Lab 3

CS-335a

Fall 2012
Computer Science Department

Manolis Surligas
surligas@csd.uoc.gr

Summary

- Correlation and dependence
- Guidelines for performing network experiments
- Setup of long running experiments
- Data parsing
- Plot parsed data

Correlation and dependence

$$R_{yx}(m) = \frac{1}{N} \sum_{n=1}^{N-m+1} y(n)x(n+m-1)$$

m=1,2,...,N+1

- Assume that we have two data sets X, Y
- The cross-correlation refers to the amount of the statistical relation between X and Y
- In other words: Describes the dependence of the values of X from Y and vice versa
- If X and Y are independent, the cross-correlation is equals 0 (assuming that they have zero mean)
- If x(t) == y(t) then R_{xx} is the auto-correlation

Correlation and dependence

- Matlab provides a built-in function for the correlation
  - c = xcorr(A) is the auto-correlation of vector A
  - c = xcorr(A, B) is the cross-correlation of A and B
- The return value c is a vector containing the values of the auto or cross-correlation with length(c) = 2*m-1
- So $R_{xy}(0)$ is the middle of the vector c
- When we plot the correlation it is very useful to use the '*coeff*' parameter that normalizes all the values of c so the Rxx(0) = 0
  - c = xcorr(A, B, 'coeff')

Correlation and dependence

- Lets see some examples!

```
1  -    x = randn(100, 1);
2       % Put whatever dependency of x you want and see
3       % how correlation changes
4  -    y = 3*x;
5  -    z = 4 + x.^3;
6  -    c1 = xcorr(x, y, 'coeff');
7  -    c2 = xcorr(x, z, 'coeff');
8  -    figure(1);
9  -    x_axis = -99:99;
10 -    plot(x_axis, c1, x_axis, c2, 'r');
11 -    legend('y = 3*x', 'z = 4 + x^3');
12
```

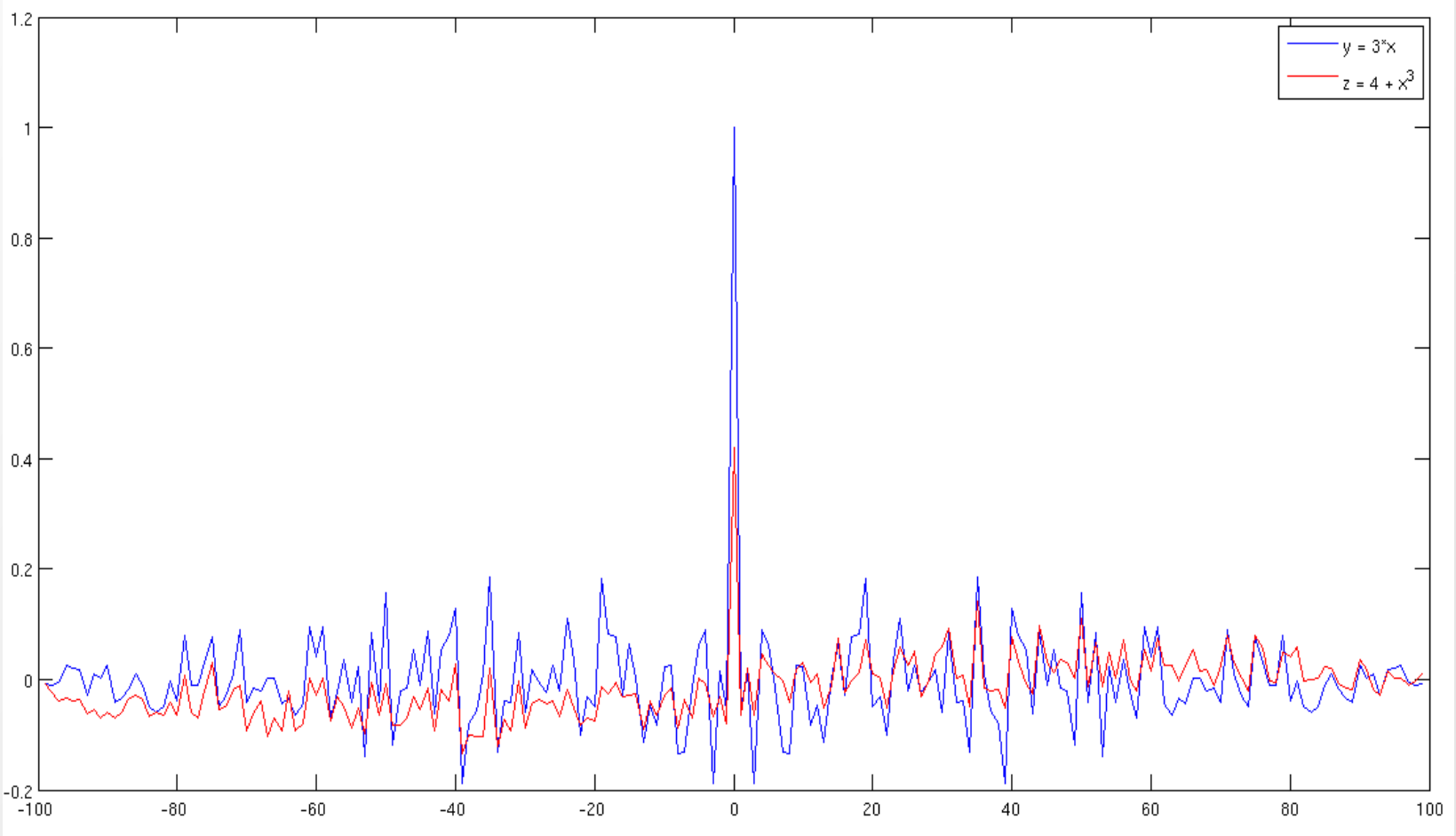
Correlation and dependence

- Lets see some examples!

```
1  -    x = randn(100, 1);
2       % Put whatever dependency of x you want and see
3       % how correlation changes
4  -    y = 3*x;
5  -    z = 4 + x.^3;
6  -    c1 = xcorr(x, y, 'coeff');
7  -    c2 = xcorr(x, z, 'coeff');
8  -    figure(1);
9  -    x_axis = -99:99;
10 -    plot(x_axis, c1, x_axis, c2, 'r');
11 -    legend('y = 3*x', 'z = 4 + x^3');
12
```

Correlation and dependence

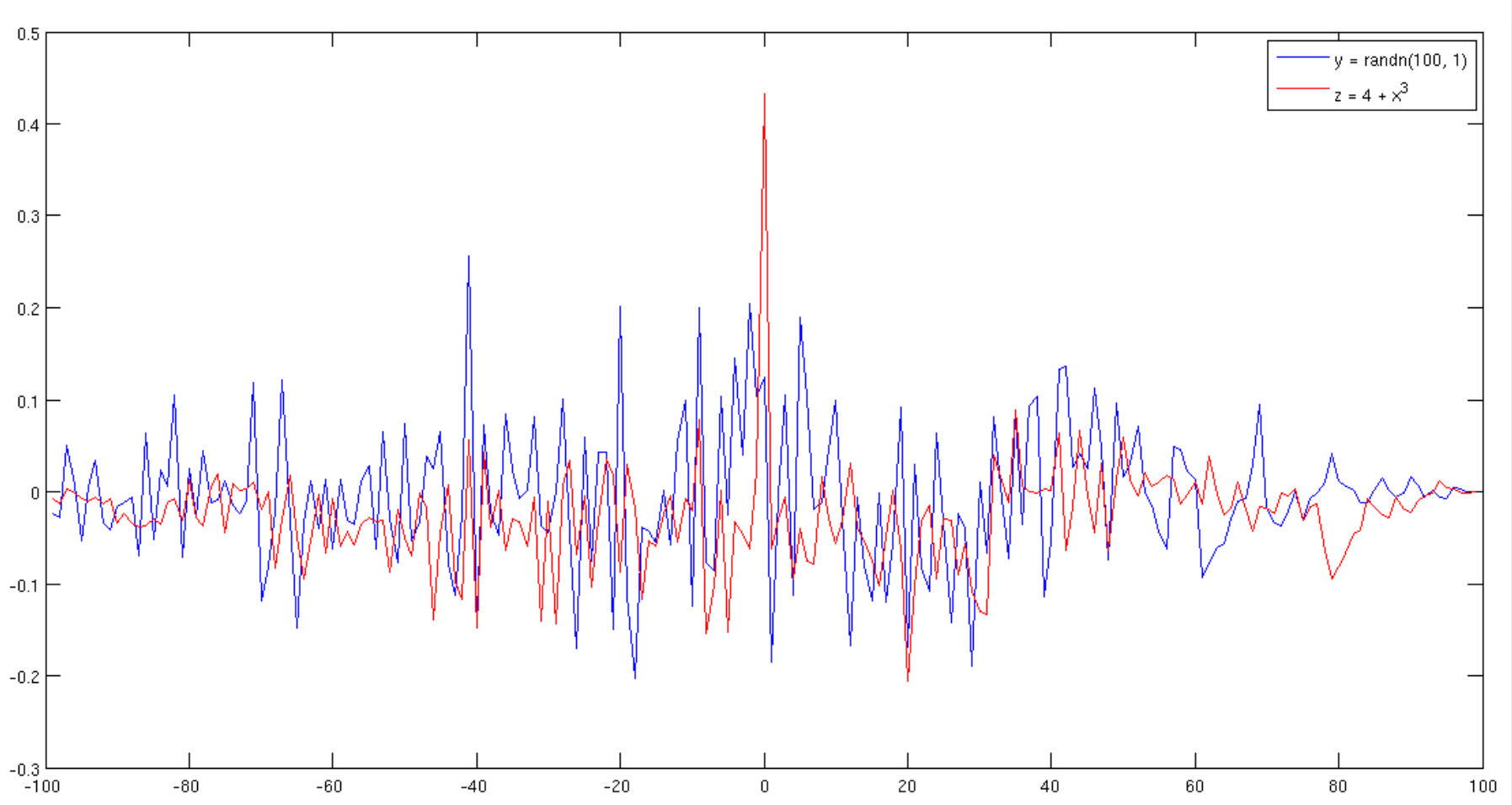
- And the result...



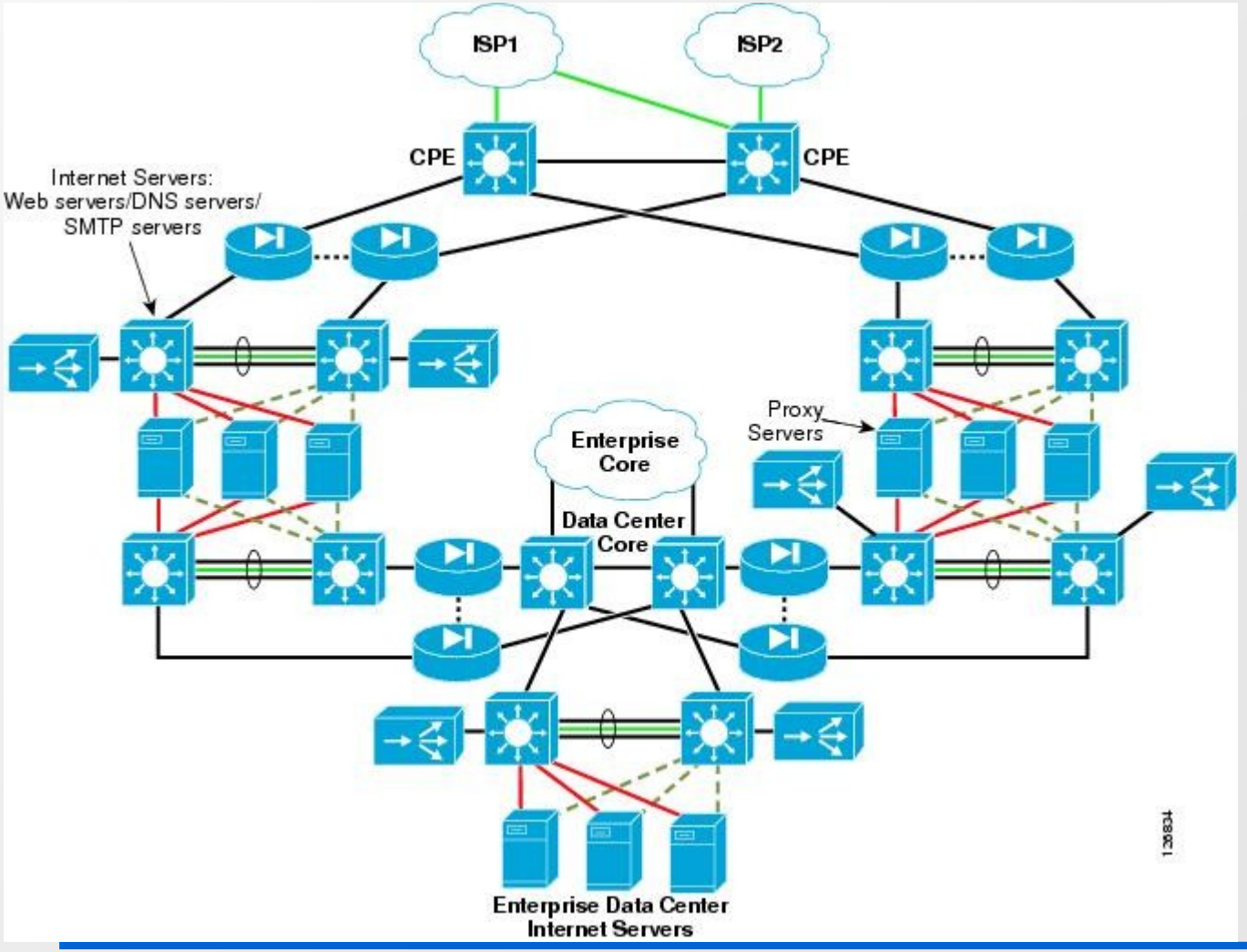
Correlation and dependence

- As expected the y values are fully dependent of x
- z is also dependent on x but in a bit smaller amount
- Notice how helpful the option 'coeff' is. It allows us to compare easily to different correlations
- In most cases we care about the value of the correlation at the $R_{xy}(0)$
- Lets now see what happens if the two data sets are 'independent'

Correlation and dependence



Network Measurements



Guidelines for performing network experiments

- Before you start with your network measurements, you should take into consideration many parameters

- In which network you will perform them?
  - If you want to find the response time of several servers you may choose a low loaded network with high bandwidth
  - If you want to find out the most popular website of the month you may choose a very big network with many users (eg university)

- Do you have the appropriate privileges?
  - Many networks nodes do not have available network monitor tools for non root accounts

Guidelines for performing network experiments

- For how long should your experiment run?
  - Searching for the mean daily data traffic, means that you have to run your measurements for several days, in order to decrease the statistical error
  - Finding the maximum throughput of a link should not take more than a day
- In which hour should perform your experiments?
  - A web-server response time may be affected, by the hour. During the night may be less. Take also into account and time differences with other countries

Guidelines for performing network experiments

- Can you find any other possible parameters?

Setup of long running experiments

- As we said before some network experiments can take days, months or even years!
- This is very painful, if all the commands should be executed by the hand
- Instead, we use some background Bash script or cron jobs
- Bash scripts may have a loop that sleeps for an amount of time and wakes up to perform a task (eg traceroute)
- These scripts may be executed with the *nohup,* so the user can close the terminal, even logout with the script still running

Setup of long running experiments

- Cron jobs is a way to run commands in specific times, even after a system reboot
- Cron provides a very powerful syntax
- Cron jobs is much more safer than loop-based scripts, in long running experiments
- For more info:
  - man cron
  - man nohup
  - http://tldp.org/LDP/abs/html/
  - http://www.corntab.com/pages/crontab-gui

Data parsing

- In network experiments, we use many times, several tools that produce text outputs (e.g ping, traceroute etc)
- Many of them, with the use of the right command line argument can produce a CSV (Comma Separated Values) output
- CSV is very easy to handle, especially in Matlab
- If the tool do not support CSV output, we right a script that parses the output and transform it, in a CSV format

Data pasring

- A parser can be written in every language
- Some languages make text parsing a piece of cake!
- Those languages provide very powerful string manipulation routines
- Can support also regular expressions, giving to the programmer more flexibility
- Some of these languages are
  - Perl
  - Python
  - Ruby
  - Bash

Data parsing with Perl

- Perl syntax is quite easy
- Perl has very powerful and fast string manipulation. Of-course supports regular expressions
- Has many built-in functions for lists, arrays and hashes
- Reads a file line by line with two lines of code!!
- Writing to file is also a piece of cake
- You do not need to become Perl guru to write a parse script. Just learn some basics on regular expressions

Data parsing with Perl

- Lets write a script that takes two command line arguments An input text file and an output

```
1  #!/usr/bin/perl -w
2
3  $file_to_parse = shift;
4  $output = shift;
```

- Now open the files for reading and writing

```
6  open(INPUT, $file_to_parse) or die("Could not open input file");
7  open(OUTPUT, ">$output") or die("Could not open output file");
```

- And read the input file, by printing each line

```
9  while(<INPUT>){
10     my($lineString) = $_;
11     chomp($lineString);
12     print "$lineString\n"
13 }
```

- Chomp() just removes the trailing '\n'

Data parsing with Perl

- In order to run it you must have installed the Perl interpreter
- chmod the file, in order to be executable
  - *chmod 755 parse.pl*
- Run the script
  - *./parse.pl dig_output.txt dig_times.txt*
- But our script until now does nothing... Lets do some real parsing!

Data parsing with Perl

- Take a careful look in the output of the dig tool
- We care only for the number that indicates response time of the DNS server in milliseconds
- All other text field should be skip
- So we change a little bit the while loop

```
 9  while(<INPUT>){
10      my($lineString) = $_;
11      chomp($lineString);
12
13      if($lineString =~ m/;; Query time:/){
14          @tokens = split(/ /, $lineString);
15          print OUTPUT "$tokens[3]\n";
16      }
17  }
```

Data parsing with Perl

```
 9  while(<INPUT>){
10      my($lineString) = $_;
11      chomp($lineString);
12
13      if($lineString =~ m/;; Query time:/){
14          @tokens = split(/ /, $lineString);
15          print OUTPUT "$tokens[3]\n";
16      }
17  }
```

- Line 13: Checks if the line contains the ;; Query time: string
- Line 14: Splits the line in the array tokens, putting an element after a space character is read
- Line 15: The fourth element of the array tokens is the response time, so print it at the output file

- Your dig output parse script is ready!!!

Data parsing with Perl

- This was an easy script
- More complex situations may occur, but the procedure remains the same
- More info at:
  - http://turtle.ee.ncku.edu.tw/docs/perl/manual/pod/perlr
  - http://www.perl.com/pub/2000/10/begperl1.html

Plot parsed data

- Put each value of the parse script in a seperate line of a file
- Using the *load* command of Matlab, put the values of the file in an array

```
1     % Load in two seperate arrays the response times of each dns server
2 -   dns_0 = load('dig_parsed0.txt');
3 -   dns_1 = load('dig_parsed1.txt');
```

- Then you can do, whatever you like with the loaded data. Eg compare their CDFs

```
5 -   figure(1);
6 -   hold all;
7 -   [h0, stats0] = cdfplot(dns_0);
8 -   [h1, stats1] = cdfplot(dns_1);
9 -   legend('UoC DNS', 'Google DNS');
```