PSP0.1 Homework

Develop the program specified on pages 3-11 using the PSP0.1 process. To follow the PSP0.1 process, make use of the process scripts available on Process dashboard. Before starting program 2, review the top-level PSP0.1 process script.

If you are already following a C standard for your programs, you can stick to it consistently. Otherwise, use the C Coding Standard example provided on page 11. It was adapted from Humphrey's simple standard for C++.

Use your Program 1 (from PSP0 Homework) as the base code to which you may add, delete from or modify code to create Program 2. The program should work correctly in a linux environment.

Record the following information on the Process Dashboard, as described in the process scripts (refer to process scripts for more information on these):

- Your estimated added and modified size of Program 2 on the Project Plan Summary form (in Planning phase).
- Your estimated total development time on the Project Plan Summary form (in Planning phase).
- The time taken for each PSP0 phase.
- Details of all the defects found/fixed in each phase.
- Enter the actual Program 2 size data obtained using the LOC Counter available on Process Dashboard (in Postmortem phase).
- Enter any process problems and solutions in the PIP form (in Postmortem phase).

Remember to mark the completion of each phase and the Project overall on the Dashboard before saving your data for submission.

Submit the following in a zip file by the due date and time:

- PSP data from Process Dashboard retrieved using the save data backup function.
- Completed PIP form
- Your well documented C program as a text file with .c extension.
- Input data file used for testing the program.
- A document showing the program test results including inputs/outputs and run time messages as evidence that the program works correctly.

An important note:

Programming environments include a wide variety of mathematical and other embedded functions. While using embedded functions for some program actions is appropriate, they should not be used to write the principal functions of the exercise program. When students use such functions, they can write many of the PSP exercises in fewer than ten LOC. This takes too little time and is of no value in a PSP exercise which is focused on the process of program development. If students use these functions, they will not be fulfilling the purpose of the exercise and so will not receive credit for doing the homework.

Program 2

Program 2 requirements

Program 2 requirements

Using your Program 1 of PSP0 Homework as the base code, develop a program in C to do the following:

- calculate the linear regression parameters β_0 and β_1 and correlation coefficients $r_{x,y}$ and r^2 for a set of *n* pairs of data,
- given an estimate, x_k calculate an improved prediction, y_k where
  $y_k = \beta_0 + \beta_1 x_k$
- enhance the linked list developed in program 1 to store the *n* data sets, where each record holds two real numbers

The program should accept input data from a file. Table 1 contains historical estimated and actual data for 10 programs. For program 11, the developer has estimated a proxy size of 386 LOC.

Thoroughly test the program. At a minimum, run the following four test cases.

- Test 1: Calculate the regression parameters and correlation coefficients between estimated proxy size and actual added and modified size in Table 1. Calculate plan added and modified size given an estimated proxy size of x_k = 386.
- Test 2: Calculate the regression parameters and correlation coefficients between estimated proxy size and actual development time in Table 1. Calculate time estimate given an estimated proxy size of x_k = 386.
- Test 3: Calculate the regression parameters and correlation coefficients between plan added and modified size and actual added and modified size in Table 1. Calculate plan added and modified size given an estimated proxy size of x_k = 386.
- Test 4: Calculate the regression parameters and correlation coefficients between plan added and modified size and actual development time in Table 1. Calculate time estimate given an estimated proxy size of x_k = 386.

Expected results are provided in Table 2.

Program Number	Estimated Proxy Size	Plan Added and Modified size	Actual Added and Modified Size	Actual Development Hours
1	130	163	186	15.0
2	650	765	699	69.9
3	99	141	132	6.5
4	150	166	272	22.4
5	128	137	291	28.4
6	302	355	331	65.9
7	95	136	199	19.4
8	945	1206	1890	198.7
9	368	433	788	38.8
10	961	1130	1601	138.2

Table 1

Continued on next page

Expected results

Test	Expected Values					Actual Values				
	β_0	β_1	$r_{x,y}$	r^2	y_k	β_0	β_1	$r_{x,y}$	r^2	y_k
Test 1	-22.55	1.7279	0.9545	0.9111	644.429					
Test 2	-4.039	0.1681	0.9333	.8711	60.858					
Test 3	-23.92	1.43097	.9631	.9276	528.4294					
Test 4	-4.604	0.140164	.9480	.8988	49.4994					

Table 2

Regression

Overview

Linear regression is a way of optimally fitting a line to a set of data. The linear regression line is the line where the distance from all points to that line is minimized. The equation of a line can be written as

$y = \beta_0 + \beta_1 x$

In Figure 1, the best fit regression line has parameters of β_0 = -4.0389 and β_1 = 0.1681.

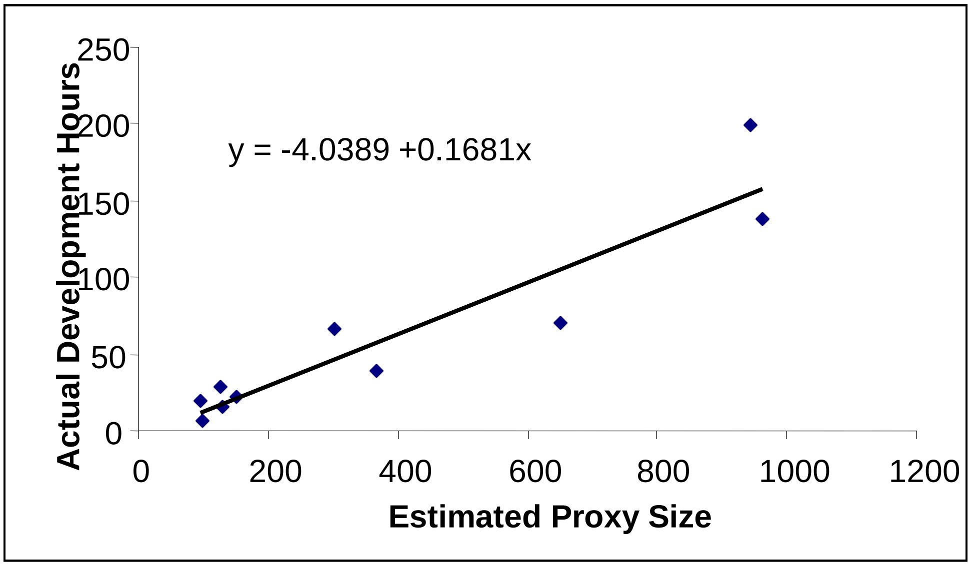

Figure 1

Continued on next page

Using regression in the PSP

Looking at Figure 1, how many hours do you think it would take to develop a program with an estimated proxy size of 500?

Using PROBE method A for time, the estimate would be $TimeEstimate = \beta_0 + \beta_1(500)$ or an estimate of 80.011 hours.

The PSP PROBE method uses regression parameters to make better predictions of size and time based on your historical data.

PROBE methods A and B differ only in the historical data (*x* values) used to calculate the regression parameters. In PROBE method A, **estimated proxy** size are used as the *x* values. In PROBE method B, **plan added and modified** size are used as the *x* values.

PROBE methods for size and time differ only in the historical data (*y* values) used to calculate the regression parameters. To predict improved size estimates, **actual added and modified LOC** are used as the *y* values. To predict time estimates, **actual development times** are used as the *y* values.

Historical Data Used		***x* values**	***y* values**
Size Estimating	PROBE A	Estimated Proxy Size	Actual Added and Modified Size
	PROBE B	Plan Added and Modified Size	Actual Added and Modified Size
Time Estimating	PROBE A	Estimated Proxy Size	Actual Development Time
	PROBE B	Plan Added and Modified Size	Actual Development Time

Correlation

Overview

The correlation calculation determines the relationship between two sets of numerical data.

The correlation $r_{x,y}$ can range from +1 to -1.

- Results near +1 imply a strong positive relationship; when x increases, so does y.
- Results near -1 imply a strong negative relationship; when x increases, y decreases.
- Results near 0 imply no relationship.

Using correlation in the PSP

Correlation is used in the PSP to judge the quality of the linear relation in various historical process data that are used for planning. For example, the relationships between estimated proxy size and actual time or plan added and modified size and actual time.

For this purpose, we examine the value of the relation r_{xy} squared, or r^2.

If r^2 is	the relationship is
$.9 \le r^2$	predictive; use it with high confidence
$.7 \le r^2 < .9$	strong and can be used for planning
$.5 \le r^2 < .7$	adequate for planning but use with caution
$r^2 < .5$	not reliable for planning purposes

Limitations of correlation

Correlation doesn't imply cause and effect.

A strong correlation may be coincidental.

From 1840 to 1960, no U.S. president elected in
a year ending in 0 survived his presidency.
Coincidence or Correlation?

Many coincidental correlations may be found in historical process data.

To use a correlation, you must understand the cause-and-effect relationship in the process.

Calculating regression and correlation

Calculating regression and correlation

The formulas for calculating the regression parameters β_0 and β_1 are

$$\beta_1 = \frac{\left(\sum_{i=1}^{n} x_i y_i\right) - \left(n x_{avg} y_{avg}\right)}{\left(\sum_{i=1}^{n} x_i^2\right) - \left(n x_{avg}^2\right)}$$

$$\beta_0 = y_{avg} - \beta_1 x_{avg}$$

The formulas for calculating the correlation coefficient $r_{x,y}$ and r^2 are

$$r_{x,y} = \frac{n\left(\sum_{i=1}^{n} x_i y_i\right) - \left(\sum_{i=1}^{n} x_i\right)\left(\sum_{i=1}^{n} y_i\right)}{\sqrt{\left[n\left(\sum_{i=1}^{n} x_i^2\right) - \left(\sum_{i=1}^{n} x_i\right)^2\right]\left[n\left(\sum_{i=1}^{n} y_i^2\right) - \left(\sum_{i=1}^{n} y_i\right)^2\right]}}$$

$$r^2 = r * r$$

where

- Σ is the symbol for summation
- *i* is an index to the *n* numbers
- *x* and *y* are the two paired sets of data
- *n* is the number of items in each set *x* and *y*
- x_{avg} is the average of the *x* values
- y_{avg} is the average of the *y* values

An example

An example

In this example, we will calculate the regression parameters (β_0 and β_1 values) and correlation coefficients $r_{x,y}$ and r^2 of the data in the Table 3.

n	*x*	*y*
1	130	186
2	650	699
3	99	132
4	150	272
5	128	291
6	302	331
7	95	199
8	945	1890
9	368	788
10	961	1601

Table 3

$$\beta_1 = \frac{\left(\sum_{i=1}^{n} x_i y_i\right) - \left(n x_{avg} y_{avg}\right)}{\left(\sum_{i=1}^{n} x_i^2\right) - \left(n x_{avg}^2\right)}$$

1. In this example there are 10 items in each dataset and therefore we set n = 10.
2. We can now solve the summation items in the formulas.

n	*x*	*y*	x^2	*x*y*	y^2
1	130	186	16900	24180	34596
2	650	699	422500	454350	488601
3	99	132	9801	13068	17424
4	150	272	22500	40800	73984
5	128	291	16384	37248	84681
6	302	331	91204	99962	109561
7	95	199	9025	18905	39601
8	945	1890	893025	1786050	3572100
9	368	788	135424	289984	620944
10	961	1601	923521	1538561	2563201
Total	$\sum_{i=1}^{10} x_i = 3828$	$\sum_{i=1}^{10} y_i = 6389$	$\sum_{i=1}^{10} x_i^2 = 2540284$	$\sum_{i=1}^{10} x_i y_i = 4303108$	$\sum_{i=1}^{10} y_i^2 = 7604693$
	$x_{avg} = \frac{3828}{10} = 382.8$	$y_{avg} = \frac{6389}{10} = 638.9$			

Continued on next page

An example, cont.

3. We can then substitute the values into the formulas

$$\beta_1 = \frac{(4303108) - (10 * 382.8 * 638.9)}{(2540284) - (10 * 382.8^2)}$$

$$\beta_1 = \frac{1857399}{1074926} = 1.727932$$

$$r_{x,y} = \frac{10(4303108) - (3828)(6389)}{\sqrt{\left[10(2540284) - (3828)^2\right]\left[10(7604693) - (6389)^2\right]}}$$

$$r_{x,y} = \frac{18573988}{\sqrt{[10749256][35227609]}} \qquad r_{x,y} = \frac{18573988}{19459460.1}$$

$$r_{x,y} = 0.9545$$

$$r^2 = 0.9111$$

4. We can then substitute the values in the β_0 formula

$$\beta_0 = y_{avg} - \beta_1 x_{avg}$$

$$\beta_0 = 638.9 - 1.727932 * 382.8 = -22.5525$$

5. We now find y_k from the formula $y_k = \beta_0 + \beta_1 x_k$

$$y_k = -22.5525 + 1.727932 * 386 = 644.4294$$

C Coding Standard Example (adapted from W Humphrey)

Purpose	To guide implementation of C programs
Program Headers	Begin all programs with a descriptive header.
Header Format	/**/ /* Program Assignment: the program number */ /* Name: your name */ /* Date: the date you started developing the program */ /* Description: a short description of the program and what it does */ /**/
Listing Contents	Provide a summary of the listing contents
Contents Example	/**/ /* Listing Contents: */ /* Reuse instructions */ /* Modification instructions */ /* Compilation instructions */ /**/
Reuse Instructions	- Describe how the program is used: declaration format, parameter values, types, and formats. - Provide warnings of illegal values, overflow conditions, or other conditions that could potentially result in improper operation.
Reuse Instruction Example	/**/ /* Reuse instructions */ /* int printLine(char *line_of_character) */ /* Purpose: to print string, ‘line_of_character’, on one print line */ /* Limitations: the line length must not exceed LINE_LENGTH */ /* Return 0 if printer not ready to print, else 1 */ /**/
Identifiers	Use descriptive names for all variable, function names, constants, and other identifiers. Avoid abbreviations or single-letter variables.
Identifier Example	Int number_of_students; /* This is GOOD */ Float: x4, j, ftave; /* This is BAD */
Comments	- Document the code so the reader can understand its operation. - Comments should explain both the purpose and behavior of the code. - Comment variable declarations to indicate their purpose.
Good Comment	If(record_count > limit) /* have all records been processed? */
Bad Comment	If(record_count > limit) /* check if record count exceeds limit */
Major Sections	Precede major program sections by a block comment that describes the processing done in the next section.
Example	/**/ /* The program section examines the contents of the array ‘grades’ and calcu- */ /* lates the average class grade. */ /**/
Blank Spaces	- Write programs with sufficient spacing so they do not appear crowded. - Separate every program construct with at least one space.
Indenting	- Indent each brace level from the preceding level. - Open and close braces should be on lines by themselves and aligned.

Indenting Example	while (miss_distance > threshold)
	{
	  success_code = move_robot (target _location);
	 if (success_code == MOVE_FAILED)
	 {
	   printf(“The robot move has failed.\n”);
	 }
	}

Capitalization	- Capitalize all defines. - Lowercase all other identifiers and reserved words. - To make them readable, user messages may use mixed case.
Capitalization Examples	#define DEFAULT-NUMBER-OF-STUDENTS 15 int class-size = DEFAULT-NUMBER-OF-STUDENTS;

Example C Size Counting Standard

Definition Name:	Example C LOC std.	Language:	C
Author:	(adapted from W Humphrey)	Date:	08/04/13

Count Type	**Type**	**Comments**
Physical/Logical	Logical	
Statement Type	**Included**	**Comments**
Executable	Yes	
Nonexecutable:		
Declarations	Yes,Notes 3, 4	
Compiler Directives	Yes, Note 4	
Comments	No	
Blank lines	No	
Other elements		
Clarifications		**Examples/Cases**
Empty statements	yes	“;;”, ; , etc.
Begin...end	note 1	
Expression evaluation	yes	when used as sub program arguments like void doSomething(int) and doSomething(2x+y) where x = 2 and y= 12
End symbols	notes 1,2	for terminating executable statements, declarations, bodies
Then, else	note 1	
Else if	yes	
Note 1		Count once every occurrence of the following key words: CASE, DO, ELSE, ENUM, FOR, IF, STRUCT, SWITCH, UNION, WHILE
Note 2		count once every occurrence of the following: ; , {} or };
Note 3		count each variable or parameter declaration
Note 4		count once each #define, #ifdef, #include, etc. statement