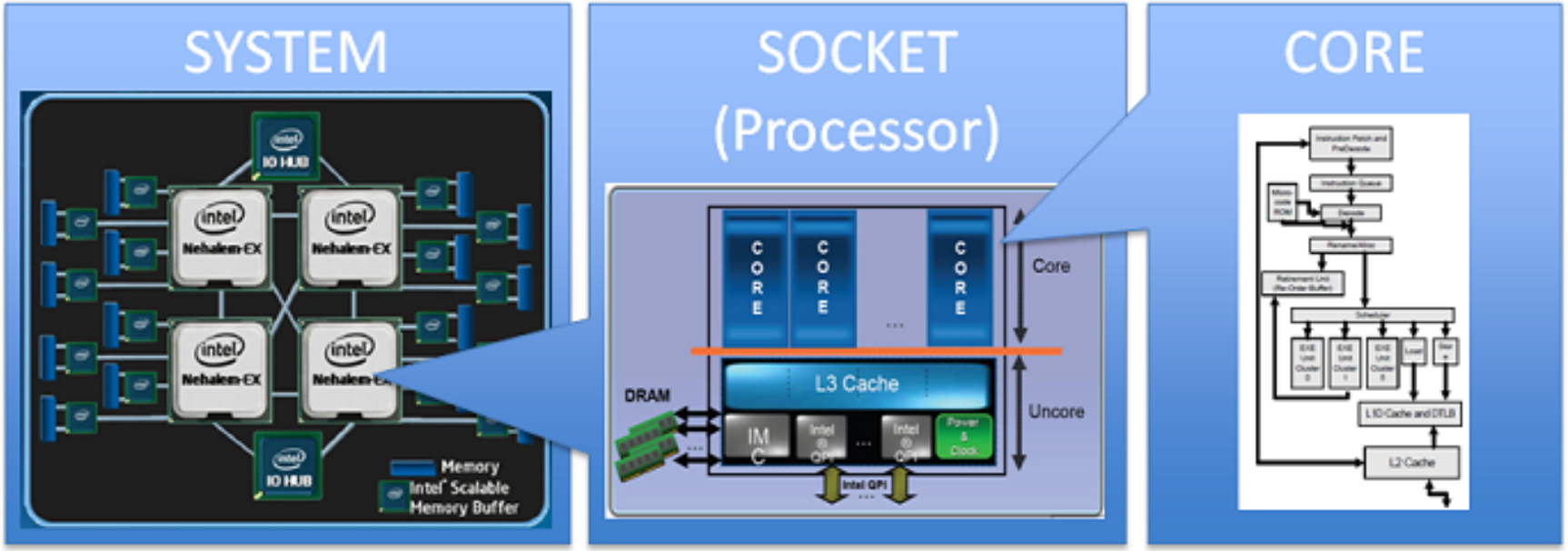
Processors and Cores



YOU WILL HAVE FUN!!!!!

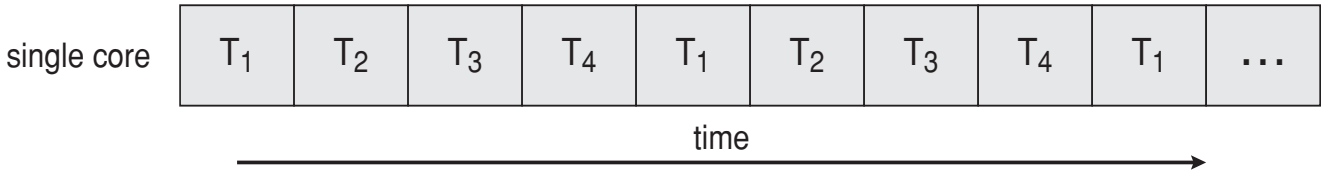
```
ycoady@linux0a:~/worm/tests$ ./test3
Task 1 will print every two seconds.
Task 1: Tick!
Task 2 will print every 1.5 seconds.
Task 2: Tock!
Task 2: Tock!
Task 1: Tick!
Task 2: Tock!
Task 2: Tock!
Task 2: Tock!
Task 2: Finished.
Segmentation fault (core dumped)
ycoady@linux0a:~/worm/tests$ ./test3
Task 1 will print every two seconds.
Task 1: Tick!
Task 2 will print every 1.5 seconds.
Task 2: Tock!
Task 2: Tock!
Task 2: Tock!
Task 2: Tock!
Task 2: Tock!
Task 2: Finished.
Segmentation fault (core dumped)
ycoady@linux0a:~/worm/tests$ ./test3
Task 1 will print every two seconds.
Task 1: Tick!
Task 2 will print every 1.5 seconds.
Task 2: Tock!
Task 2: Tock!
Task 1: Tick!
Task 2: Tock!
Task 1: Tick!
Task 2: Tock!
Task 2: Tock!
Task 2: Finished.
ycoady@linux0a:~/worm/tests$
```

Multicore Programming (Cont.)

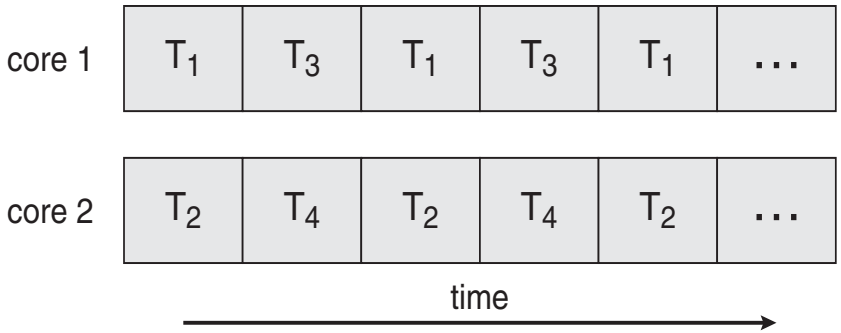
- Types of parallelism
  - **Task parallelism** – distributing threads across cores, each thread performing unique operation
    - A2 part 1 Worm!
  - **Data parallelism** – distributes subsets of the same data across multiple cores, same operation on each
    - **A2 part 2 TODAY!!!!!**

Concurrency vs. Parallelism

- **Concurrent execution on single-core system:**



- **Parallelism on a multi-core system:**



Solving L1 Problems

- Ian Barrodale (ianbarrodale@gmail.com)
- Adjunct Professor
- Computer Science Department
- University of Victoria

Given **A** and **b** where **A** is an mxn matrix, and **b** is an mx1 vector, the system of equations **Ax = b** is

(i) SQUARE if m = n (same number of rows and columns), and there is a *unique* solution **x**, provided that det(**A**) ≠ 0

(ii) UNDERDETERMINED if m < n (fewer rows than columns), and the number of solutions **x** is *infinite*

(iii) OVERDETERMINED if m > n (more rows than columns), and there are no *exact* solutions **x**.

We'll look mainly at case (iii), but first at case (ii) briefly.

Case (ii) example: Three equations in five variables

$$x_1 + 2x_2 + 3x_3 + 4x_4 + 5x_5 = 22$$

$$x_1 + 2x_2 + 2x_3 + 2x_4 + 3x_5 = 13$$

$$x_1 + x_2 + 2x_3 + 2x_4 + 2x_5 = 12$$

A selection of 7 solutions is shown

x_1	x_2	x_3	x_4	x_5	$\sum_i \lvert x_i \rvert$
-2	0	5	1	1	9
-7	1	9	0	0	17
0	0	3	2	1	6
-4	0	7	0	1	12
1	-1	3	1	2	8
-6	0	9	-1	1	17
-15	3	15	-1	-2	36

Case (ii) example: Three equations in five variables

$$x_1 + 2x_2 + 3x_3 + 4x_4 + 5x_5 = 22$$

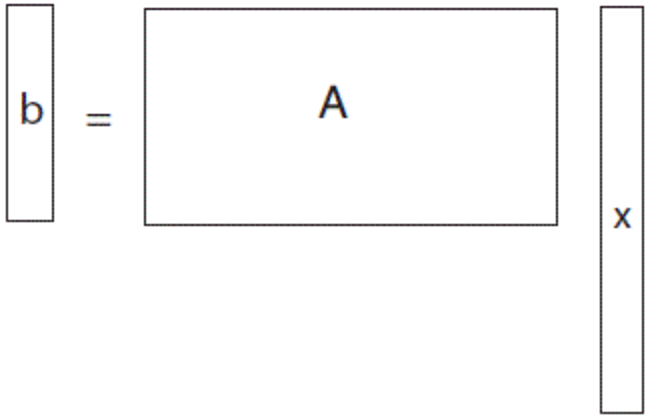
$$x_1 + 2x_2 + 2x_3 + 2x_4 + 3x_5 = 13$$

$$x_1 + x_2 + 2x_3 + 2x_4 + 2x_5 = 12$$

A selection of 7 solutions is shown with the "smallest" of *all possible solutions* indicated

x_1	x_2	x_3	x_4	x_5	$\sum_i \lvert x_i \rvert$
-2	0	5	1	1	9
-7	1	9	0	0	17
0	0	3	2	1	6 ← L1
-4	0	7	0	1	12
1	-1	3	1	2	8
-6	0	9	-1	1	17
-15	3	15	-1	-2	36

Underdetermined systems of linear equations



- $x \in \mathbb{R}^n$
- $m \ll n$ linear equations about x

$$b = Ax, \quad b \in \mathbb{R}^m$$

- Would like to recover x
- Arises in many fields of science and engineering

Linear programming formulation

$$\begin{array}{ll} \text{minimize} & \sum_i |x_i| \\ \text{subject to} & Ax = b \end{array}$$

is equivalent to

$$\begin{array}{ll} \text{minimize} & \sum_i t_i \\ \text{subject to} & Ax = b \\ & -t_i \leq x_i \leq t_i \end{array}$$

with variables $x, t \in \mathbb{R}^n$

An Application: Compressed Sensing

- **Wikipedia: Compressed sensing** is a signal processing technique for efficiently acquiring and reconstructing a signal, by finding solutions to underdetermined linear systems. This takes advantage of the signal's sparseness…allowing the entire signal to be constructed from relatively few measurements. (This is analogous to Sudoku, where that game's rules allow a player to deduce the value of every point on the board, despite knowing only a few initial samples.)

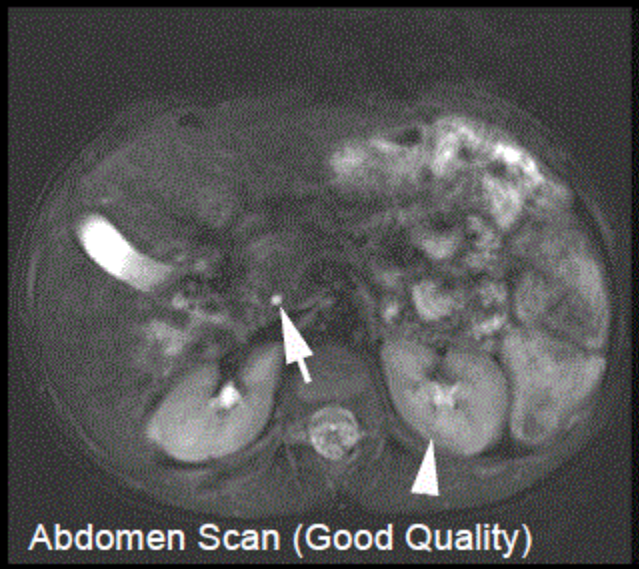
- Tomography is a prominent application, as compressed sensing could produce the same scans in less time for MRI, and with reduced radiation dose than for X-ray Computed Tomography (CT).

The Need for Speed

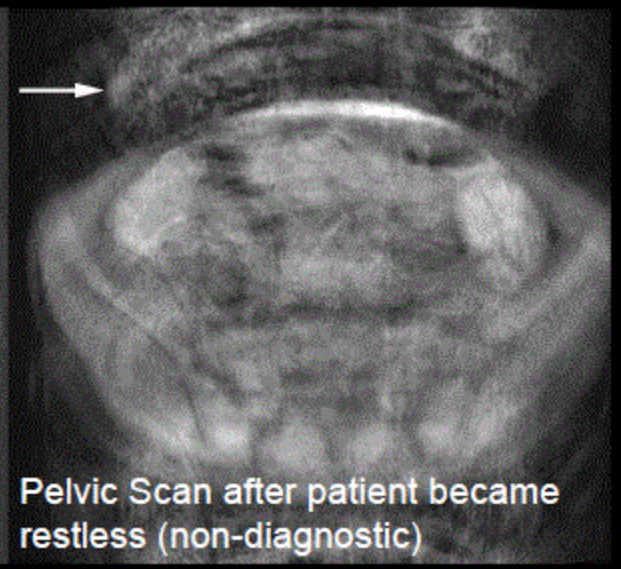
- MRI data collection is inherently slow
- Faster imaging would:
  - Decrease scan-time
  - Decrease image artifacts
  - Increase spatial/temporal resolution
  - Increase coverage
  - Enable new applications
- Possible solution:
  - Faster imaging by reducing data (exploiting redundancies)

True Need for Speed: Pediatric MRI

- Fact: Impact of MRI on children health is limited
- Challenges:
  - Children can not keep still
  - Children can not breathhold
  - Low tolerance of long exams
  - Wide range of indications, patient sizes

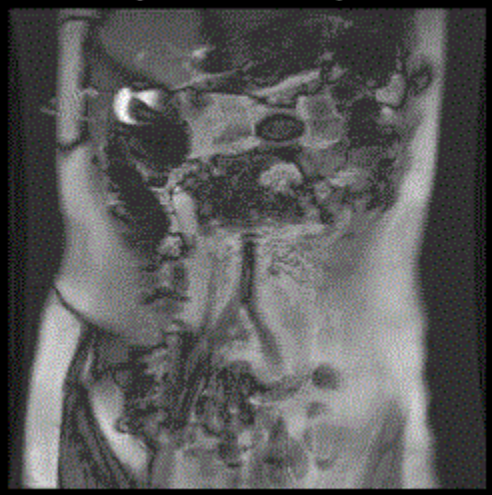


Abdomen Scan (Good Quality)



Pelvic Scan after patient became restless (non-diagnostic)

un-cooperative 8 year old



Our principal concern in this presentation is "solving" overdetermined systems of equations **Ax = b**, which have more rows than columns and no *exact* solution.

Case (iii) example: Five equations in two variables

$$x_1 + x_2 = 1$$

$$x_1 + 2x_2 = 1$$

$$x_1 + 3x_2 = 2$$

$$x_1 + 4x_2 = 3$$

$$x_1 + 5x_2 = 2$$

Since no values for x_1 and x_2 can satisfy all five equations, an *approximate solution* must suffice.

An approximate solution $\mathbf{x}^a$ to $\mathbf{Ax} = \mathbf{b}$ can be judged by the "size" of $(\mathbf{b} - \mathbf{A}\mathbf{x}^a)$, defined by $\|\mathbf{b} - \mathbf{A}\mathbf{x}^a\|$. An approximate solution is "best" if it minimizes $F(\mathbf{x}) = \|\mathbf{b} - \mathbf{A}\mathbf{x}\|$. For which norm? And can Linear Programming (LP) be applied here?

The problem at hand, for three popular norms, is to:

$$\text{Minimize } F_1(\mathbf{x}) = \sum_{i=1}^{m} \left| b_i - \sum_{j=1}^{n} x_j a_{i,j} \right|, \text{ or}$$

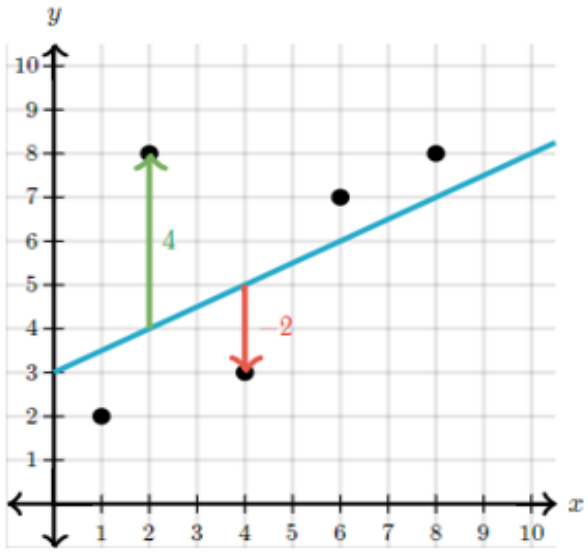
$$\text{Minimize } F_2(\mathbf{x}) = \sum_{i=1}^{m} \left| b_i - \sum_{j=1}^{n} x_j a_{i,j} \right|^2, \text{ or}$$

$$\text{Minimize } F_\infty(\mathbf{x}) = max \left| b_i - \sum_{j=1}^{n} x_j a_{i,j} \right| \text{ for } 1 \le i \le m$$

Relevant History: The first (L1) and the third (L_∞) problem above can both be solved by LP, and the second (L2) problem by calculus. Calculus was invented in the mid-17th century by Newton and Leibnitz, and the LP Simplex method by Dantzig in 1947. Boscovich proposed the L1 method for solving $\mathbf{Ax}=\mathbf{b}$ in 1753, Gauss proposed *and solved* L2 problems in 1809, and then Laplace in 1799, Fourier in 1824, and also Chebyshev (during the latter half of the 19th century), all studied the L_∞ method.

Examples are now shown of L1 fits by straight lines (n=2) to m data points and comparing them to L2 fits (via the famous *method of least-squares*.)

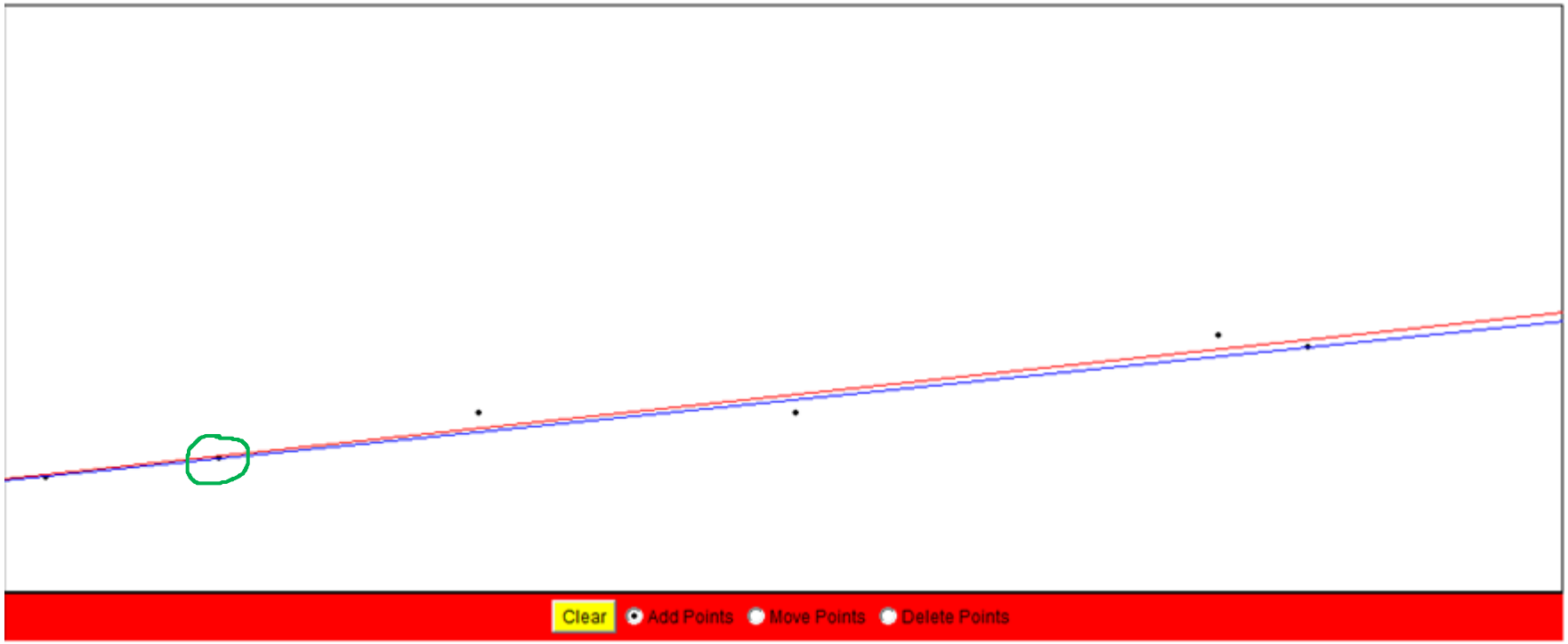
Assume the data are real (measurements or estimates) and contain errors, and that the data may also have some *outliers* (points differing significantly from most other observations). Even a single outlier can unduly influence an L_∞ fit, so this method would not be appropriate.



The vertical line segments shown here are positive (green) and negative (red) *residuals;* the closer a data point's residual is to zero, the better the fit. L1 fits are influenced by the *signs* of the residuals, and L2 fits by their *sizes* (absolute values).

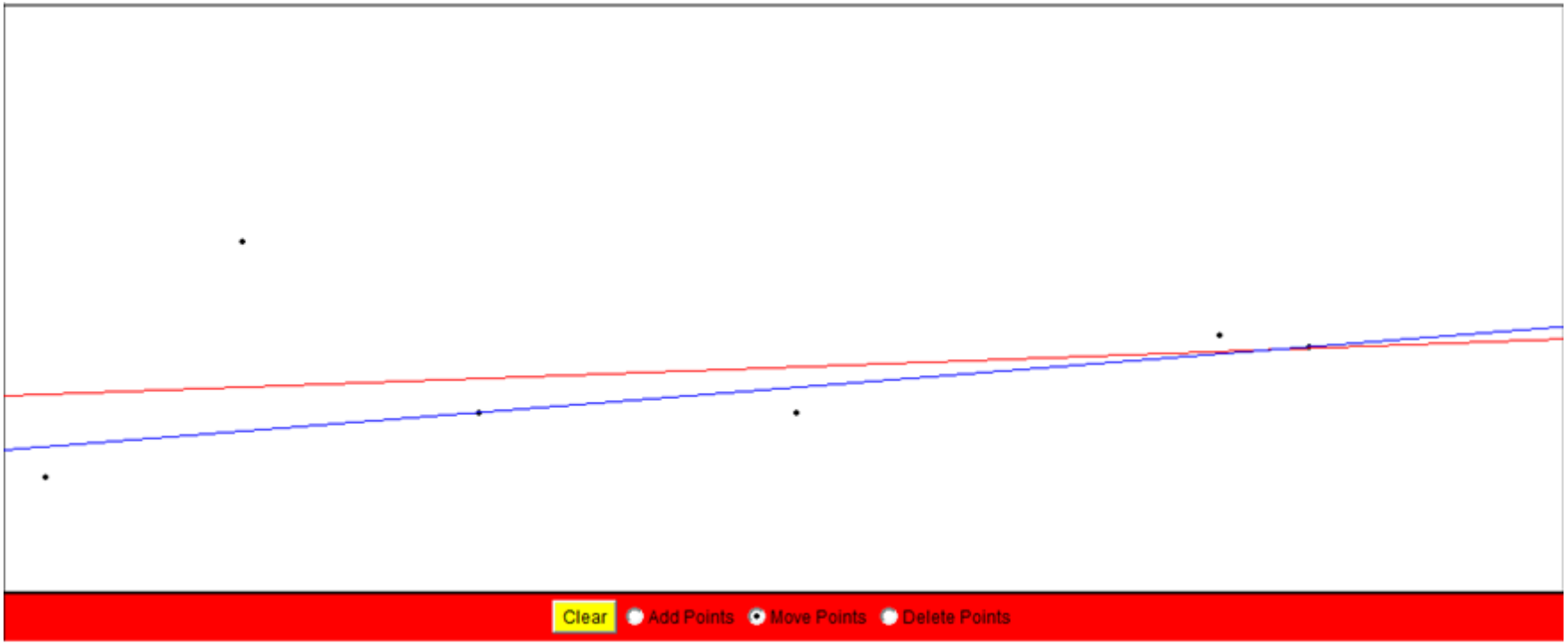
Six points, with no outliers

L1 is blue, L2 is red



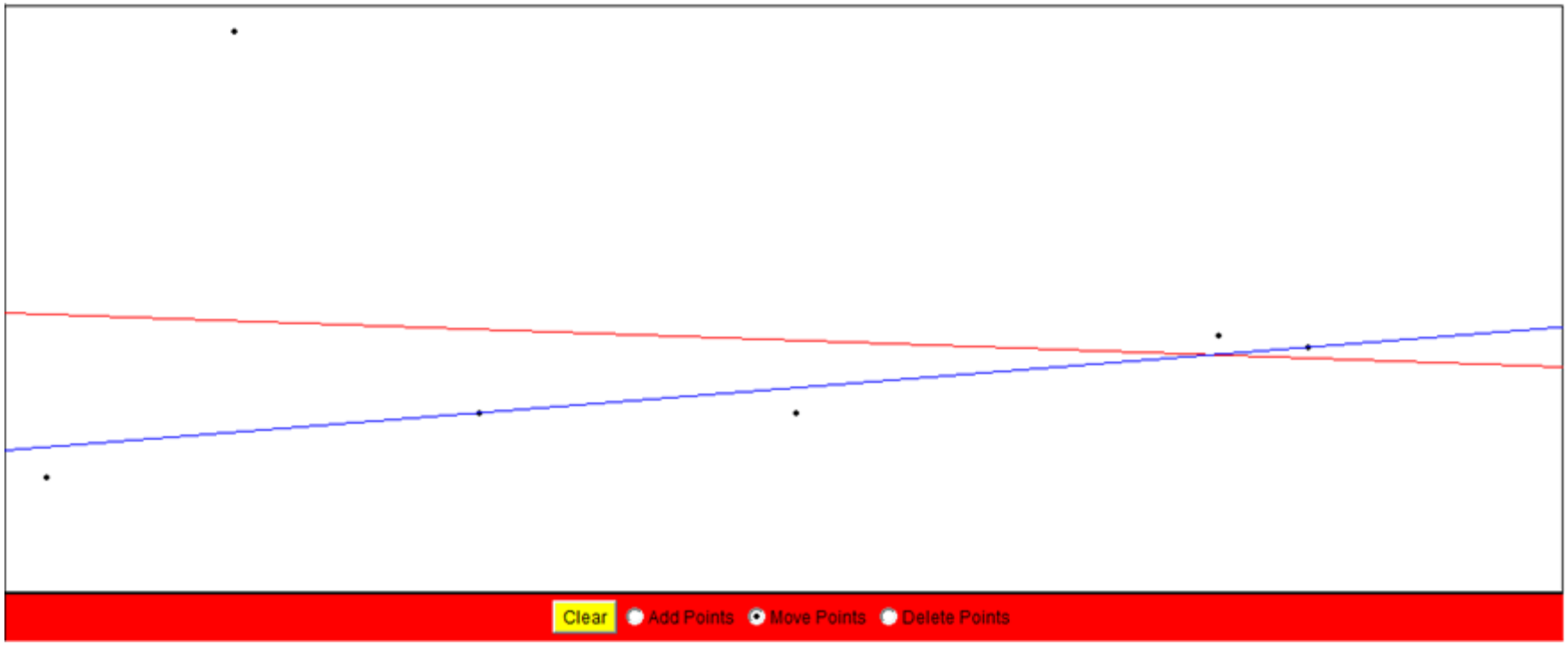
Five points preserved; one outlier

L1 is blue, L2 is red



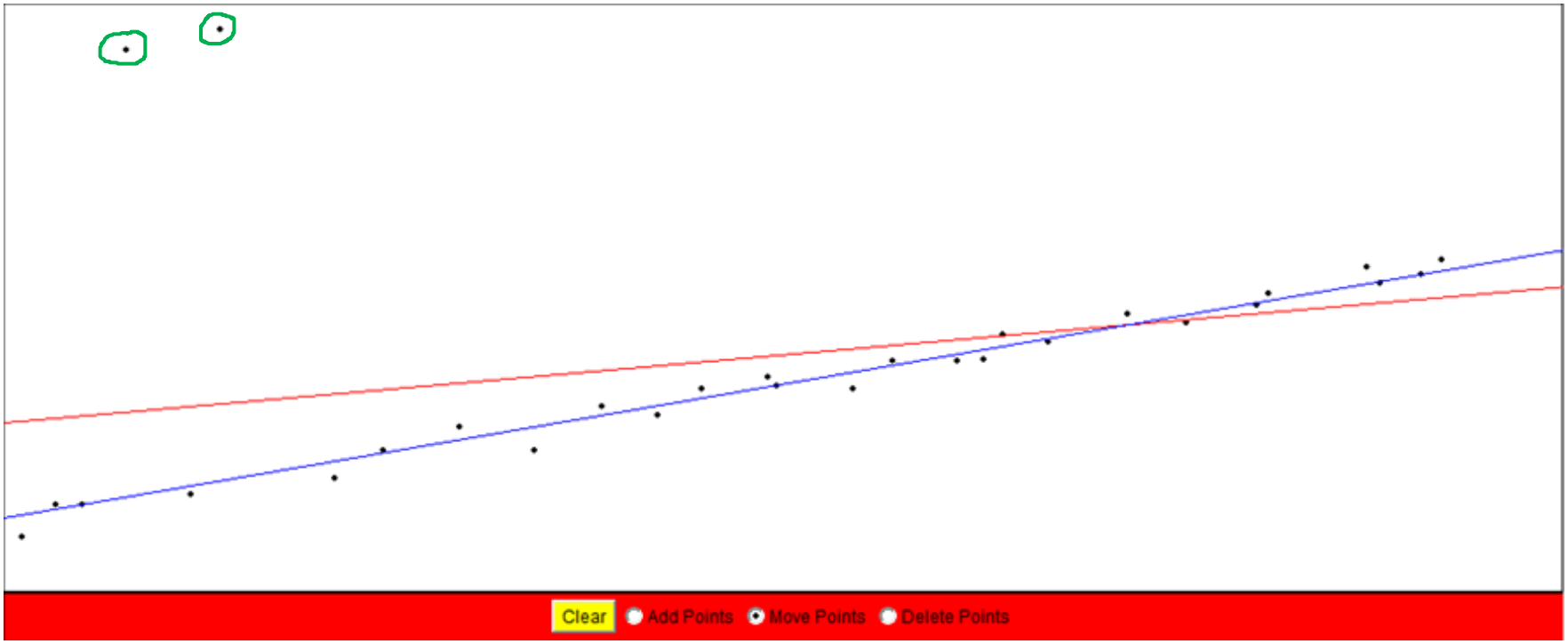
Five points preserved; one OUTLIER

L1 is blue, L2 is red



27 “good” points + two OUTLIERS

L1 is blue, L2 is red



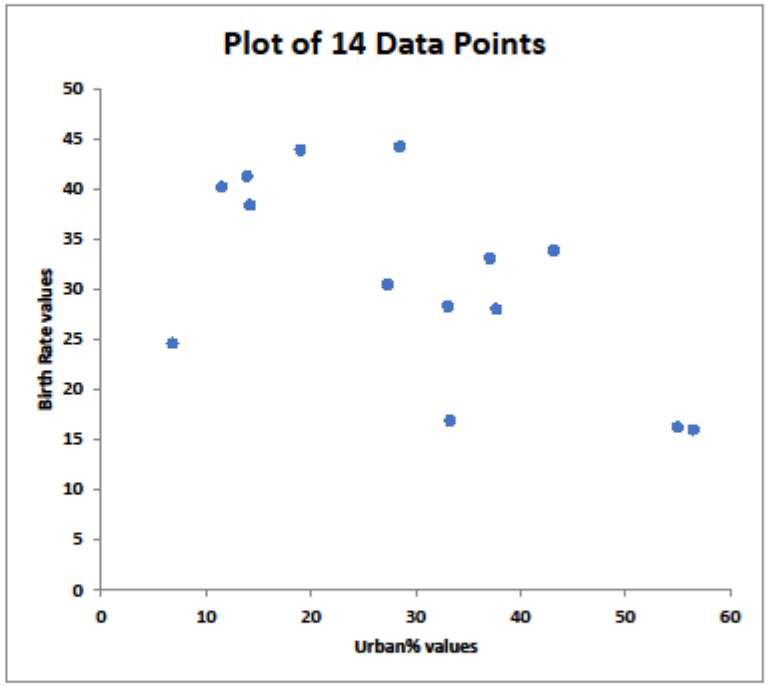
The need for an alternative to the popular fitting method of least-squares

- The data set displayed on the next slide illustrates a weakness in the popular method of least-squares. The list of data is for 14 countries in North and Central America with populations of more than one million people in 1985. For each country we show its BirthRate (the number of births per year per 1,000 people) for the period 1980-1985, and its Urban% (the percentage of the population living in cities of more than 100,000) in 1980.

1980 >100,000 Urban%	1980-1985 per1,000 BirthRate	
55.0	16.2	CAN
27.3	30.5	COS
33.3	16.9	CUB
37.1	33.1	DOM
11.5	40.2	ELS
14.2	38.4	GUA
13.9	41.3	HAI
19.0	43.9	HON
33.1	28.3	JAM
43.2	33.9	MEX
28.5	44.2	NIC
37.7	28.0	PAN
6.8	24.6	TRI
56.5	16.0	USA

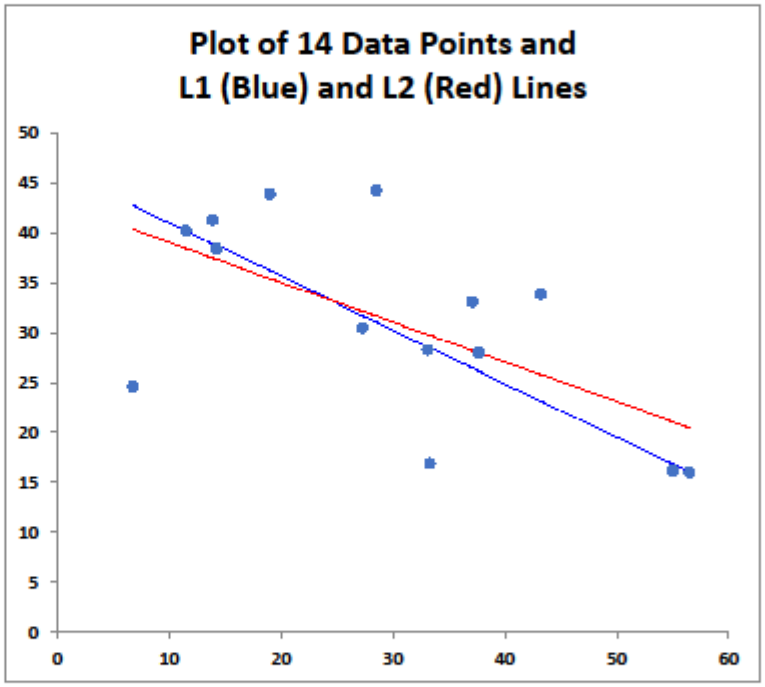
SORTED

6.8	24.6	TRI
11.5	40.2	ELS
13.9	41.3	HAI
14.2	38.4	GUA
19.0	43.9	HON
27.3	30.5	COS
28.5	44.2	NIC
33.1	28.3	JAM
33.3	16.9	CUB
37.1	33.1	DOM
37.7	28.0	PAN
43.2	33.9	MEX
55.0	16.2	CAN
56.5	16.0	USA



14 DATA POINTS			
>100,000	per1,000		L1
Urban%	BirthRate		residuals
55.0	16.2	CAN	-0.607
27.3	30.5	COS	-1.203
33.3	16.9	CUB	-11.576
37.1	33.1	DOM	6.667
11.5	40.2	ELS	0.000
14.2	38.4	GUA	-0.348
13.9	41.3	HAI	2.391
19.0	43.9	HON	7.733
33.1	28.3	JAM	-0.284
43.2	33.9	MEX	10.748
28.5	44.2	NIC	13.142
37.7	28.0	PAN	1.890
6.8	24.6	TRI	-18.128
56.5	16.0	USA	0.000
SORTED			
6.8	24.6	TRI	-18.128
11.5	40.2	ELS	0.000
13.9	41.3	HAI	2.391
14.2	38.4	GUA	-0.348
19.0	43.9	HON	7.733
27.3	30.5	COS	-1.203
28.5	44.2	NIC	13.142
33.1	28.3	JAM	-0.284
33.3	16.9	CUB	-11.576
37.1	33.1	DOM	6.667
37.7	28.0	PAN	1.890
43.2	33.9	MEX	10.748
55.0	16.2	CAN	-0.607
56.5	16.0	USA	0.000

OUTPUT	L1	L2
Intercept	46.384	42.991
Slope	-0.538	-0.399
SAR	74.71644	



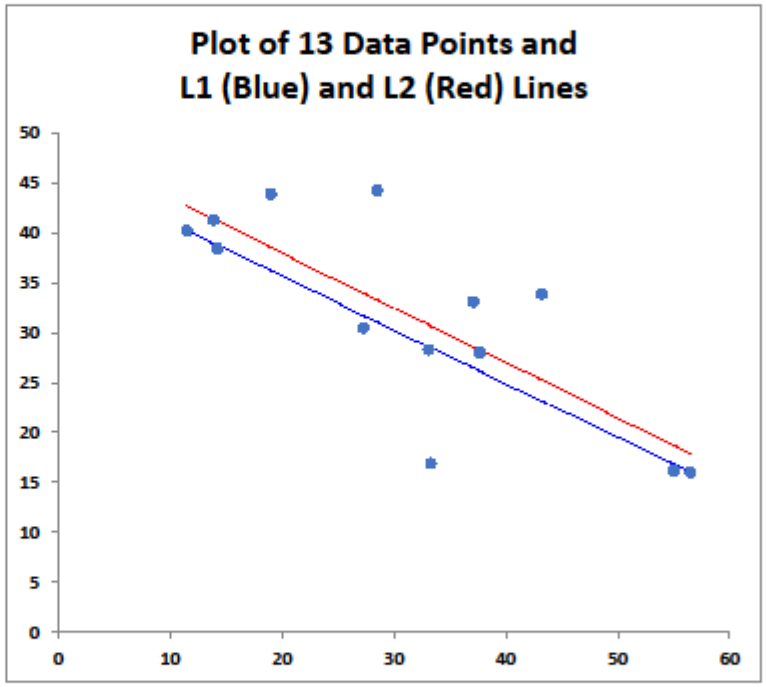
13 DATA POINTS

>100,000 Urban%	per1,000 BirthRate		L1 residuals
55.0	16.2	CAN	-0.607
27.3	30.5	COS	-1.203
33.3	16.9	CUB	-11.576
37.1	33.1	DOM	6.667
11.5	40.2	ELS	0.000
14.2	38.4	GUA	-0.348
13.9	41.3	HAI	2.391
19.0	43.9	HON	7.733
33.1	28.3	JAM	-0.284
43.2	33.9	MEX	10.748
28.5	44.2	NIC	13.142
37.7	28.0	PAN	1.890
56.5	16.0	USA	0.000

SORTED

11.5	40.2	ELS	0.000
13.9	41.3	HAI	2.391
14.2	38.4	GUA	-0.348
19.0	43.9	HON	7.733
27.3	30.5	COS	-1.203
28.5	44.2	NIC	13.142
33.1	28.3	JAM	-0.284
33.3	16.9	CUB	-11.576
37.1	33.1	DOM	6.667
37.7	28.0	PAN	1.890
43.2	33.9	MEX	10.748
55.0	16.2	CAN	-0.607
56.5	16.0	USA	0.000

OUTPUT	L1	L2
Intercept	46.384	48.943
Slope	-0.538	-0.549
SAR	56.58889	



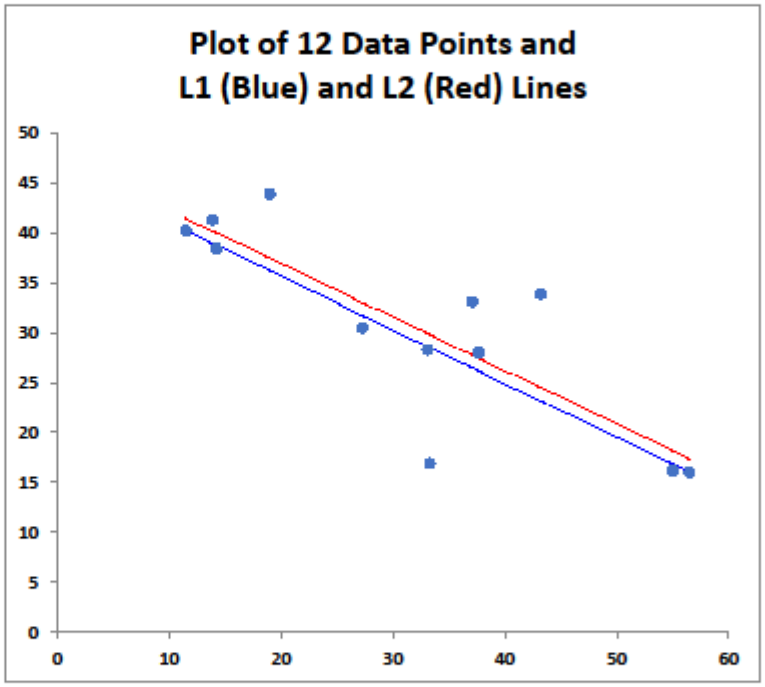
12 DATA POINTS

>100,000 Urban%	per1,000 BirthRate		L1 residuals
55.0	16.2	CAN	-0.607
27.3	30.5	COS	-1.203
33.3	16.9	CUB	-11.576
37.1	33.1	DOM	6.667
11.5	40.2	ELS	0.000
14.2	38.4	GUA	-0.348
13.9	41.3	HAI	2.391
19.0	43.9	HON	7.733
33.1	28.3	JAM	-0.284
43.2	33.9	MEX	10.748
37.7	28.0	PAN	1.890
56.5	16.0	USA	0.000

SORTED

11.5	40.2	ELS	0.000
13.9	41.3	HAI	2.391
14.2	38.4	GUA	-0.348
19.0	43.9	HON	7.733
27.3	30.5	COS	-1.203
33.1	28.3	JAM	-0.284
33.3	16.9	CUB	-11.576
37.1	33.1	DOM	6.667
37.7	28.0	PAN	1.890
43.2	33.9	MEX	10.748
55.0	16.2	CAN	-0.607
56.5	16.0	USA	0.000

OUTPUT	L1	L2
Intercept	46.384	47.586
Slope	-0.538	-0.535
SAR	43.44667	



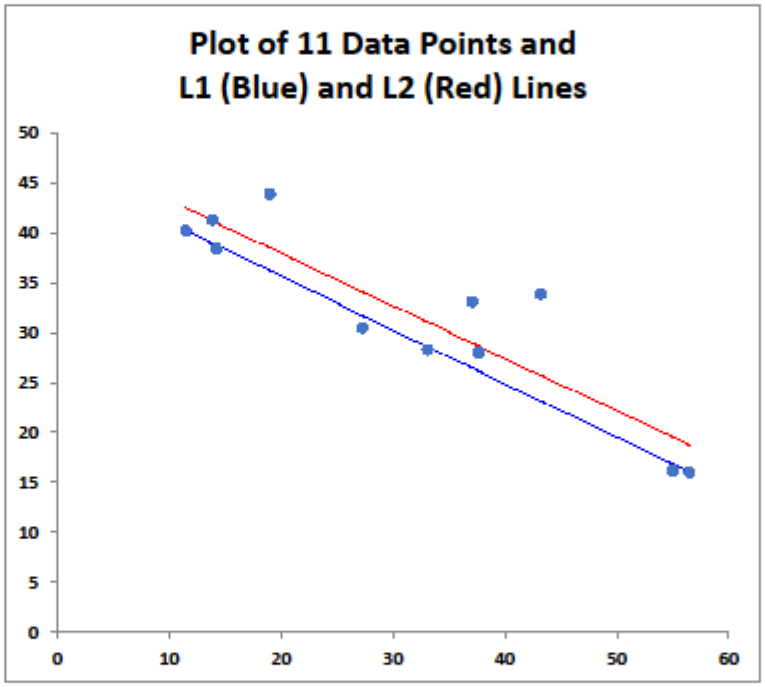
11 DATA POINTS

>100,000 Urban%	per1,000 BirthRate		L1 residuals
55.0	16.2	CAN	-0.607
27.3	30.5	COS	-1.203
37.1	33.1	DOM	6.667
11.5	40.2	ELS	0.000
14.2	38.4	GUA	-0.348
13.9	41.3	HAI	2.391
19.0	43.9	HON	7.733
33.1	28.3	JAM	-0.284
43.2	33.9	MEX	10.748
37.7	28.0	PAN	1.890
56.5	16.0	USA	0.000

SORTED

11.5	40.2	ELS	0.000
13.9	41.3	HAI	2.391
14.2	38.4	GUA	-0.348
19.0	43.9	HON	7.733
27.3	30.5	COS	-1.203
33.1	28.3	JAM	-0.284
37.1	33.1	DOM	6.667
37.7	28.0	PAN	1.890
43.2	33.9	MEX	10.748
55.0	16.2	CAN	-0.607
56.5	16.0	USA	0.000

OUTPUT	L1	L2
Intercept	46.384	48.498
Slope	-0.538	-0.527
SAR	31.87022	



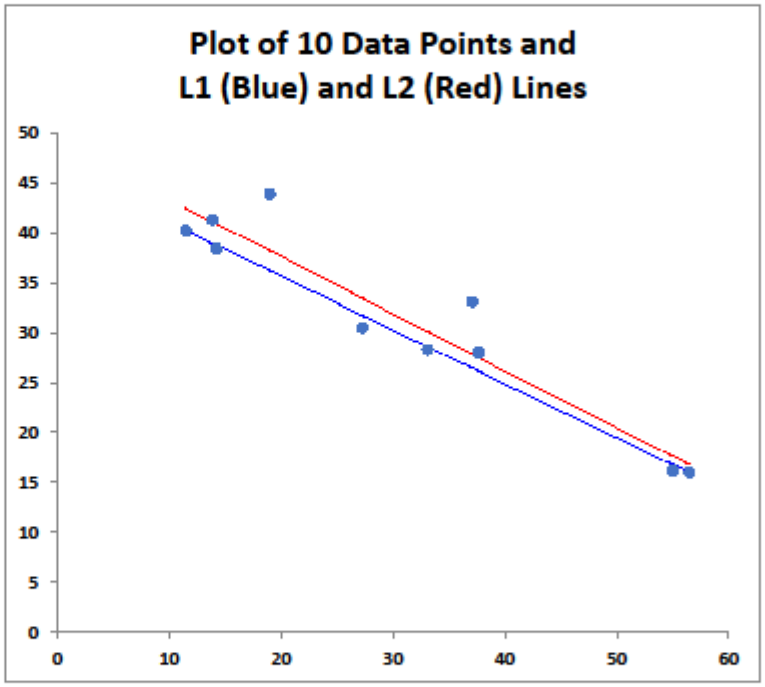
10 DATA POINTS

>100,000 Urban%	per1,000 BirthRate		L1 residuals
55.0	16.2	CAN	-0.607
27.3	30.5	COS	-1.203
37.1	33.1	DOM	6.667
11.5	40.2	ELS	0.000
14.2	38.4	GUA	-0.348
13.9	41.3	HAI	2.391
19.0	43.9	HON	7.733
33.1	28.3	JAM	-0.284
37.7	28.0	PAN	1.890
56.5	16.0	USA	0.000

SORTED

11.5	40.2	ELS	0.000
13.9	41.3	HAI	2.391
14.2	38.4	GUA	-0.348
19.0	43.9	HON	7.733
27.3	30.5	COS	-1.203
33.1	28.3	JAM	-0.284
37.1	33.1	DOM	6.667
37.7	28.0	PAN	1.890
55.0	16.2	CAN	-0.607
56.5	16.0	USA	0.000

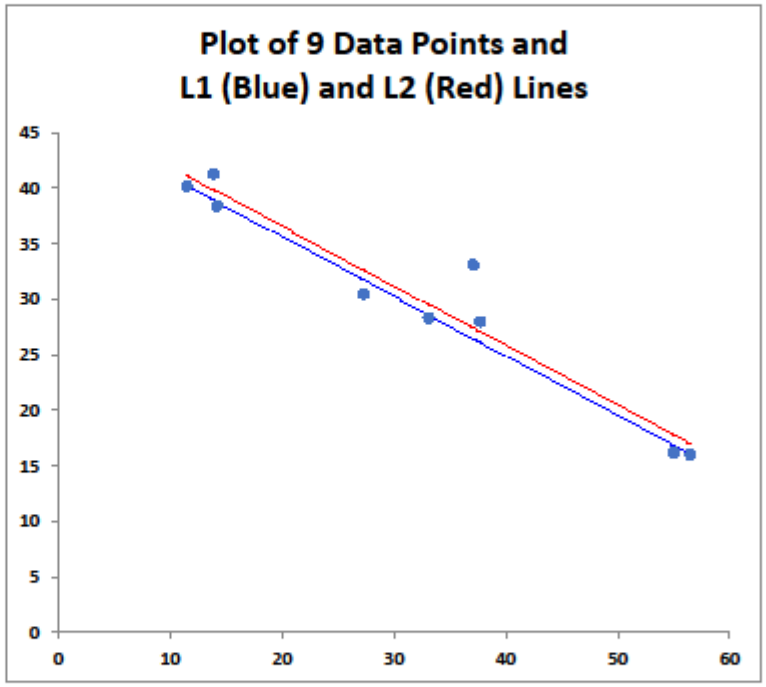
OUTPUT	L1	L2
Intercept	46.384	48.986
Slope	-0.538	-0.570
SAR	21.12267	



9 DATA POINTS

>100,000 Urban%	per1,000 BirthRate		L1 residuals
55	16.2	CAN	-0.607
27.3	30.5	COS	-1.203
37.1	33.1	DOM	6.667
11.5	40.2	ELS	0.000
14.2	38.4	GUA	-0.348
13.9	41.3	HAI	2.391
33.1	28.3	JAM	-0.284
37.7	28	PAN	1.890
56.5	16	USA	0.000

OUTPUT	L1	L2
Intercept	46.384	47.320
Slope	-0.538	-0.537
SAR	13.38933	



SORTED

11.5	40.2	ELS	0.000
13.9	41.3	HAI	2.391
14.2	38.4	GUA	-0.348
27.3	30.5	COS	-1.203
33.1	28.3	JAM	-0.284
37.1	33.1	DOM	6.667
37.7	28.0	PAN	1.890
55.0	16.2	CAN	-0.607
56.5	16.0	USA	0.000

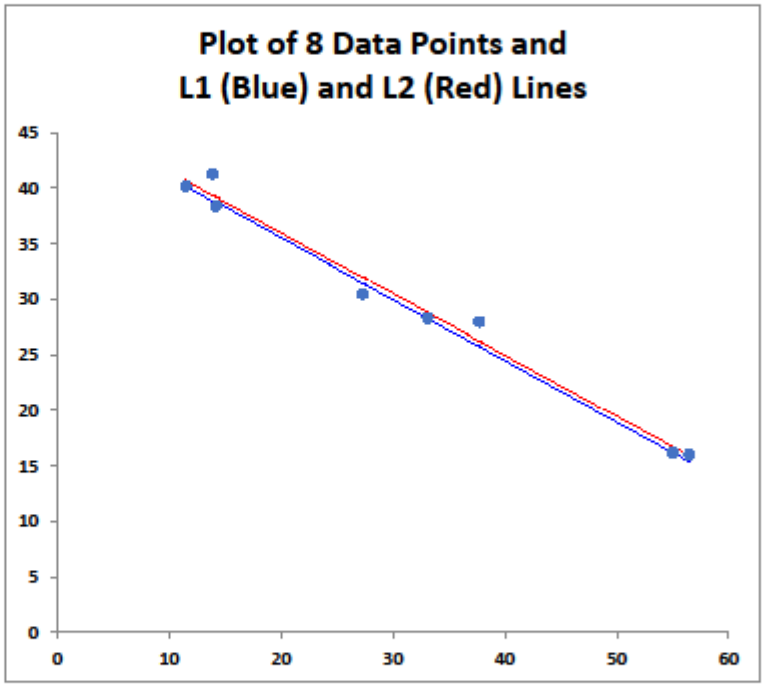
8 DATA POINTS

>100,000 Urban%	per1,000 BirthRate		L1 residuals
55.0	16.2	CAN	-0.035
27.3	30.5	COS	-0.995
11.5	40.2	ELS	0.000
14.2	38.4	GUA	-0.313
13.9	41.3	HAI	2.422
33.1	28.3	JAM	0.000
37.7	28.0	PAN	2.234
56.5	16.0	USA	0.592

SORTED

11.5	40.2	ELS	0.000
13.9	41.3	HAI	2.422
14.2	38.4	GUA	-0.313
27.3	30.5	COS	-0.995
33.1	28.3	JAM	0.000
37.7	28.0	PAN	2.234
55.0	16.2	CAN	-0.035
56.5	16.0	USA	0.592

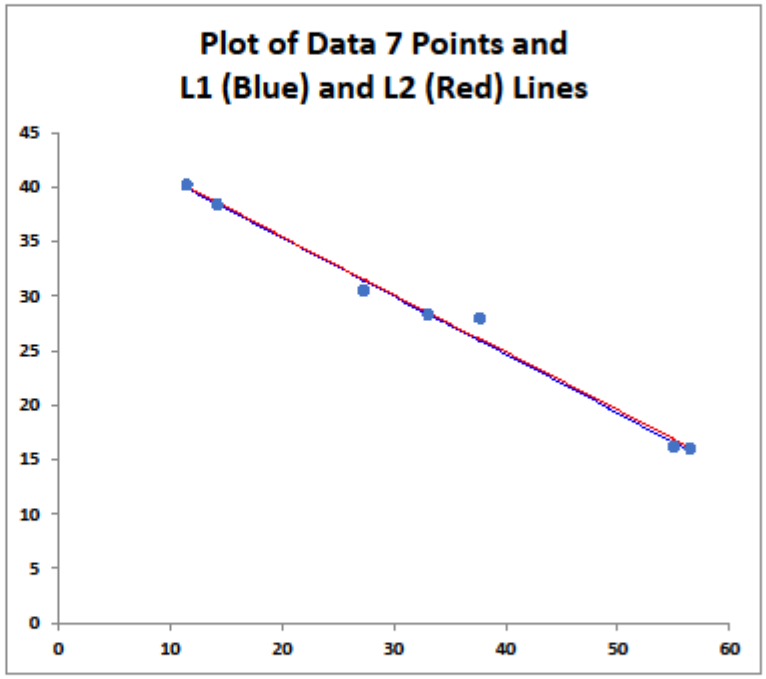
OUTPUT	L1	L2
Intercept	45.920	46.618
Slope	-0.530	-0.522
SAR	6.590741	



7 DATA POINTS

>100,000 Urban%	per1,000 BirthRate		L1 residuals
55.0	16.2	CAN	-0.397
27.3	30.5	COS	-0.899
11.5	40.2	ELS	0.357
14.2	38.4	GUA	0.000
33.1	28.3	JAM	0.000
37.7	28.0	PAN	2.158
56.5	16.0	USA	0.205

OUTPUT	L1	L2
Intercept	45.988	46.142
Slope	-0.534	-0.533
SAR	4.016402	

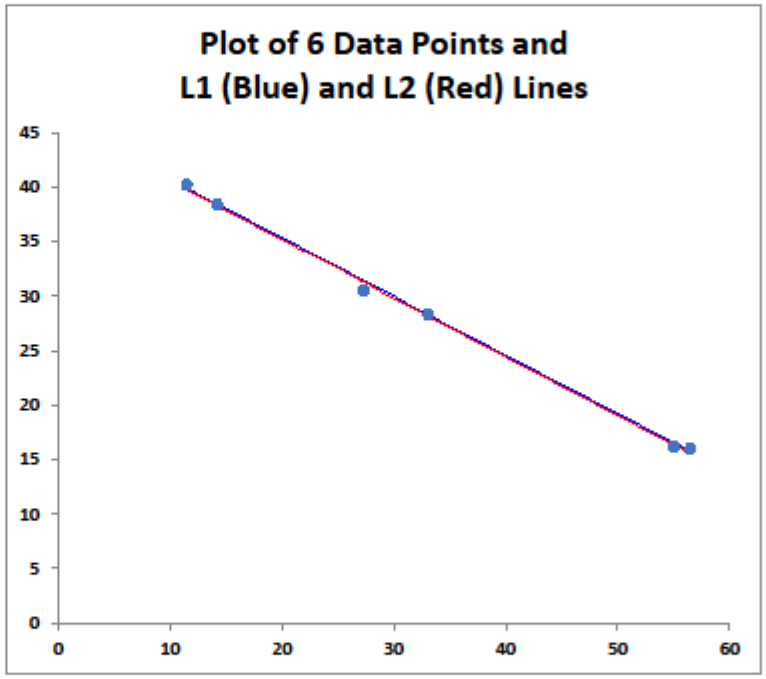


SORTED

11.5	40.2	ELS	0.000
14.2	38.4	GUA	-0.313
27.3	30.5	COS	-0.995
33.1	28.3	JAM	0.000
37.7	28.0	PAN	2.234
55.0	16.2	CAN	-0.035
56.5	16.0	USA	0.592

6 DATA POINTS			
>100,000	per1,000		L1
Urban%	BirthRate		residuals
55.0	16.2	CAN	-0.397
27.3	30.5	COS	-0.899
11.5	40.2	ELS	0.357
14.2	38.4	GUA	0.000
33.1	28.3	JAM	0.000
56.5	16.0	USA	0.205

OUTPUT	L1	L2
Intercept	45.988	45.980
Slope	-0.534	-0.538
SAR	1.858201	

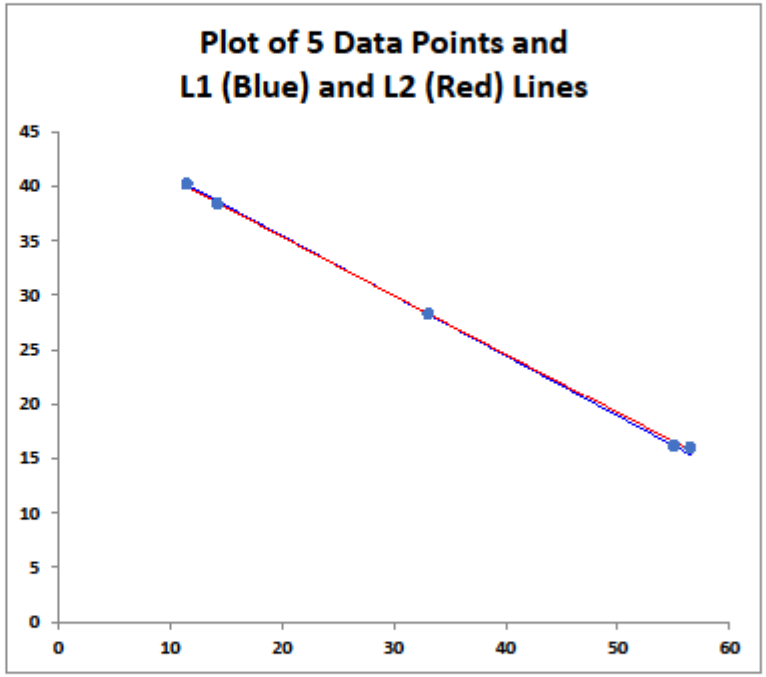


SORTED			
11.5	40.2	ELS	0.357
14.2	38.4	GUA	0.000
27.3	30.5	COS	-0.899
33.1	28.3	JAM	0.000
55.0	16.2	CAN	-0.397
56.5	16.0	USA	0.205

5 DATA POINTS

>100,000 Urban%	per1,000 BirthRate		L1 residuals
55.0	16.2	CAN	-0.035
11.5	40.2	ELS	0.000
14.2	38.4	GUA	-0.313
33.1	28.3	JAM	0.000
56.5	16.0	USA	0.592

OUTPUT	L1	L2
Intercept	46.536	46.239
Slope	-0.551	-0.541
SAR	0.9389	



SORTED

11.5	40.2	ELS	0.000
14.2	38.4	GUA	-0.313
33.1	28.3	JAM	0.000
55.0	16.2	CAN	-0.035
56.5	16.0	USA	0.592

14 DATA POINTS

OUTPUT	L1	L2
Intercept	46.384	42.991
Slope	-0.538	-0.399
SAR	74.7164	

Urban	Births	Final Results				Resids
6.8	24.6	-0.0222	-0.0222	-0.5378	*2*	-18.1276
11.5	40.2	1.2556	0.2556	46.3844	*1*	0.0000
13.9	41.3	-0.9467	0.0533	2.3907	*5*	2.3907
14.2	38.4	0.9400	-0.0600	0.3480	*-6*	-0.3480
19.0	43.9	-0.8333	0.1667	7.7333	*7*	7.7333
27.3	30.5	0.6489	-0.3511	1.2031	*-8*	-1.2031
28.5	44.2	-0.6222	0.3778	13.1422	*9*	13.1422
33.1	28.3	0.5200	-0.4800	0.2840	*-10*	-0.2840
33.3	16.9	0.5156	-0.4844	11.5764	*-11*	-11.5764
37.1	33.1	-0.4311	0.5689	6.6671	*12*	6.6671
37.7	28.0	-0.4178	0.5822	1.8898	*13*	1.8898
43.2	33.9	1.1044	0.1044	18.1276	*-3*	10.7476
55.0	16.2	0.0333	-0.9667	0.6067	*-15*	-0.6067
56.5	16.0	-0.2956	0.7044	10.7476	*14*	0.0000
		-0.7844	-0.7844	74.7164		
		4	*-16*			

13 DATA POINTS

OUTPUT	L1	L2
Intercept	46.384	48.943
Slope	-0.538	-0.549
SAR	56.5889	

Urban	Births	Final Results				Resids
11.5	40.2	-0.0222	-0.0222	-0.5378	*2*	0.0000
13.9	41.3	1.2556	0.2556	46.3844	*1*	2.3907
14.2	38.4	0.9400	-0.0600	0.3480	*-5*	-0.3480
19.0	43.9	-0.8333	0.1667	7.7333	*6*	7.7333
27.3	30.5	0.6489	-0.3511	1.2031	*-7*	-1.2031
28.5	44.2	-0.6222	0.3778	13.1422	*8*	13.1422
33.1	28.3	0.5200	-0.4800	0.2840	*-9*	-0.2840
33.3	16.9	0.5156	-0.4844	11.5764	*-10*	-11.5764
37.1	33.1	-0.4311	0.5689	6.6671	*11*	6.6671
37.7	28.0	-0.4178	0.5822	1.8898	*12*	1.8898
43.2	33.9	-0.9467	0.0533	2.3907	*4*	10.7476
55.0	16.2	0.0333	-0.9667	0.6067	*-14*	-0.6067
56.5	16.0	-0.2956	0.7044	10.7476	*13*	0.0000
		-1.8889	-0.8889	56.5889		
		3	*-15*			

12 DATA POINTS

OUTPUT	L1	L2
Intercept	46.384	47.586
Slope	-0.538	-0.535
SAR	43.4467	

Urban	Births	Final Results				Resids
11.5	40.2	-0.0222	-0.0222	-0.5378	*2*	0.0000
13.9	41.3	1.2556	0.2556	46.3844	*1*	2.3907
14.2	38.4	0.9400	-0.0600	0.3480	*-5*	-0.3480
19.0	43.9	-0.8333	0.1667	7.7333	*6*	7.7333
27.3	30.5	0.6489	-0.3511	1.2031	*-7*	-1.2031
33.1	28.3	0.5200	-0.4800	0.2840	*-8*	-0.2840
33.3	16.9	0.5156	-0.4844	11.5764	*-9*	-11.5764
37.1	33.1	-0.4311	0.5689	6.6671	*10*	6.6671
37.7	28.0	-0.4178	0.5822	1.8898	*11*	1.8898
43.2	33.9	-0.9467	0.0533	2.3907	*4*	10.7476
55.0	16.2	0.0333	-0.9667	0.6067	*-13*	-0.6067
56.5	16.0	-0.2956	0.7044	10.7476	*12*	0.0000
		-1.2667	-1.2667	43.4467		
		3	*-14*			

11 DATA POINTS

OUTPUT	L1	L2
Intercept	46.384	48.498
Slope	-0.538	-0.527
SAR	31.8702	

Urban	Births	Final Results				Resids
11.5	40.2	-0.0222	-0.0222	-0.5378	*2*	0.0000
13.9	41.3	1.2556	0.2556	46.3844	*1*	2.3907
14.2	38.4	0.9400	-0.0600	0.3480	*-5*	-0.3480
19.0	43.9	-0.8333	0.1667	7.7333	*6*	7.7333
27.3	30.5	0.6489	-0.3511	1.2031	*-7*	-1.2031
33.1	28.3	0.5200	-0.4800	0.2840	*-8*	-0.2840
37.1	33.1	-0.4311	0.5689	6.6671	*9*	6.6671
37.7	28.0	-0.4178	0.5822	1.8898	*10*	1.8898
43.2	33.9	-0.9467	0.0533	2.3907	*4*	10.7476
55.0	16.2	0.0333	-0.9667	0.6067	*-12*	-0.6067
56.5	16.0	-0.2956	0.7044	10.7476	*11*	0.0000
		-1.7822	-0.7822	31.8702		
		3	*-13*			

10 DATA POINTS

OUTPUT	L1	L2
Intercept	46.384	48.986
Slope	-0.538	-0.570
SAR	21.1227	

Urban	Births	Final Results				Resids
11.5	40.2	-0.0222	0.0222	-0.5378	*2*	0.0000
13.9	41.3	1.2556	-0.2556	46.3844	*1*	2.3907
14.2	38.4	0.9400	0.0600	0.3480	*-5*	-0.3480
19.0	43.9	-0.8333	-0.1667	7.7333	*6*	7.7333
27.3	30.5	0.6489	0.3511	1.2031	*-7*	-1.2031
33.1	28.3	-0.9467	-0.0533	2.3907	*4*	-0.2840
37.1	33.1	-0.4311	-0.5689	6.6671	*9*	6.6671
37.7	28.0	-0.4178	-0.5822	1.8898	*10*	1.8898
55.0	16.2	0.0333	0.9667	0.6067	*-11*	-0.6067
56.5	16.0	0.5200	0.4800	0.2840	*-8*	0.0000
		-1.4867	-0.5133	21.1227		
		3	*12*			

9 DATA POINTS

OUTPUT	L1	L2
Intercept	46.384	47.320
Slope	-0.538	-0.537
SAR	13.3893	

Urban	Births	Final Results				Resids
11.5	40.2	-0.0222	0.0222	-0.5378	*2*	0.0000
13.9	41.3	1.2556	-0.2556	46.3844	*1*	2.3907
14.2	38.4	0.9400	0.0600	0.3480	*-5*	-0.3480
27.3	30.5	0.6489	0.3511	1.2031	*-6*	-1.2031
33.1	28.3	-0.4178	-0.5822	1.8898	*9*	-0.2840
37.1	33.1	-0.4311	-0.5689	6.6671	*8*	6.6671
37.7	28.0	-0.9467	-0.0533	2.3907	*4*	1.8898
55.0	16.2	0.0333	0.9667	0.6067	*-10*	-0.6067
56.5	16.0	0.5200	0.4800	0.2840	*-7*	0.0000
		-0.6533	-0.3467	13.3893		
		3	*11*			

8 DATA POINTS

OUTPUT	L1	L2
Intercept	45.920	46.618
Slope	-0.530	-0.522
SAR	6.5907	

Urban	Births	Final Results				Resids
11.5	40.2	-0.0463	-0.0463	-0.5509	*2*	0.0000
13.9	41.3	1.5324	0.5324	46.5356	*1*	2.4222
14.2	38.4	0.8750	-0.1250	0.3125	*-5*	-0.3125
27.3	30.5	0.2685	-0.7315	0.9954	*-6*	-0.9954
33.1	28.3	0.2130	1.2130	2.2343	*8*	0.0000
37.7	28.0	-0.8889	0.1111	2.4222	*4*	2.2343
55.0	16.2	-1.0139	-2.0139	0.0347	*-9*	-0.0347
56.5	16.0	1.0833	2.0833	0.5917	*10*	0.5917
		-0.4630	-0.4630	6.5907		
		3	*-7*			

7 DATA POINTS

OUTPUT	L1	L2
Intercept	45.988	46.142
Slope	-0.534	-0.533
SAR	4.0164	

Urban	Births	Final Results				Resids
11.5	40.2	**0.0529**	**-0.0529**	**-0.5344**	*2*	0.3571
14.2	38.4	**-1.7513**	**0.7513**	**45.9884**	*1*	0.0000
27.3	30.5	**-0.3069**	**-0.6931**	**0.8995**	*-5*	-0.8995
33.1	28.3	**-0.2434**	**1.2434**	**2.1582**	*7*	0.0000
37.7	28.0	**1.1429**	**-0.1429**	**0.3571**	*3*	2.1582
55.0	16.2	**1.1587**	**-2.1587**	**0.3968**	*-8*	-0.3968
56.5	16.0	**-1.2381**	**2.2381**	**0.2048**	*9*	0.2048
		-0.4868	-0.5132	4.0164		
		-4	*-6*			

6 DATA POINTS

OUTPUT	L1	L2
Intercept	45.988	45.980
Slope	-0.534	-0.538
SAR	1.8582	

Urban	Births	Final Results				Resids
11.5	40.2	-0.0529	0.0529	-0.5344	*2*	0.3571
14.2	38.4	1.7513	-0.7513	45.9884	*1*	0.0000
27.3	30.5	0.3069	0.6931	0.8995	*-5*	-0.8995
33.1	28.3	-1.1587	2.1587	0.3968	*-7*	0.0000
55.0	16.2	-1.1429	0.1429	0.3571	*3*	-0.3968
56.5	16.0	1.2381	-2.2381	0.2048	*8*	0.2048
		-1.7566	-0.2434	1.8582		
		4	*6*			

5 DATA POINTS

OUTPUT	L1	L2
Intercept	46.536	46.239
Slope	-0.551	-0.541
SAR	0.9389	

Urban	Births	Final Results				Resids
11.5	40.2	-0.0463	0.0463	-0.5509	*2*	0.0000
14.2	38.4	1.5324	-0.5324	46.5356	*1*	-0.3125
33.1	28.3	-1.0139	2.0139	0.0347	*-6*	0.0000
55.0	16.2	0.8750	0.1250	0.3125	*-4*	-0.0347
56.5	16.0	1.0833	-2.0833	0.5917	*7*	0.5917
		-0.0556	-0.9444	0.9389		
		3	*5*			

Some relevant publications from the Dark Ages*

***(Fortran, Algol, and Cobol programming languages, only mainframe computers, punched cards and paper tape input, 3 or 4 runs allowed per day, output printed on paper and stacked on shelves, debugging by replacing cards or splicing inserts into paper tape, algorithms designed with flowcharts and code often written by hand before committing to cards or tape, etc.)**

Barrodale, I. and A. Young (1966) "Algorithms for best L_1 and L_∞ linear approximations on a discrete set", *Numer. Math.* 8, 295-306.

Barrodale, I. and F.D.K. Roberts (1973) "An improved algorithm for discrete ℓ_1 linear approximation", *SIAM J. Numer. Anal.* 10, 839-848. (The original version of this article appeared as Barrodale, I. and F.D.K. Roberts (1972) "An improved algorithm for discrete ℓ_1 linear approximation", Technical Summary Report 1172, Mathematics Research Center, The University of Wisconsin, Madison, Wisconsin 53706. Recd. August 17, 1971.)

Barrodale, I. and F.D.K. Roberts (1974) "Algorithm 478 – Solution of an overdetermined system of equations in the ℓ_1 norm [F4]", *Comm. ACM* 17, 319-320.

Barrodale, I. and C. Phillips (1974) "An improved algorithm for discrete Chebyshev linear approximation", in *Proc. Fourth Manitoba Conference on Numerical Math.*, 177-190.

Barrodale, I. and C. Phillips (1975) "Algorithm 495 – Solution of an overdetermined system of linear equations in the Chebyshev norm [F4]", *ACM TOMS* 1, 264-270.

A Programming Assignment

A best L1 straight line fit to m data points (t_i, d_i) can be found by minimizing

$$\sum_{i=1}^{m} \left| b_i - \sum_{j=1}^{n} x_j a_{i,j} \right|$$

where n = 2, and then for i = 1,2,…,m, put $b_i = d_i$, $a_{i,1} = 1$, and $a_{i,2} = t_i$.

Using just the known theorem that at least one best L1 line must interpolate two of the data points, write a program to calculate a best L1 line fit to the first 6, 10, 14, and 18 data points shown below, which represent the annual Consumer Price Index (CPI) for Canada, starting at 1995 (t = 1). For t = 18, note that there are 153 possible pairs to check (or can this be streamlined?).

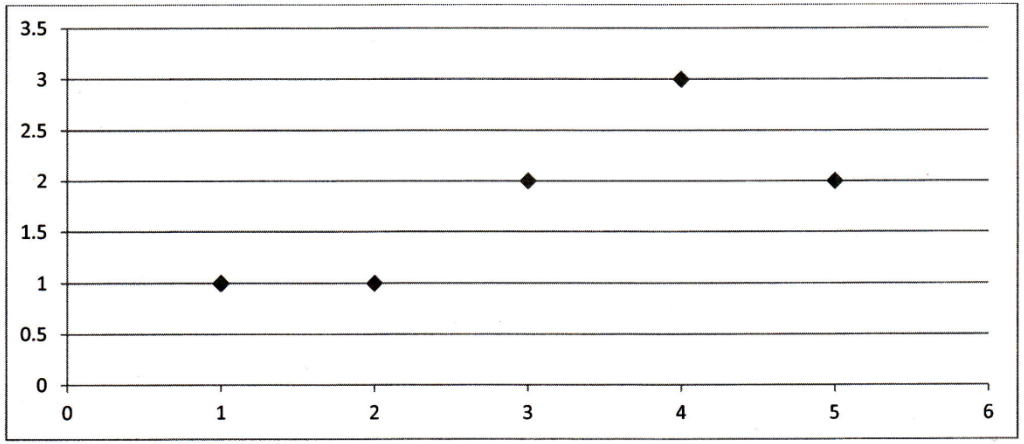
You can check your calculated values for the minimum Sum of Absolute Residuals (SAR) with those provided in the first column of the lower Figure on page 8 of my report available at http://hdl.handle.net/1828/11460 (or by navigating to *UvicSpace* on the web, and entering into its search bar, with quotes, "Fits to Data (Part 1)". To complete this assignment, please provide your computed values for the line's intercept and slope for each of the four cases (6,10,14,18 data points).

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
87.6	88.9	90.4	91.3	92.9	95.4	97.8	100	102.8	104.7	107	109.1	111.5	114.1	114.4	116.5	119.9	121.7

EXTRA SLIDES IF TIME PERMITS

Fit the straight line $a_1 + a_2 t$ to the data:

t	1	2	3	4	5
f(t)	1	1	2	3	2



This curve-fitting problem can be posed as an over-determined system of linear equations:

$$a_1 + a_2 = 1$$
$$a_1 + 2a_2 = 1$$
$$a_1 + 3a_2 = 2$$
$$a_1 + 4a_2 = 3$$
$$a_1 + 5a_2 = 2$$

$$\text{Minimize} \quad \sum_{i=1}^{m} (u_i + v_i)$$

$$\text{subject to} \quad f_i = \sum_{j=1}^{n} (b_j - c_j)\varphi_{j,i} + u_i - v_i, \quad i = 1, 2, \cdots, m,$$

$$\text{and} \quad b_j, c_j, u_i, v_i \geqq 0.$$

TABLE 2

Full initial simplex tableau for the worked example

Costs ↓	→ Basis	R	0 b_1	0 b_2	0 c_1	0 c_2	1 u_1	1 u_2	1 u_3	1 u_4	1 u_5	1 v_1	1 v_2	1 v_3	1 v_4	1 v_5
1	u_1	1	1	1	−1	−1	1					−1				
1	u_2	1	1	2**	−1	−2		1					−1			
1	u_3	2	1	3***	−1	−3			1					−1		
1	u_4	3	1	4	−1	−4				1					−1	
1	u_5	2	1	5*	−1	−5					1					−1
Marginal costs →		9	5	15	−5	−15	0	0	0	0	0	−2	−2	−2	−2	−2

TABLE 3

Solution to the worked example using condensed tableaux

Basis	R	b_1	b_2
u_1	1	1	1
u_2	1	1	2**
u_3	2	1	3***
u_4	3	1	4
u_5	2	1	5*
Marginal costs	9	5	15

Basis	R	b_1	u_3
u_1	1/3	2/3*	−1/3
v_2	1/3	−1/3	2/3
b_2	2/3	1/3	1/3
u_4	1/3	−1/3	−4/3
v_5	4/3	2/3	5/3
Marginal costs	7/3	2/3	−1/3

Basis	R	u_1	u_3
b_1	1/2	3/2	−1/2
v_2	1/2	1/2	1/2
b_2	1/2	−1/2	1/2
u_4	1/2	1/2	−3/2
v_5	1	−1	2
Marginal costs	2	−1	0

Historical note

- In 1809 Carl Friedrich Gauss published his major book "Theory of the Motion of the Heavenly Bodies Moving about the Sun in Conic Sections“; Reprinted by Dover Publications Inc; New York, 1963.
- Such an orbit is defined by n=6 parameters, which are connected with the astronomical data (consisting of measurements of angles and of time) by highly nonlinear equations. Gauss used m observations, where m is much larger than 6. From initial estimates of the 6 parameters he linearized the m equations, to get an overdetermined system of m linear equations in 6 unknowns. Gauss invented the method of least squares which he used to compute the most probable estimate of the solution of the m linearized equations, and that process was iterated until the 6 parameters had effectively converged.

By introducing n-dimensional nonnegative vectors $\underset{\sim}{x}'$ and $\underset{\sim}{x}''$ satisfying $\underset{\sim}{x} = \underset{\sim}{x}' - \underset{\sim}{x}''$, problem (1) can be written as

$$\min_{\underset{\sim}{x}', \underset{\sim}{x}''} \left[\sum_{i=1}^{n} | x_i' - x_i'' | \;\middle|\; \underset{\sim}{A} (\underset{\sim}{x}' - \underset{\sim}{x}'') = \underset{\sim}{b} \right]. \tag{2}$$

The solution to (2), and hence (1), can be obtained by applying the simplex method to the linear programming problem

$$\min_{\underset{\sim}{x}', \underset{\sim}{x}''} \left[\sum_{i=1}^{n} (x_i' + x_i'') \mid \underset{\sim}{A} (\underset{\sim}{x}' - \underset{\sim}{x}'') = \underset{\sim}{b}, \; \underset{\sim}{x}' \geq \underset{\sim}{0}, \; \underset{\sim}{x}'' \geq \underset{\sim}{0} \right] \tag{3}$$

since the simplex method automatically ensures that $x_i' x_i'' = 0$ for $i = 1,2,\ldots,n$.

A Programming Assignment

A best L1 straight line fit to m data points (t_i, d_i) can be found by minimizing

$$\sum_{i=1}^{m} \left| b_i - \sum_{j=1}^{n} x_j a_{i,j} \right|$$

where n = 2, and then for i = 1,2,…,m, put $b_i = d_i$, $a_{i,1} = 1$, and $a_{i,2} = t_i$.

Using just the known theorem that at least one best L1 line must interpolate two of the data points, write a program to calculate a best L1 line fit to the first 6, 10, 14, and 18 data points shown below, which represent the annual Consumer Price Index (CPI) for Canada, starting at 1995 (t = 1). For t = 18, note that there are 153 possible pairs to check (or can this be streamlined?).

You can check your calculated values for the minimum Sum of Absolute Residuals (SAR) with those provided in the first column of the lower Figure on page 8 of my report available at http://hdl.handle.net/1828/11460 (or by navigating to *UvicSpace* on the web, and entering into its search bar, with quotes, "Fits to Data (Part 1)". To complete this assignment, please provide your computed values for the line's intercept and slope for each of the four cases (6,10,14,18 data points).

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
87.6	88.9	90.4	91.3	92.9	95.4	97.8	100	102.8	104.7	107	109.1	111.5	114.1	114.4	116.5	119.9	121.7

A Programming Assignment

A best L1 straight line fit to m data points (t_i, d_i) can be found by minimizing

$$\sum_{i=1}^{m} \left| b_i - \sum_{j=1}^{n} x_j a_{i,j} \right|$$

where n = 2, and for i = 1,2,…,m, we put $b_i = d_i$, $a_{i,1} = 1$, and $a_{i,2} = t_i$.

Using just the known theorem that at least one best L1 line must interpolate two of the data points, write a program to calculate a best L1 line fit to the first 6, 10, 14, and 18 data points shown below, which represent the annual Consumer Price Index (CPI) for Canada, starting at 1995 (t = 1). For t = 18, note that there are 153 possible pairs to check (or can this be streamlined?).

You can check your calculated values for the minimum Sum of Absolute Residuals (SAR) with those provided in the first column of the lower Figure on page 8 of my report available at http://hdl.handle.net/1828/11460 (or by navigating to *UvicSpace* on the web, and entering into its search bar, with quotes, "Fits to Data (Part 1)". To complete this assignment, please provide your computed values for the line's intercept and slope for each of the four cases (6,10,14,18 data points).

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
87.6	88.9	90.4	91.3	92.9	95.4	97.8	100	102.8	104.7	107	109.1	111.5	114.1	114.4	116.5	119.9	121.7

(BATTLE)WORM!

- In this part of the assignment, you will implement a simple system for running a series of tasks within one process.
  - Tasks in this system will run separate functions, and your scheduler will need to switch between them.
- These tasks will have the ability to perform *blocking* operations. When a running task blocks, your scheduler will need to **stop** running the current task and select a new one to run in the meantime.
- You will not use preemption; your scheduler only switches tasks when the currently-running task blocks. This is sometimes referred to as *cooperative scheduling*, since it relies on all the tasks to cooperate by blocking periodically.
- While the system you implement could be used for other purposes, you will be implementing it specifically to support the game **Worm!**, a clone of the classic game *Snake*.
- NOT AT ALL LIKE BATTLESNAKE… but a start!
- DOES NOT USE PTHREADS!!!!!!!

Details

- This game implementation uses several tasks:
  - A main task that starts up the game
  - A task to update the worm state in the game board
  - A task that redraws the board periodically
  - A task that reads input from the user and processes controls
  - A task that generates "apples" at random locations on the board
  - A task that updates existing "apples" by spinning them and removing them after some time
- Each task runs in a loop that contains a blocking operation.
- Your job is to implement a **scheduling system** that will run these tasks in ***round-robin*** fashion, switching between them as the currently-executing task blocks.
- The starter code has been provided, and take time to look it over as soon as you can!!!

Questions & Answers

- How should task_readchar work?
  - If there's already a character to read, we could return it to the task immediately. However, if *getch()* returns ERR, no input is available.
  - First, you need to record why the task is blocked (it's waiting for input), and then invoke your scheduler to choose a new task to run.
  - Later, when the scheduler is choosing a task to run (at some point in the future) there will be an input character. The scheduler can then *swapcontext()* back to the task we blocked.

Scheduler System Details

- You will be implementing what is known as *cooperative scheduling*.
  - This is a simple technique for implementing a scheduler without preemption. Once a task is started it will run until it issues a blocking operation or exits.
  - In our case, tasks will run until they exit, wait for another task, sleep for a fixed amount of time, or wait for user input. When you hit one of these points you should invoke the scheduler function to select and start another task to run.
- As part of this process, you will need to check to see if any previously-blocked tasks can now unblock.
  - This could happen because they are
    - waiting for tasks that have now completed,
    - their sleep timer has elapsed,
    - or user input is now available.

BATTLEWORM!

- **Where in the scheduler should tasks actually be executed?**
  Once your scheduler() function has selected a task to run, you call *swapcontext* to switch to that task. When the blocked task resumes, it will look like swapcontext() just returned zero.

- **What are task_exit and the exit context for?**
  The exit context tells the processor where to go when the task's main function returns. The starter code set this up so your tasks will call task_exit when they finish. You should mark the task as DONE and then invoke the scheduler here.

- **How do we even START to test this?**
  Look in the tests folder! Tests start small and build…

BATTLEWORM!

- **When should the scheduler run?**
  When a task calls some function that blocks that task, invoke the scheduler. The scheduler should loop over tasks until it **finds one** that can run. Once you find the task, *swapcontext* to it. At this point, the scheduler is done.

- **Do you have any hints for task_readchar?**
  If a task blocks waiting for user input, the scheduler will only know if the task can resume if it calls getch(). If getch() returns ERR to the scheduler, the task can't run. If it returns something else, then you can run the task. Whatever value was returned by getch() should then be returned by task_readchar().

BATTLEWORM!

- **Is a context a list of instructions we need to run?**
  Yes, (well, sorta!) in combination with what is needed for accessing the program source and memory. You can think of contexts as a specific spot we can return to in the middle of a set of steps, a little like a PCB only smaller, and the tasks will share the same memory!

- **A hint:**
  If you are switching to a task that is the same as the current task, just return from the scheduler instead of calling swapcontext().