CSE1729	Introduction to Programming Problem Set 6 Solutions	March 24, 2017

1. *Pearson's Coefficient* is widely used in the natural sciences as a measure of correlation between two variables. Given two lists, $X = (x_1\ x_2\ \ldots\ x_n)$ and $Y = (y_1\ y_2\ \ldots\ y_n)$, each containing n values, Pearson's Coefficient is defined by the expression:

$$\frac{\sum_{i=1}^{n}(x_i-\bar{X})(y_i-\bar{Y})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{X})^2}\sqrt{\sum_{i=1}^{n}(y_i-\bar{Y})^2}},$$

where $\bar{X}$ and $\bar{Y}$ denote the averages of the x_i and the y_i, which is to say that

$$\bar{X} = \frac{1}{n}\sum_i x_i \quad \text{and} \quad \bar{Y} = \frac{1}{n}\sum_i y_i\,.$$

(a) Write a function that takes a list and returns its sum.

```
(define (list-sum elements)
  (if (null? elements)
      0
      (+ (car elements)
         (list-sum (cdr elements)))))
```

(b) Write a function that takes a list $X = (x_1\ x_2\ \ldots\ x_n)$ and returns the average $\bar{X}$. (Note: You will have to compute the length of the list in order to compute the average.)

```
(define (average X)
  (/ (list-sum X)
     (length X)))
```

(c) Write a function that maps a list X to the square of its deviation. Thus the list $X = (x_1\ x_2\ \ldots\ x_n)$ should be carried to the list

$$((x_1-\bar{X})^2\ \ldots\ (x_n-\bar{X})^2).$$

You should use the `map` function. (For example, your function, when evaluated on the list (1 2 3 4 5), should return (4 1 0 1 4).)

```
(define (var-map X)
  (define (square x) (* x x))
  (let ((mean (average X)))
    (map (lambda (x) (square (- x mean))) X)))
```

(d) Write a function that returns the *standard deviation* of a list, $\sqrt{(1/n)\sum_{i=1}^{n}(x_i-\bar{X})^2}$. This should be easy, you already have the functions you need.

```
(define (stdev X)
  (sqrt (average (var-map X))))
```

(e) Write a new version (called `map2`) of the `map` function which operates on two lists. Your function should accept three parameters, a function f and two lists X and Y, and return a new list composed of the function f applied to corresponding elements of X and Y. In particular, given two lists $(x_1\ x_2\ \ldots\ x_n)$ and $(y_1\ y_2\ \ldots\ y_n)$ (and the function f), `map2` should return

$$(f(x_1,y_1)\ \ldots\ f(x_n,y_n)).$$

```
(define (map2 f X Y)
  (if (null? X)
      '()
      (cons (f (car X) (car Y))
            (map2 f (cdr X) (cdr Y)))))
```

(f) Write a function that, given two lists $X = (x_1\ x_2\ \ldots\ x_n)$ and $Y = (y_1\ y_2\ \ldots\ y_n)$, computes the covariance list:

$$((x_1-\bar{X})(y_1-\bar{Y})\ \ldots\ (X_n-\bar{X})(Y_n-\bar{Y}))\,.$$

```
(define (covar-elements X Y)
  (let ((meanX (average X))
        (meanY (average Y)))
    (map2 (lambda (x y)
            (* (- x meanX)
               (- y meanY)))
          X Y)))
```

(g) Write a function that computes Pearson's Coefficient. It might be useful to observe that an equivalent way to write Pearson's coefficient, by dividing the top and bottom by n, is

$$\frac{{}^1\!/_n \sum_{i=1}^{n}(x_i-\bar{X})(y_i-\bar{Y})}{\sigma(X)\sigma(Y)}\,.$$

```
(define (pearson X Y)
  (/ (average (covar-elements X Y))
     (* (stdev X) (stdev Y))))
```

2. *Least-Squares Line Fitting.* Line fitting refers to the process of finding the "best" fitting line for a set of points. This is one of the most fundamental tools that natural scientists use to fit mathematical models to experimental data.

   As an example, consider the red points in the scatter plot of Figure 1. They do not lie on a line, but there is a line that "fits" them very well, shown in black. This line has been chosen—among all possible lines—to be the one that minimizes the sum of *squares* of the vertical distances from the points to the line. (This may seem like an odd thing to minimize, but it turns out that there are several reasons that it is the "right" thing minimize.)

   Since the equation of a line is $y = a+bx$, this boils down to finding a slope b and x-intercept a for given lists X and Y corresponding to point data. It turns out that the best fitting line can be found with the following two equations:

$$b = r \cdot \frac{\sigma(Y)}{\sigma(X)}\,, \quad \text{and} \quad a = \bar{Y} - b\bar{X}\,,$$

   where $\sigma(X)$ and $\sigma(Y)$ refer to the standard deviations of X and Y and r is Pearson's Coefficient.

   (a) Write a function that takes two lists X and Y (of the same length) and returns a pair (a, b) corresponding to the best fit line for the data.

```
(define (best-fit pX pY)
  (let* ((a (* (pearson pX pY) (stdev pY) (/ 1 (stdev pX))))
         (b (- (average pY) (* a (average pX)))))
    (cons a b)))
```

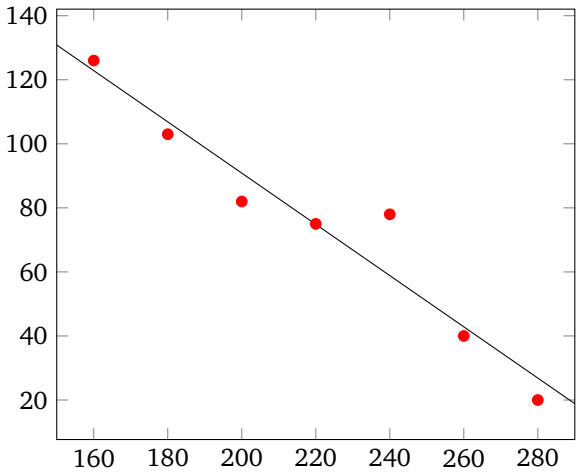

Figure 1: An example of a scatter plot and a best-fit line.

(b) Write an alternate version of `best-fit` that returns the best fitting line as a *function*. Specifically, write a new function `best-fit-fn` so that `(best-fit-fn pX pY)` returns the *function* which, given a number x, returns $ax + b$ where a and b are the best fit coefficients.

Use this to the best fit line for the following data:

$$X = \{160, 180, 200, 220, 240, 260, 280\}, \qquad Y = \{126, 103, 82, 75, 78, 40, 20\}.$$

Call the function `fitline` so that you can graph it in the next problem.

```
(define (best-fit-fn pX pY)
  (let* ((a (* (pearson pX pY) (stdev pY) (/ 1 (stdev pX))))
         (b (- (average pY) (* a (average pX)))))
    (lambda (x) (+ (* a x) b))))

(define fitline (best-fit-fn X Y))
```

(c) We would like to plot the original data along with the best fit line. To do this we will need to "zip" lists X and Y into a list of data points. You wrote `zip` in last week's homework. Rewrite this function as `zip2` and make use of `map2` function (the implementation should be much simpler). The plotting package requires a list of *vectors* rather than *pairs*. Vectors haven't been covered in class, but it will suffice to replace the `cons` keyword with `vector` in your implementation of `zip2`.

Now plot the best fit line and data using the following command:

```
(plot (list (axes)
            (function fitline 140 300)
            (points (zip2 X Y))))
```

```
(define (zip2 la lb)
  (if (null? la)
      '()
      (cons (vector (car la) (car lb))
            (zip2 (cdr la) (cdr lb)))))
```

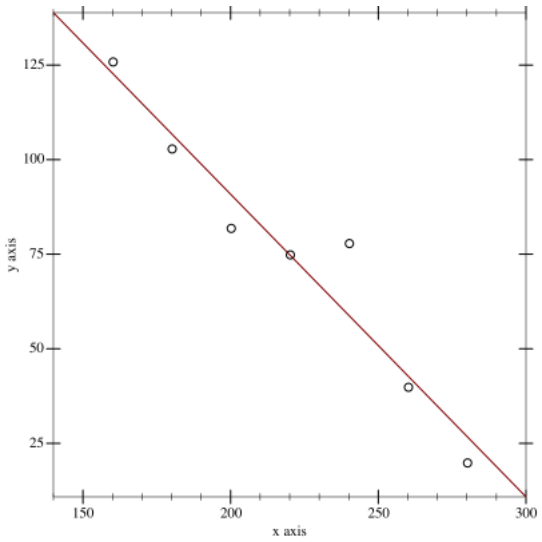

Figure 2: The result of the Racket plot.

This produces the image of Figure 2.

3. Define a SCHEME function, `merge`, which takes two lists ℓ_1 and ℓ_2 as arguments. Assuming that each of ℓ_1 and ℓ_2 are *sorted* lists of integers (in increasing order, say), `merge` must return the sorted list containing all elements of ℓ_1 and ℓ_2.

   To carry out the merge, observe that since ℓ_1 and ℓ_2 are already sorted, it is easy to find the smallest element among all those in ℓ_1 and ℓ_2: it is simply the smaller of the first elements of ℓ_1 and ℓ_2. Removing this smallest element from whichever of the two lists it came from, we can recurse on the resulting two lists (which are still sorted), and place this smallest element at the beginning of the result.

```
(define (merge la lb)
  (cond ((null? la) lb)
        ((null? lb) la)
        ((< (car la) (car lb)) (cons (car la)
                                     (merge (cdr la) lb)))
        (else (cons (car lb)
                    (merge la (cdr lb))))))
```