the rate ratios over two partitions p and q when q is a refinement of p. The rate ratios will be defined for elements (sets of s) defined by the less granular partition p. For every element p_i in p, let r_i represent p_i's refinement in q. Then the risk ratio of f and g over p is given by:

$$rateRatio(f, g, p_i, r_i) := \frac{rate\ (f, p_i)}{rate\ (g, r_i)} = \frac{freq\ (f, p_i)}{freq\ (g, r_i)}.$$

Generalization

Let's say that $p, q, \pi : [Interval]$ represent three CCIPs of $\mathbb{R}^+$. We say that π **generalizes** p and q iff:

$$generalizes?\ (\pi, p, q) := refines?(p, \pi) \wedge refines?\ (q, \pi).$$

We say that π is the **most granular generalization** of p and q iff π refines every other generalization of p and q.

We can easily prove that the intervals in π start and end at boundary points shared by both p and q. What's more, we can define an algorithm that accepts two arbitrary CCIPs and returns there most granular generalization. This algorithm is a scanning algorithm and runs in linear time. The `rebase` function defined in the code sample presented in Appendix 2 presents an implementation of this algorithm in R.

Incompatible Partitions

Consider the case where we have two datasets that have been stratified using two partitions p and q. Assume, furthermore that p and q have no common start and endpoints in common. In this instance, we say that p and q are **incompatible** as their generalization is the empty set .

In general, we cannot safely combine incompatible datasets. To see this consider the following scenario in which $p = [[0, 2]]$ and and $q = [[1, 2]]$. Assume that 100 people in p_1 have condition x and that 50 people in q_1 have condition y. We cannot safely calculate the overlap between these two groups because we do not know if all of the people in p_1 with condition x fall within the interval $[0, 1]$, $[1, 2]$, or are distributed between them. Thus, the overlap could be 0, 100%, or any value in between.

If however, we can make additional assumptions about the underlying distribution of people across the domain partitioned by p and q, such as assuming that the distribution is approximately uniform within each interval, we can use regression and interpolation techniques to estimate these overlaps.

Rounding Errors

When working with real-world datasets we have to account for the accumulation of rounding errors when combining rates across partition subsets and intervals. Imagine that we have an interval and a rate of 0.53. If we extended the accuracy of the rate to four significant digits, we find that the true rate could have been any value between 0.5250 and 0.5349. In general, when we perform arithmetic operations over these estimates, our precision will degrade.

Answers

The Interview challenge opens with the following questions:

> PART 1: Using these four datasets, calculate the following quantities, or if they cannot be calculated exactly, explain why
>
> 1. The ratio of gonorrhea rate to HIV diagnosis rate among people ages 25-44, and among people ages 45 or greater, in 2020
> 2. The ratio of heroin use rate to HIV diagnosis rate among people ages 12 and older in 2020
>
> PART 3: We have counts of new gonorrhea diagnoses in age groups: 0-14 years, 15-19 years, 20-24 years, 25-34 years, 35-39 years, 40-44 years, 45-54 years, 55-64 years, and 65+ years old. We want to estimate the count of new gonorrhea diagnoses for each individual age from 0 to 85.
>
> Describe an algorithm that will create single-year age estimates of new gonorrhea diagnosis counts. What assumptions does your algorithm make? There are a range of reasonable answers.

Part 1 Answers

I've included a code package that includes code written in R to answer these questions. We can calculate the frequency of Gonorrhea and HIV for each of the age ranges included in the given datasets. We can then calculate the sum of people in both datasets who have these conditions in the 25-44 age cohorts. Dividing these sums we find that:

1. (a) The ratio of gonorrhea rate to HIV diagnosis rate among people aged 25-44 is **19.8**.

Note that this calculation is approximate but accurate to 3 significant digits. We relied on underlying rate estimates for Gonorrhea and HIV. Both of these datasets rounded rate estimates to the nearest tenth of 1/100,000th. Thus, given an estimate such as 25.7 per 100,000, the true rate is anywhere between 25.65 and 25.74. These errors can compound enough to shift our estimate but not by enough to alter the figure given. If we extend our estimate to four significant digits the potential error ranges from 19.82 to 19.84.

1. (b) The ratio of gonorrhea rate to HIV diagnosis rate among people aged 45 and older is **6.7**

This calculation is accurate to the number of digits shown. However, given the rounding error implicit in the underlying rate estimates for Gonorrhea and HIV, the true ratio could range from 6.72 to 6.75.

1. We cannot accurately estimate the ratio of heroin use rate to HIV diagnosis rate for people ages 12 and older.

The reason we cannot estimate this datum is because the age partitions used by the HIV rate dataset and the population age distribution are not compatible (as defined above). The best that we can do is to use some form of curve fitting or interpolation to estimate population density over age and then use this to estimate the number of people falling within the age ranges used by the heroin dataset.

Part 2 Answers

The interview challenge continues by giving a specification for a function that "... takes three datasets as inputs (two with rates per population and one with population) and returns the ratio of these rates by the most granular age brackets possible". The `getRateRatio` function defined in the code sample given in Appendix 2 implements this function specification.

Part 3 Answers

In Appendix 1, we describe a range of methods for refining partitions and estimating population distributions across those refinements. Specifically, the functions `refineUniform` and `refineLinear` presented in Appendix 2 implement the algorithms described in Appendix 1.

Appendix 1: Incompatible Partitions

In this appendix, we consider those situations in which we have two datasets that use incompatible partitions.

In general, we cannot safely combine incompatible datasets. To see this consider the following scenario in which $p = [[0, 2]]$ and and $q = [[1, 2]]$. Assume that 100 people in p_1 have condition x and that 50 people in q_1 have condition y. We cannot safely calculate the overlap between these two groups because we do not know if all of the people in p_1 with condition x fall within the interval $[0, 1]$, $[1, 2]$, or are distributed between them. Thus, the overlap could be 0, 100%, or any value in between.

If however, we can make additional assumptions about the underlying distribution of people across the domain partitioned by p and q we can estimate the overlap between partition sets.

Additional Assumptions

The fundamental problem is that we do not know how a population is distributed within any partition interval p_i. If we subdivide p_i into two equal subintervals $p_{i,0}$ and $p_{i,1}$, we find that all of the people within p_i fall within one, the other, or are distributed between both.

We want to be able to refine partitions and estimate the number of people within those refined subintervals. That is, we want to be able to subdivide each partition interval p_i into a set of subintervals r_i and estimate the population falling within each interval.

To do this, we must first ask ourselves what makes a good estimate. The worst that we can do is subdivide an interval into n subintervals and discover that the entire population is concentrated into just one of them. The best that we can do is exactly estimate the proportion of the population in each of them.

Thus, we want to ensure that we are only dealing with distributions for which our maximum interval estimate error has some finite bound. This suggests that our criteria will vary based on our estimation technique. For example, if we assume that population is distributed evenly within a partition interval and the real population is not distributed evenly, the proportional estimate error is unbound.

Let $f : \mathbb{R}^+ \to \mathbb{R}^+$ represent the underlying probability distribution over some set $s : \mathbb{R}^+$ such as potential ages. It is often natural to assume that f is **differentiable** and whose derivative magnitude is bounded by some constant κ.

$$\forall x \in \mathbb{R}^+, |\frac{df(x)}{dx}| \leq \kappa.$$

Many theoretical distribution functions satisfy this requirement. For example, both the normal distribution function and the log-normal distribution functions do.

Given this assumption, we can define concrete bounds on the number of people who can potentially be in our intervals. Let p_i represent an interval $[a, a + w]$ of width w that contains $|s_i|$ people. Let's consider dividing this interval into n subintervals $r_{i,j}$ and focus on the first subinterval $r_{i,0}$. Then the minimum number of people who could be within $r_{i,0}$ occurs when f is a linear function of slope κ and assumes some minimum value $f(a)$ at the start of $r_{i,0}$. In this instance, the number of people in $r_{i,0}$ will equal:

$$||r_{i,0}|| = \int_a^{a+w/n} f(x)\ dx \geq \int_a^{a+w/n} \kappa\ x + f(a)\ dx = \frac{\kappa\ w^2}{2\ n^2} + \frac{w}{n} f(a)$$

where:

$$f(a) = \frac{|s_i|}{w} - \frac{\kappa\ w}{2}.$$

From similar calculations we find that:

$$\frac{s_i}{n} + \frac{k\ w^2}{2n}(\frac{1}{n} - 1) \leq ||r_{i,j}|| \leq \frac{s_i}{n} + \frac{k\ w^2}{2n}(1 - \frac{1}{n})$$

Uniform Estimator

Next, Let's consider how we might subdivide a partition interval and estimate the number of people within each subinterval. The simplest approach is to assume that the underlying distribution function is approximately uniformly distributed across these intervals. Thus, given an interval $p_i := [a, a+w]$ with s_i people in it, we can divide the interval into n subintervals $r_{i,j}$ and assign $m_{i,j} := s_i/n$ people to each.

Error Bounds

If we assume that the underlying distribution is differentiable with maximum derivative κ, we can prove that the maximum interval population estimate error ϵ is constrained by:

$$|\epsilon| \leq |\frac{k\ w^2}{2n}(1 - \frac{1}{n})|$$

Algorithm

The `refineUniform` function presented in Appendix 2 implements this method.

Linear Estimator

We may be able to do better than uniform estimator however. Consider a partition p, where each interval p_i has s_i people in it. Often it is reasonable to assume that increases and decreases from s_i to s_{i+1} reflect changes in the underlying distribution. If so, we can try to reflect these underlying changes by assuming that the distribution across a series of subintervals is not uniform but linearly increasing and decreasing.

To understand how we can do this, imagine that we draw line segments connecting the starting points of each interval. This produces a continuous (non-smooth) curve $g(x)$. We then assume that the underlying population distribution function $f(x)$ is approximately proportional to $g(x)$:

$$\exists \lambda \in \mathbb{R}, \forall x \in \mathbb{R}^+ f(x) \approx \lambda \ g(x).$$

Our challenge then is to calculate the scaling factor λ such the number of people allocated to each interval p_i equals the observed values $|s_i|$.

Let's focus on a single interval $p_i = [a, a+w]$. Over this interval $g(x)$ will be a linear equation thus: $\exists k_0 \in \mathbb{R}, g(x) := k_0 x + g(a)$. We will derive an estimate for the underlying distribution function $\hat{f}_i(x) := \lambda \ g(x) = \lambda(k_0 x + g(a)) = \lambda k_0 x + \hat{f}_i(a)$ that we assume approximates $f(x)$ over the interval.

Our challenge is to solve for λ.

$$|s_i| = \int_a^{a+w} \hat{f}_i(x) \ dx = \int_a^{a+w} \lambda \ g(x) \ dx = \lambda \int_a^{a+w} k_0 \ x + g(a) \ dx = \lambda(\frac{k_0 \ w^2}{2} + g(a)w)$$

Thus:

$$\lambda = \frac{2 \ |s_i|}{w \ (k_0 \ w + 2 \ g(a))}.$$

From these considerations we derive our estimator.

$$\hat{f}_i(x) := \frac{2 \ |s_i|(k_0 x + g(a))}{w(k_0 w + 2g(a))}.$$

If we divide our interval p_i into n equal subintervals $r_{i,j}$ the number of people $m_{i,j}$ we estimate lie within each subinterval will equal:

$$m_{i,j} = \int_{a+jw/n}^{a+(j+1)w/n} \hat{f}_i(x) dx$$

$$= \frac{|s_i|(2jk_0 w + k_0 w + 2n \ g(a))}{n^2(k_0 w + 2 \ g(a))}.$$

Error Bounds

We can now ask ourselves what the maximum error bounds are for this estimator. In our case. the worst possibility is for the true distribution function to be linear over each interval with a slope κ that is opposite to ours. Note that, we assume that the function is smoothed at the interval transitions to avoid discontinuities.

Algorithm

The `refineLinear` function presented in Appendix 2 implements this method.

Appendix 2: R Code Sample

```r
# This package contains definitions that can be used to represent and combine
# stratified datasets.
#
# An annotated partition is a list of contiguous non-overlapping closed-open
# intervals that span a subset of the real number line where each interval is
# associated with a value.
#
# The functions defined below assume that the intervals are listed in
# increasing order. The last interval may be unbounded.
#
# At the end of this file, we use these definitions to describe the prevalence
# of HIV and gonorrhea across Baltimore City.

UNBOUND=quote(unbound)

# Accepts two real numbers x and y that may be "unbound" and compares them.
boundCompare <- function (x, y) {
  if (x == y) 0
  else if (x < y || y == UNBOUND) -1
  else if (x > y || x == UNBOUND) 1
}

# Accepts two real numbers x and y that may be "unbound" and returns true iff x
# is less than or equal to y.
boundLte <- function (x, y) boundCompare (x, y) <= 0

# Accepts two real numbers who may be "unbound" and returns the smaller of them.
boundMin <- function (x, y) {
  if (x == UNBOUND) y
  else if (y == UNBOUND) x
  else min (x, y)
}
```

```
# Constructs an annotated interval
# These objects represent closed-open intervals in the real number line.
# @param info the "annotation value" associated with the interval
# @param start the interval's lower bound
# @param end the interval's upper bound
# Note that interval's upper bounds can be unbounded
interval <- function (info, start, end = UNBOUND) {
  list (info=info, start=start, end=end)
}

# Accepts three arguments:
# @param f a function that accepts two lists of annotations of intervals that
#   span the same range of the positive real number line and merges them
# @param xs, an annotated partition
# @param ys, an annotated partition
# and returns a new annotated partition based on the most granular intervals
# spanned by xs and ys whose annotations are derived from f.
rebase <- function (f, xs, ys) {
  zs <- list ()
  i <- 1
  j <- 1
  while (i <= length (xs) && j <= length (ys)) {
    x <- xs[[i]]
    y <- ys[[j]]
    if (x$start == y$start) {
      xInfos <- c ()
      yInfos <- c ()
      start <- x$start
      while (i <= length (xs) && j <= length (ys)) {
        x <- xs[[i]]
        y <- ys[[j]]
        if (
          boundCompare (x$end, y$end) == 0 ||
          (y$end == UNBOUND && i == length (xs)) ||
          (x$end == UNBOUND && j == length (ys))
        ) {
          zs <- append (zs,
            list (interval (
              info=f (
                c (xInfos, x$info),
                c (yInfos, y$info)
              ),
              start=start,
              end=boundMin (x$end, y$end)
          )))
          i <- i + 1
          j <- j + 1
          break
        } else if (boundCompare (x$end, y$end) == -1) {
          xInfos <- c (xInfos, x$info)
          i <- i + 1
        } else { # x$end > y$end
          yInfos <- c (yInfos, y$info)
          j <- j + 1
        }
      }
    } else if (x$start < y$start) {
      i <- i + 1
    } else { # x$start > y$start
      j <- j + 1
    }
  }
  zs
}

# Accepts two partitions:
# @param ps a partition that represents the number of people in a population who
#   fall within a contiguous set of age ranges
# @param rs a partition that represents the proportion of people within a
#   contiguous set of age ranges who have some condition
# and returns a partition that gives the absolute number of people who have the
# condition within the most granular set of age ranges spanned by both
# ps and rs.
getFrequency <- function (ps, rs) {
  rebase (
    function (sizes, rates) {
      if (length (rates) != 1) {
        stop ("Error: the population partition is not a \"refinement\" of the rate partitio
      }
      rates[1]*sum (sizes)
    }, ps, rs
  )
}

# Accepts three arguments:
# @start the age range lower bound
# @end the age range upper bound
# @freqs the number of people who have a given condition for a set of age ranges
# and returns the total number of people who have the given condition between
# the start and end age range (inclusive).
getFreqSum <- function (start, end, freqs) {
  sum <- 0
  for (freq in freqs) {
    if (!boundLte (freq$end, end)) break

    if (start <= freq$start) sum <- sum + freq$info
  }
  sum
}

# Accepts two partitions:
# @param xs a partition that gives the number of people within a contiguous set
#   of age ranges who have condition x
# @param ys a partition that gives the number of people within a contiguous set
#   of age ranges who have condition y
# and returns the rate ratios of the two conditions over the most granular
# partition spanned by both xs and ys.
getFreqRateRatio <- function (xs, ys) {
  rebase (function (freqs0, freqs1) sum (freqs0)/sum (freqs1), xs, ys)
}

# Accepts three partitions:
# @param ps a partition that gives the number of people who fall into a
#   contiguous set of real intervals
# @param xs a partition that gives the proportion of people within a range of
#   intervals who have a condition x
# @param ys a partition that gives the proportion of people within a range of
#   intervals who have a condition y
# and returns the rate ratio of the number of people who have condition x and y
# over the most granular common partition that spans xs and ys.
getRateRatio <- function (ps, xs, ys) {
  xFreq <- getFrequency (ps, xs)
  yFreq <- getFrequency (ps, ys)
  getFreqRateRatio (xFreq, yFreq)
}

# Accepts one argumnet:
# @param xs, an annotated partition
# and returns a new annotated partition in which every interval in xs has been
# subdivided into subintervals of width one and the value assigned to those
# intervals has been distributed equally across the subintervals.
# Note: This function skips all unbound intervals.
refineUniform <- function (xs) {
  ys <- list ()
  for (i in 1:length (xs)) {
    x <- xs[[i]]
    if (x$start == UNBOUND || x$end == UNBOUND) next
    w <- x$end - x$start
    for (j in 0:(n - 1)) {
      ys <- append (ys,
        list (interval (
          info=x$info/w,
          start=x$start + j,
          end=x$start + j + 1
      )))
    }
  }
  ys
}

# Accepts one argument:
# @param xs, an annotated partition
# and returns a new annotated partition in which every interval in xs has been
# subdivided into equal subintervals of width one and the value assigned to
# those intervals has been distributed across the subintervals using the
# linear (not the uniform) estimator.
# Note: This function skips all unbound intervals.
refineLinear <- function (xs) {
  ys <- list ()
  for (i in 1:length (xs)) {
    x <- xs[[i]]
    if (x$start == UNBOUND || x$end == UNBOUND) next
    prevInfo <- if (i - 1 > 0) xs[[i - 1]]$info else 0
    nextInfo <- if (i + 1 <= length (xs)) xs[[i + 1]]$info else 0
    w <- x$end - x$start
    k <- (nextInfo - x$info)/w
    for (j in 0:(w - 1)) {
      ys <- append (ys,
        list (interval (
          info=x$info*(2*j*k*w + k*w + 2*w*prevInfo)/
               ((w^2)*(k*w + 2*prevInfo)),
          start=x$start + j,
          end=x$start + j + 1
      )))
    }
  }
  ys
}

# A partition listing the number of Baltimore residents who's ages fall within
# a contiguous set of age ranges.
population = list (
  interval (36355, 0, 4),
  interval (33773, 5, 9),
  interval (33590, 10, 14),
  interval (33872, 15, 19),
  interval (37183, 20, 24),
  interval (53357, 25, 29),
  interval (54804, 30, 34),
  interval (43408, 35, 39),
  interval (34271, 40, 44),
  interval (30273, 45, 49),
  interval (33423, 50, 54),
  interval (37639, 55, 59),
  interval (36895, 60, 64),
  interval (29868, 65, 69),
  interval (22486, 70, 74),
  interval (13910, 75, 79),
  interval (8977, 80, 84),
  interval (9073, 85)
)

# A partition listing the rates of HIV amongst Baltimore residents falling
# within certain age ranges.
hivRates = list (
  interval (45.6e-5, 13, 24),
  interval (53.6e-5, 25, 34),
  interval (46.6e-5, 35, 44),
  interval (26.7e-5, 45, 54),
  interval (5.4e-5, 55, 64),
  interval (30.4e-5, 65)
)

#  A partition list the rates of Gonorrhea amongst Baltimore residents falling
# within certain age ranges.
gonorrheaRates = list (
  interval (25.7e-5, 0, 14),
  interval (2021.6e-5, 15, 19),
  interval (2647.6e-5, 20, 24),
  interval (1477.8e-5, 25, 29),
  interval (1047.3e-5, 30, 34),
  interval (773.0e-5, 35, 39),
  interval (490.3e-5, 40, 44),
  interval (298.6e-5, 45, 54),
  interval (139.5e-5, 55, 64),
  interval (23.6e-5, 65)
)

# A partition listing the rates of heroin use amongst Baltimore residents
# falling within certain age ranges
# Note: The handout appears to have a typo in which heroin usage rates are
# reported as "rates per 100,000"
heroinRates = list (
  interval (0.00112, 12, 17),
  interval (0.005426598, 18, 25),
  interval (0.009849983, 26)
)

# A partition listing the number of Baltimore residents who's ages fall within
# a contiguous set of age ranges.
populationRefine = list (
  interval (103718, 0, 14),
  interval (33872, 15, 19),
  interval (37183, 20, 24),
  interval (108161, 25, 34),
  interval (43408, 35, 39),
  interval (34271, 40, 44),
  interval (63696, 45, 54),
  interval (74534, 55, 64),
  interval (84314, 65)
)

# The number of people within certain age ranges who have HIV
hivFreq = getFrequency (population, hivRates)

# The number of people within certain age ranges who have Gonorrhea
gonorrheaFreq = getFrequency (population, gonorrheaRates)

# The number of people within certain age ranges who use heroin
heroinFreq = getFrequency (population, heroinRates)

# The rate ratio of gonorrhea and HIV for people aged 25 to 44
# Note: the answer to question 1.(a)
gonorrheaHivFreq2544 =
  getFreqSum (25, 44, gonorrheaFreq)/
  getFreqSum (25, 44, hivFreq)

# The rate ratio of gonorrhea and HIV for people aged 45 and older
# Note: the answer to question 1.(b)
gonorrheaHivfreq =
  getFreqSum (45, UNBOUND, gonorrheaFreq)/
  getFreqSum (45, UNBOUND, hivFreq)

# Estimates the number of people within the age ranges used to partition
#   populationRefine that have gonorrhea.
# Note: this simulates the dataset that could have been provided for 3.
```

```
gonorrheaFreqRefine = rebase (
  function (xs, ys) sum (xs),
  refineLinear (gonorrheaFreq),
  populationRefine)

# Estimates the number of people for each age cohort who has gonorrhea.
# Note: this answers question 3.
gonorrheaFreqLinearRefinement = refineLinear (gonorrheaFreqRefine)
```