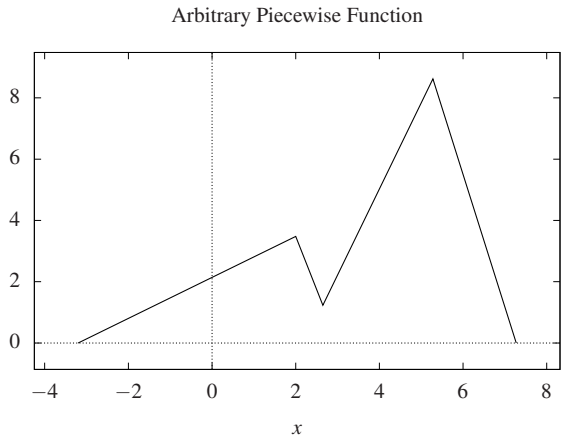
All students should read §7.5–7.6.1 in preparation for the next assignment and the next lecture.

The following numbered questions should be split across your group and the solutions discussed during the next lecture period. Students should review the learning goals for the day, determine which are applicable to their questions and provide answers or commentary to their group members. When using the Internet to formulate answers (some questions may require this), keep track of **where** you find your information on the web. You may be asked for, and are expected to have (in Email-able form), URLs supporting your investigations.

1. Question 7.1.9 (§7.1.3); also calculate the algebraic variate $F^{-1}(u)$.

> **BEWARE: the following is a big question, no group should expect anyone to do more than this question 2 for the assignment. If you're in a big group, consider two people pair-programming this solution — or using some sort of code collaboration effort. You'll see evidence that the solution is correct, so "double coverage" isn't required.**

2. Consider a data file such as `piecewise-function.dat` (plotted at the right) that represents a piecewise linear function defined by k points in the first two quadrants of the xy-cartesian plane:



$$\{(x_i, y_i)\}_{i=1}^{k}, \quad k \geq 2, \quad x_i < x_{i+1}, \quad y_i >= 0 \qquad (1)$$

As you can probably surmise by inspection: the data is simply one point per line, the points being line-by-line sequential in the file, with k lines in the file.

Write code that performs the following steps:

(a) Reads points (x_i, y_i) from a disk file (assume the function has a finite domain of course, it's a finite disk file), **and normalizes the area under the piecewise curve to** 1 by scaling the y_i appropriately. (You have created a piecewise continuous probability distribution $f(x)$ with this step).

(b) After normalizing, calculate the CDF values $F(x_i)$ for the k x_i ($F(x_1) = 0$). Notice two things:

i These are **cumulative areas**, so the $F(x_2) < F(x_3) < F(x_4)$.

ii Some of the areas under line segments are triangles, some are *trapezoids*.

Store these per-segment CDF values in a data structure so that the $(i, F(x_i))$ pairs can be easily searched.

(c) Now, for segments $2, \ldots, k$, calculate the **per segment** quadratic coefficients A_i, B_i, C_i that represent the segment's $\mathcal{F}_i(x)$. For clarity, we use a script $\mathcal{F}_i$ for the quadratic equation (integral of $f_i(x)$) associated with segment i, this isn't the CDF for segment i per se:

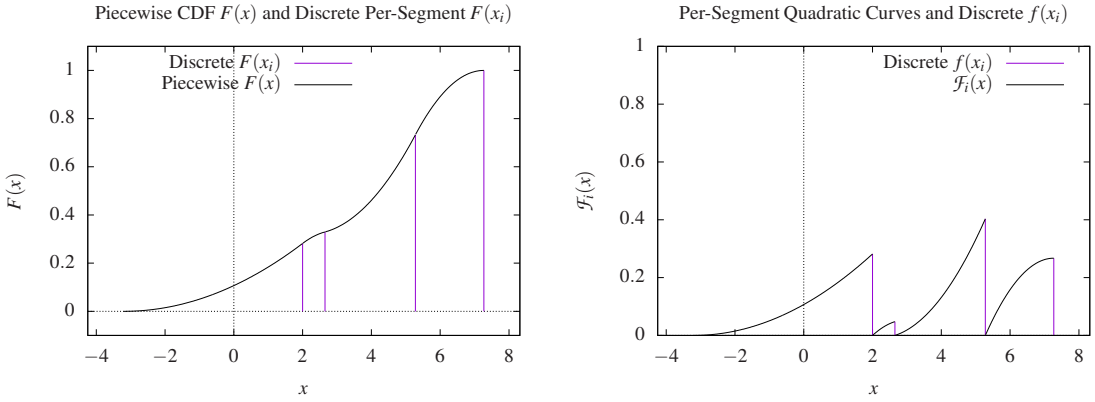
$$F(x_1) = 0 \qquad F(x) = F(x_{i-1}) + \mathcal{F}_i(x) \quad x \in (x_{i-1}, x_i])$$

See the plots on the following page for a visualization of what you're doing and $F_i(x)$ vs $\mathcal{F}_i$.

This is a long LGA problem, so you're saved the pencil work:

$$\mathcal{F}_i(x) = \underbrace{\frac{m_i}{2}}_{A_i} x^2 + \underbrace{(y_{i-1} - m_i x_{i-1})}_{B_i} x + \underbrace{\left(\frac{m_i}{2} x_{i-1}^2 - y_{i-1} x_{i-1}\right)}_{C_i}$$

where m_i is the slope of the line segment from (x_{i-1}, y_{i-1}) to (x_i, y_i). Store A_i, B_i, and C_i so they can be easily retrieved for each segment i. Notice from (1), that you may assume no two points in the piecewise function have the same x values, so there are no "divide by zero" concerns for m_i.



(d) Finally, implement a variate algorithm ($F^{-1}(u)$) that produces points from the distribution created in part a; the algorithm will be **syncronized** and have **monotonicity** (see the reading in chapter 6). The specific steps to take for each $u \leftarrow Random()$ are:

i. Find the left-most segment i with $F(x_i) > u$, which is the definition of $F^*(u)$ for **discrete distributions**.

ii. Decrease u so that it is less than $F_i(x_i)$:

$$u' = u - F(x_{i-1})$$

Geometrically, the value of u' is now between 0 and the area of segment i.

iii. Solve

$$u' = \mathcal{F}_i(x) = A_i x^2 + B_i x + C_i \quad \text{using} \quad x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

which I hope you all remember as the quadratic equation. Recall this solves for the $y = 0$ root of the equation, so $c = C_i - u'$. Of course, you have two candidates for x due to the $\pm$, but one of them is gauranteed to fall within $[x_{i-1}, x_i]$.

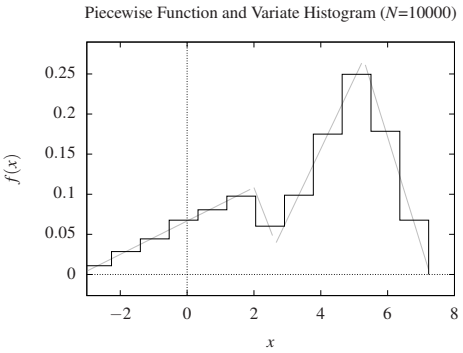
Your program should have the following command line interface:

```
$ ./piecewisevariate  points.dat  seed  N
```

Where the application produces `N` points using `seed` for the $Random()$ draws of $F^{-1}(u)$ in part d.

Hint: An ideal implementation of $F^{-1}(u)$ would use binary search to find the piecewise linear segment of interest. You might even be able to use the binary search logic from your group's `lga-discrete-random-variates.pdf` solution. (Yes! You are actually calculating a *discrete* random variable's CDF in part b!)

(e) Demonstrate with a histogram or empirical CDF that your code works as expected for the provided data file. Something like this would be convincing evidence:



This graph shows the underlying distribution as a light grey line, beware that this is the *normalized* distribution from part a, not the original function! (I don't think you need to show this, by the way, I just don't want you thrown off by the discrepency.)

3. Consider the **flawed** non-stationary interarrival rate algorithm presented in the text (§7.5.1) and considered briefly in lecture (show_non-stationary-arrivals.pdf):

$$a_{i+1} \leftarrow a_i + Exponential\left(\frac{1}{\lambda(t)}\right) \tag{2}$$

and the **thinning algorithm** which is a correct, **accept-reject technique** for generating non-stationary arrival times. This question will have you compare the distribution of arrival times between the two techniques.

For each of the following arrival rate functions

A. $\lambda_L = 3t + \frac{1}{10}$

B. $\lambda_C = \cos\left(\frac{2\pi t}{5}\right) + 1$

C. $\lambda_S = \begin{cases} \text{if } x = t - \lfloor \frac{t}{10} \rfloor 10 \leq 5 & \text{then} \quad 1 + \frac{x}{2} \\ \text{if } x = t - \lfloor \frac{t}{10} \rfloor 10 > 5 & \text{then} \quad \frac{7}{2} - \frac{x-5}{2} \end{cases}$

λ_S generates a sawtooth pattern 10 time units wide with a height ranging from 1 to $\frac{7}{2}$.

Use the following algorithm to generate n arrivals within a (mock) simulation time of $\tau = 8\pi$:

```
sims ← 0
while( n > 0 ) do (
    a ← 0
    while( n > 0 ) do
        a = NextArrival(a, λ(t))
        if( a > τ ) break
        store a as an arrival time
        decrement n
    )
    increment sims
)
```

Since we need to analyze all the arrival times en masse, it is OK to store them in a list or set data structure. The *NextArrival* function should implement either the flawed next-arrival equation (2) (which would need to know $\lambda(a)$ or the function definition of $\lambda(t)$) or the **thinning algorithm** (which would also need to know $\lambda_{\max}$). How you implement this in your preferred language is up to you — the nugget of pseudo code is just a guide.

Once you have everything coded up satisfactorily, do the following for each arrival rate function $\lambda_L(t)$, $\lambda_C(t)$, $\lambda_S(t)$:

(a) First, generate two or three sets of flawed arrival data using different seeds, plot them together so you get a feel for how results vary due to simple randomness. (see the note on the next page for a hint on how to do this easily).

(b) Then generate a set of good arrival data using the thinning algorithm and compare them to the flawed results.

(c) How are the flawed arrival times different than the correct non-stationary arrival times? Are there characteristics or features of the $\lambda(t)$ functions that seem to exacerbate the differences? Summarize your results and provide plots supporting your conclusions. Be sure to always use the same N when comparing the arrival times generated by the two techniques. You are welcome to investigate other $\lambda(t)$ functions as well :)

Comparing arrival times hint: There are two graphical ways to compare the arrival times generated by these two methods. The first is to look at the **cumulative arrival function**, which is similar to an emperical CDF — an example is in the show_non-stationary-arrivals.pdf lecture slides. But the cumulative function does not really give you a feel for **how** the arrivals differ.
An easy solution is to present **histogram like plots** for the various sets of arrival times; ideally plotted against each other on the same axes.

1. First decide on the number of bins k. The same k guidelines for density histograms work well here (show-histograms-cedf.pdf).

$$\lfloor \log_2 n \rfloor \leq k \leq \lfloor \sqrt{n} \rfloor \text{ with a bias towards } k \approx \left\lfloor \frac{5}{3}\sqrt[3]{n} \right\rfloor$$

2. Calculate the bin width $\delta = \frac{\tau}{k}$, in this case δ is an interval of time.
3. Count all the arrivals that occur within each bin and divide by *sims*, the number of simulations required to generate N total arrivals per $\lambda_*(t)$. This is the average number of arrivals in the particular bin i, call it α_i.
4. Lastly, divide all the α_is by δ: $\frac{\alpha_i}{\delta}$. This is the number of arrivals per time interval, in other words the **arrival rate**; the same as $\lambda_*(t)$. So the contour of this plot should match $\lambda_*(t)$.

Hint: If you already have a script or procedure for calculating your preferred k for the **density histograms** we prefer in this course. Then the above steps amount to multiplying the "density height" by $\frac{n}{sims}$.

[0]If you *really* want them to match you can pursue this in your leisure time.