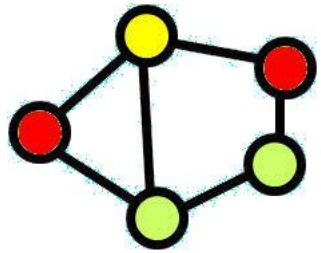
Slide supporting material



Lesson 19: Matlab® Tools for Teletraffic Engineering

Giovanni Giambene

Queuing Theory and Telecommunications: Networks and Applications
2nd edition, Springer

Binomial Coefficient and Factorial

Binomial Coefficient and Factorial

The Matlab® command to compute binomial coefficients is

nchoosek(N,K)

where N and K are non-negative integers.

This is equal to N!/[K!(N–K)!], the number of combinations of N things taken K at a time.

Note that factorial values can be computed in Matlab® by means of the Gamma function:

N! = gamma(N+1)

Generation of Random Variables

Linear Congruential Random Number Generator X $\in$ (0, 1)

- The Linear Congruential Generator (LCG) represents one of the oldest and best-known **pseudorandom number generator** algorithms. It is easy to implement and fast.
- The generator is defined by the recurrence relation:

$$X_{n+1} = \left(aX_n + c\right) \mod m$$

- where X_n is the sequence of pseudorandom values, and

  m, $m > 0$ is the "modulus"
  a, $0 < a < m$ is the "multiplier"
  c, $0 \le c < m$ is the "increment"
  X_0, $0 \le X_0 < m$ is the "seed" or "initial value".

- The period of a general LCG is at most m, and for some choices of a it can be much smaller than that. If $c = 0$, the generator is often called a **multiplicative congruential method**. In Matlab®, **randn** is based on multiplicative LCG, where the seed is determined by the clock.

Generation of Random Variables

- Use of the **statistical toolbox of Matlab®**: a lot of information can be acquired by typing "help stats".
- Some random number generators supported:
  - **rand**([1,v_max]) to generate an array of v_max values with uniform distribution between 0 and 1;
  - **randn**([1,v_max]) to generate an array of v_max values with Gaussian distribution, null mean value, and unitary variance;
  - **exprnd**(ones([1,v_max])/lambda) to generate an array of v_max values with exponential distribution and mean value 1/lambda.

Histograms

Histogram of Random Variables

- Histograms are experimental tools to characterize the probability density functions (pdfs) of random variables. Let us consider a random variable X defined on real numbers.
- **We can divide the real axis in intervals (also called 'bins') with the same size L.** Then, we repeat n times the experiment characterizing random variable X, recording how many times the outcomes fall into a generic interval; let x_j denote the outcome of the experiment at the j-th trial. We can thus show in a bar graph, called histogram, the number of times n_i that x_j falls into the generic i-th bin.
- If the number of occurrences in each interval is divided by the total number n of trials we have the **relative frequencies $f_i = n_i/n$**.

Histogram of Random Variables (cont'd)

- As the size L of the bins in abscissa reduces and the number of trials increases (i.e., $n \to \infty$), the **piecewise constant curve with horizontal segments of length L and height f_i/L tends to be more smoothed and approaches the pdf of X**.

- There are different methods to determine the size of the bins in order to achieve a good smoothed histogram. The **rule-of-thumb by Freedman-Diaconis determines the bin size L as a function of the number of trials n** as:

$$L = 2 \times IQR \times n^{-1/3}$$

where IQR is the interquartile range, obtained as IRQ = Q3–Q1, where Q1 is the first and Q3 is the third quartile of the data: Q1 = $PDF^{-1}(0.25)$ and Q3 = $PDF^{-1}(0.75)$, being PDF^{-1} the inverse of the Probability Distribution Function (PDF).

Histogram of Random Variables (cont'd)

- We have a vector of values assumed by a random variable and we need to derive some important measures, such as: mean, variance, and the pdf that can be approximated by means of a histogram (actually the corresponding piecewise constant curve).
- "mean(.)" and "var(.)" Matlab® commands can be used. Moreover, the "hist(.)" function can be used to generate histograms, collecting the occurrences of a random variable in each bin. The histogram can be plotted by means of the "bar(.)" command of Matlab®. For instance, we use the command

$$N = hist(Y,X)$$

where X is a vector of the bins and Y is a vector of the random values.

This command returns the occurrences of Y in the **bins** with centers specified by X. The first bin includes data between $-\infty$ and the first center and the last bin includes data between the last center and $+\infty$.

Histogram of Random Variables (cont'd)

histogram.m

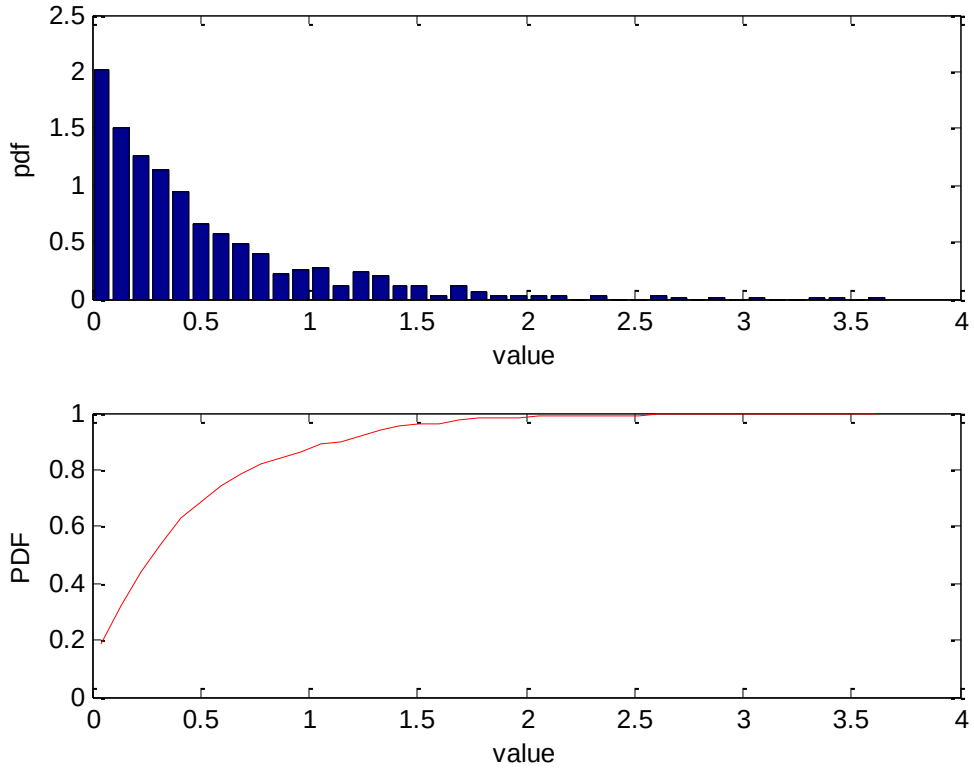
```
clear all
% We consider that the stored vector statis.txt contains the values assumed by the
% random variable for which we would like to determine the histogram.
% The values of the random variable are stored in ASCII format in statis.txt.
% We read the values in statis.txt as follows:
% load statis
% However, in the example below we use a random variable 'statis' generated by Matlab according to
% an exponential distribution (alternatives are provided for uniform and normal distributions) with
% v_max samples
v_max=1000;
%statis=randn([1,v_max]); % for a Gaussian random variable with null mean and unitary variance
%statis=rand([1,v_max]); % for a uniform random variable from 0 to 1
lambda=2; statis=exprnd(ones([1,v_max])/lambda); % for an exponentially-distributed random
%variable with mean value 1/lambda
%
N=40;  % N this is the number of classes (bins) to be used for the histogram
%  If the number of samples of the random variable is 1000, N should not be greater than 100 in order
%  to have a sufficiently-reliable estimate of the frequency of each class.
```

Histogram of Random Variables (cont'd)

```
[Y,X] = hist(statis,N); % This histogram command saves in Y the number of occurrences of the classes in X
L=X(2)-X(1); % This is needed to derive the amplitude of each class (all classes have the same bin size)
tot=sum(Y.*L);
Z=Y./tot; % Normalization of Y values so that the area below the graph is 1: Z represents a probability density function, pdf
subplot(211)
bar(X,Z)
xlabel('value')
ylabel('pdf')
% Determination of the Probability Distribution Function, PDF, as an integral (made on rectangles) of the pdf
l=length(Z);
PDF(1)=Z(1)*L;
for i =2:l
  PDF(i)=PDF(i-1)+Z(i)*L;
end
% As an alternative we could simply use:   PDF=cumsum(Z*L);
subplot(212)
plot(X,PDF,'-r')
xlabel('value')
ylabel('PDF')
mean(statis) % Computation of the mean of statis
var(statis)  % Computation of the variance of statis
```

Histogram of Random Variables (cont'd)

- These are the results (i.e., pdf and PDF) we achieve for the exponential distribution with $\lambda = 2$.



Confidence Intervals: Reliability of Results

Simulations and Random Numbers

- The expression **"Monte Carlo (MC) method"** is very general.
- MC methods are stochastic techniques, meaning they are based on the use of random numbers and statistics to investigate problems.
  - You can find MC methods used in everything from economics to nuclear physics to traffic engineering problems.
- Computer-generated numbers are not really random, since computers are deterministic. But, **given a number to start with, the seed**, a number of mathematical operations can be performed on the seed in order to generate **unrelated pseudorandom numbers** (e.g., the LCG method show at the beginning of this lesson).
  - **For a given seed, we have a given sequence of values assumed by a random variable.**

Confidence Intervals: Reliability of Results

- **We repeat the simulations for a better reliability of results using each time a different seed**. Let us suppose to make J independent runs to evaluate a given quantity of an experiment, and let λ_i, i = 1, …, J be the results obtained.

- The **mean value** is computed as:

$$\lambda_{mean} = \sum_{i=1}^{J} \lambda_i / J$$

- The **standard deviation** is computed as:

$$\lambda_{sd} = \frac{\sqrt{\sum_{i=1}^{J} (\lambda_i - \lambda_{mean})^2 / (J-1)}}{\sqrt{J}}$$

Confidence Intervals: Reliability of Results (cont'd)

- The probability that the value of the random variable falls into the interval $[\lambda_{mean}(1-\text{tolint}/2), \lambda_{mean}(1+\text{tolint}/2)]$ is equal to *tolevel* (typical values of *tolevel* are 0.90, 0.95, 0.99) where

$$\text{tolint} \cong \alpha \frac{\lambda_{sd}}{\lambda_{mean}}$$

  and α = Quantile[StudentTDistribution(J–1), 1–(1–*tolevel*/2)] is tabulated as shown in the next slide depending on *tolevel* and J.

- **The amplitude of the confidence interval is $\alpha \times \lambda_{sd}$ around the central, mean value.**

- Relatively-large confidence intervals [measured in percentage as $\pm 100\alpha\lambda_{sd} / (2\lambda_{mean})$] highlight non-reliable results.

Confidence Intervals: Reliability of Results (cont'd)

- The following table gives the values of α as function of the number of independent results (J repetitions) and of the tolerance level *tolevel.* For instance, we need to use α = 2.306 for the **95-th percentile confidence intervals** with 9 repetitions.

J	***tolevel=90%***	***tolevel=95%***	***tolevel=99%***
3	*2.920*	*4.303*	*9.925*
4	*2.353*	*3.182*	*5.841*
5	*2.132*	*2.776*	*4.604*
6	*2.015*	*2.570*	*4.032*
7	*1.943*	*2.447*	*3.707*
8	*1.895*	*2.365*	*3.500*
9	*1.860*	*2.306*	*3.355*
10	*1.833*	*2.262*	*3.250*

Confidence Intervals: Reliability of Results (cont'd)

The "errorbar(.)" command of Matlab® can be used to plot graphs with reliability intervals.

For instance:

errorbar(X,Y,E)

plots Y with error bars [Y–E Y+E].

Confidence Intervals: an Example

intervals_confidence.m

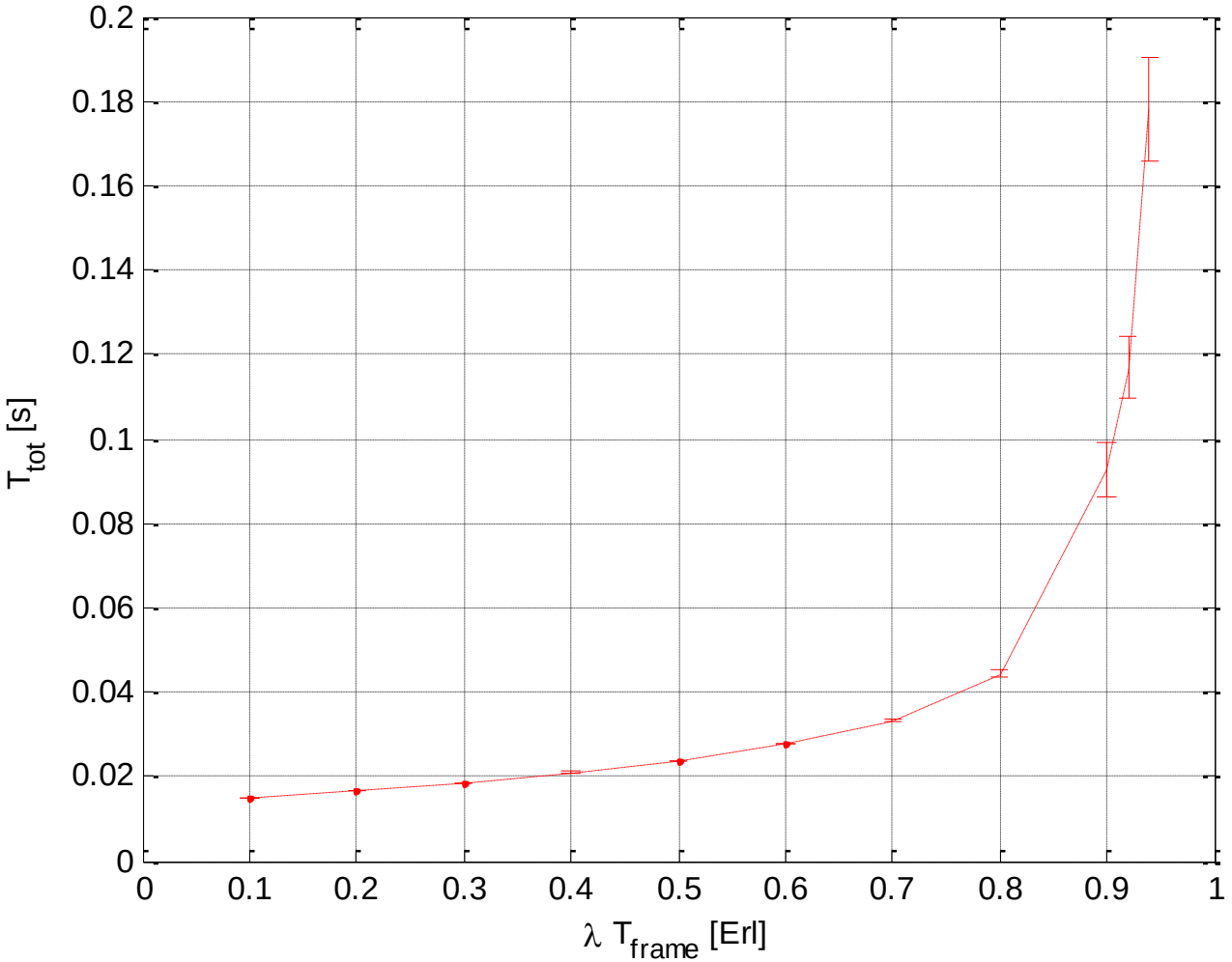
```
% Simulation with 9 repeated runs (results) for each point

X=[0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 0.92 0.94];
delay_sim(1, :)=[0.0150   0.0150   0.0150   0.0150   0.0148   0.0149   0.0151   0.0150  0.0150];
delay_sim(2, :)=[0.0166   0.0167   0.0165   0.0167   0.0167   0.0167   0.0167   0.0166  0.0167];
delay_sim(3, :)=[0.0186   0.0185   0.0183   0.0187   0.0186   0.0187   0.0185   0.0187  0.0190];
delay_sim(4, :)=[0.0210   0.0209   0.0206   0.0210   0.0210   0.0212   0.0209   0.0213  0.0211];
delay_sim(5, :)=[0.0236   0.0239   0.0231   0.0238   0.0239   0.0239   0.0237   0.0242  0.0241];
delay_sim(6, :)=[0.0277   0.0274   0.0271   0.0278   0.0279   0.0279   0.0275   0.0280  0.0280];
delay_sim(7, :)=[0.0336   0.0334   0.0308   0.0338   0.0338   0.0339   0.0330   0.0339  0.0337];
delay_sim(8, :)=[0.0449   0.0446   0.0390   0.0445   0.0453   0.0453   0.0446   0.0459  0.0453];
delay_sim(9, :)=[0.0968   0.0924   0.0500   0.0975   0.0953   0.0970   0.0933   0.1102  0.0992];
delay_sim(10, :)=[0.1231   0.1157   0.0692   0.1177   0.1156   0.1235   0.1221   0.1385  0.1257];
delay_sim(11, :)=[0.1650   0.1577   0.1605   0.1597   0.1643   0.1614   0.2016   0.1786  0.2538];
```

Confidence Intervals: an Example (cont'd)

```
% calculating confidence intervals
% coefficient alpha (renamed below k) for 9 repeated runs
k=2.306;
for tt=1:11
    sum_del=0;
for j=1:9
    sum_del=sum_del + ((delay_sim(tt, j)-mean(delay_sim(tt, :)))^2)/8;
end
    sd_del=sqrt(sum_del)/sqrt(9);
    Conf_Del(tt)=sd_del*k/2;
    Del(tt)=mean(delay_sim(tt, :));
end
errorbar(X, Del, Conf_Del, '--r')
xlabel('\lambda T_{frame} [Erl]')
ylabel('T_{tot} [s]')
grid
```

Confidence Intervals: an Example (cont'd)

Graph of simulation results with 95-percentile confidence intervals



Iterative Methods for Markov Chains

Erlang-B Formula Computation, M/M/S/S

- The blocking probability P_b of an M/M/S/S queuing systems with input traffic of ρ Erlang is expressed by the formula below:

$$P_b \equiv P_S = \frac{\rho^S}{S!\sum_{i=0}^{S}\frac{\rho^i}{i!}}$$

Erlang-B formula

- The Erlang-B formula cannot be directly computed when the number of servers, S, is high due to the presence of factorial terms.
- An **iterative method** has been adopted to compute the Erlang-B formula for increasing number of resources S as shown in the next slide.

Erlang-B Formula Computation, M/M/S/S (cont'd)

- Iterative method to compute the Erlang-B formula:

$$P_b(\rho, 0) = 1$$

$$\frac{1}{P_b(\rho, S)} = 1 + \frac{i}{\rho P_b(\rho, S-1)}$$

erlangb.m

```
function erlangb=erlangb(a,c)
ris=1;
for i=1:c
 ris=1/(1+i/(a*ris));
end
erlangb=ris;
return
```

- In the next slide we show a Matlab® script to compute the **server utilization** $\rho(1 - P_b)/S$ of an M/M/S/S queuing system where ρ is the maximum traffic load allowing $P_b < 1\%$ for a certain S.

Erlang-B: Optimization of Resources with a Constraint

utilization.m

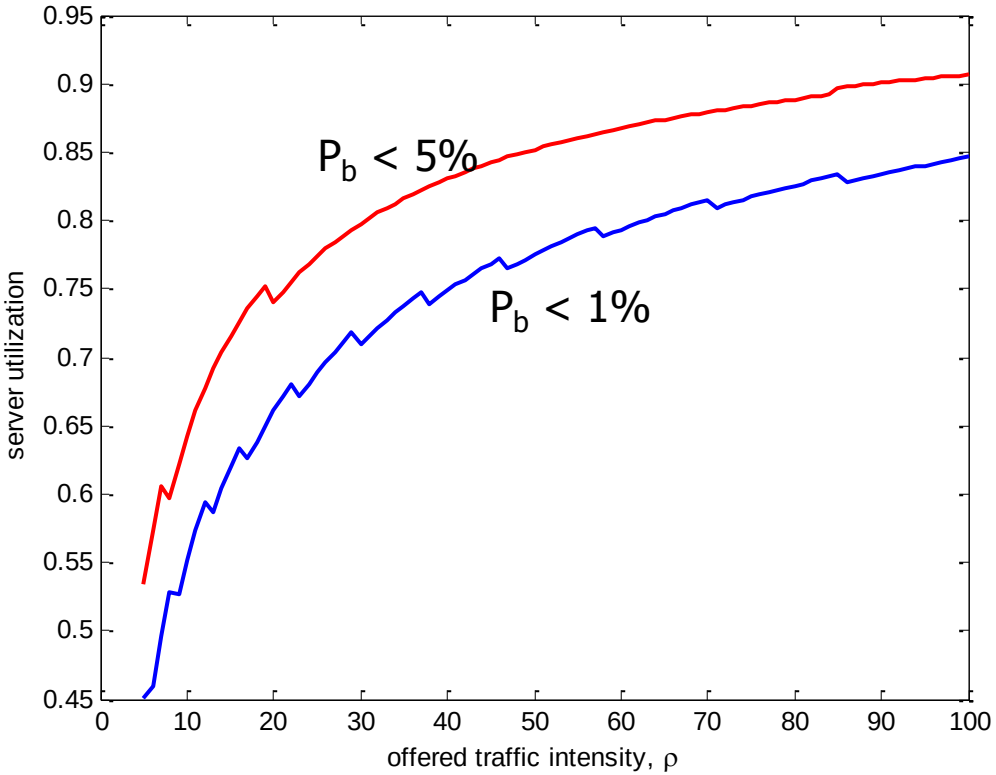
```
clear all
rho=5:1:100; %input traffic intensity in Erlangs
l=length(rho);
req=0.01; %requirement on the blocking probability, 1%
for j=1:l
load=rho(j);
S=0;
Pb=1;
while Pb>req
   S=S+1;  % while cycle on the number of servers
   Pb=erlangb(load,S); % use of the ERLANG-B formula in a function
end
u(j)=load*(1-erlangb(load,S))/S;
end
plot(rho,u,`-')
xlabel('offered traffic intensity, \rho')
ylabel('server utilization')
```

Erlang-B: Optimization of Resources with a Constraint

This is the resulting graph of the server utilization

$$\rho(1 - P_b)/S$$

of an M/M/S/S system for both $P_b < 1\%$ and $P_b < 5\%$ as a function of ρ.



Numerical Inversion of Probability Generating Functions

Numerical Inversion of Generating Functions

- Let us consider for instance the following PGF:

$$X(z) = \frac{1}{4} + \frac{1}{4}z + \frac{1}{4}z^2 + \frac{1}{4}z^3$$

- We can immediately invert this PGF to obtain the following distribution:

$$X = \begin{cases} 0, & \text{Prob} = \frac{1}{4} \\ 1, & \text{Prob} = \frac{1}{4} \\ 2, & \text{Prob} = \frac{1}{4} \\ 3, & \text{Prob} = \frac{1}{4} \end{cases}$$

Numerical Inversion of Generating Functions

- The inversion of this PGF can also be obtained by noting that a PGF can be seen as a **Taylor series expansion centred at z = 0** (i.e., MacLaurin series expansion):

$$\text{Prob}\{X = k\} = \frac{1}{k!}\frac{d^k}{dz^k}X(z)\Big|_{z=0}$$

- We can use the **Matlab® symbolic toolbox** to compute the derivatives of X(z) at z = 0 as detailed in the following code:

Numerical Inversion of Generating Functions (cont'd)

We consider the Matlab® command to differentiate

diff(S,'v', n)

This command allows us to differentiate n times expression S with respect to the symbolic variable v.

Numerical Inversion of Generating Functions (cont'd)

- The Matlab® script code below is used to invert the PGF:

inversion_PGF.m

```
clear all
syms z
Xz=(1/4)+(1/4)*z+(1/4)*z^2+(1/4)*z^3;
for i=2:5
prob(i)=eval(diff(Xz,'z',i-1))/gamma(i);
end
prob(1)=eval(Xz);
z=0;
p=eval(prob);
```

Numerical Inversion of Generating Functions (cont'd)

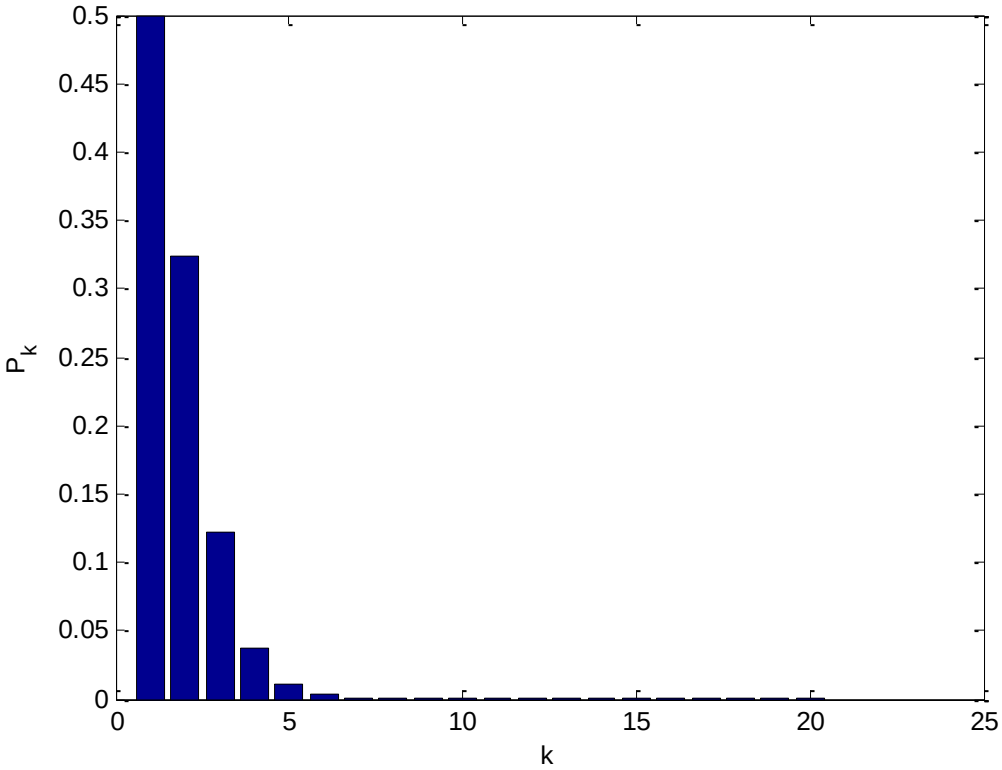
- Let us consider another example to invert the following PGF P(z) related to the number of requests in an M/D/1 queue:

$$P(z) = (1-\rho)\frac{(z-1)e^{\rho(z-1)}}{z - e^{\rho(z-1)}}$$

- This PGF cannot be inverted in a closed form and the proposed Matlab® approach can be used to numerically determine the distribution P_k corresponding to P(z):

Numerical Inversion of Generating Functions (cont'd)

- The resulting M/D/1 state probability distribution for ρ = 0.5 Erlangs is shown in the graph below:



Numerical Solution of Equations

Numerical Solution of Equations

The numerical solution of non-linear **systems of equations** is supported by the following Matlab® command of the Optimization toolbox:

X=fsolve(FUN,X0)

The "fsolve(.)" method uses the vector X0 as starting point and tries to solve the equations in FUN (equations of the form 'FUN = 0'). fsolve adopts an iterative method to return a solution vector X of the equations in FUN.

In case of multiple solutions, the solution determined by the fsolve method depends on the starting point selected in X0.

Numerical Solution of Equations (cont'd)

- **The fsolve Matlab® routine permits to solve sets of non-linear equations using the (quasi-)Newton method.**
- The Newton method uses an iterative process to approximate one root of a function $f(x)$, i.e., $f(x) = 0$.
  - The specific root that the process locates depends on the initial value x_0.
  - It is useful to determine initially the number of solutions allowed by the problem.
- Newton method in one variable: given a function $f(x)$ and its derivative $f'(x)$, we begin with a first guess x_0 for a root of the function. Provided that the function $f(x)$ has a "reasonable behavior", a better approximation of the root is x_1 obtained as:

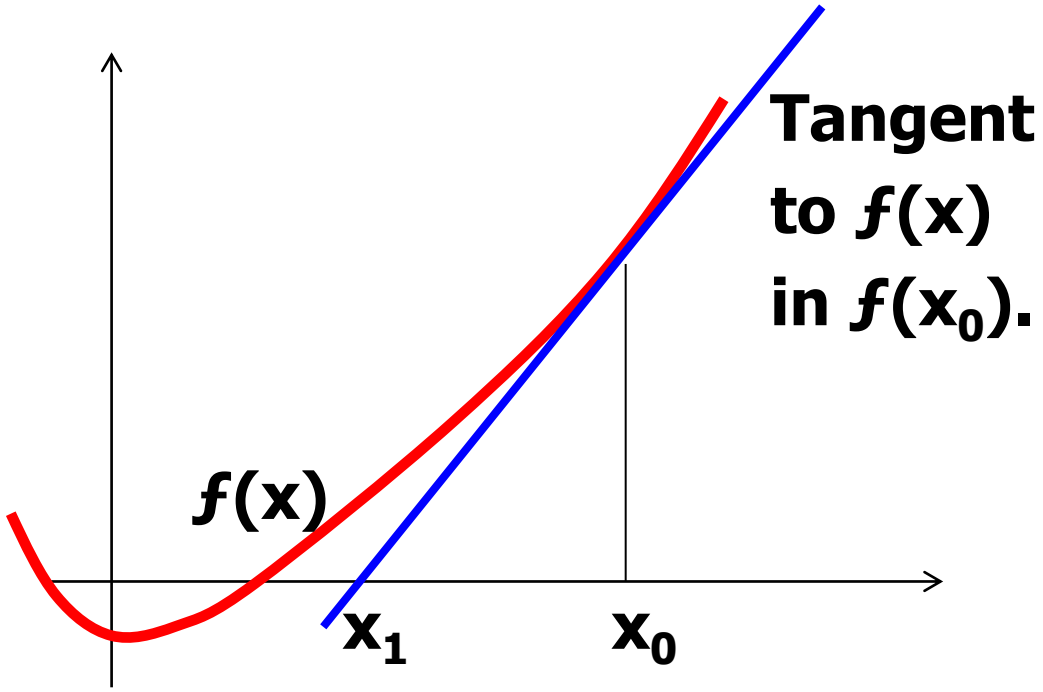
$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

- **Geometrically, x_1 is the intersection with the x-axis of a line tangent to $f(x)$ at $f(x_0)$.**

Numerical Solution of Equations (cont'd)

- The process is repeated according to the equation below until a sufficiently-accurate root is reached (a relative error stop criterion is used):

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$



Numerical Solution of Equations: a First Example

- We have to solve the following transcendent equation F(σ) in the unknown σ:

$$\sigma = e^{(\sigma-1)/\rho} \quad \Leftrightarrow \quad F(\sigma) = \sigma - e^{(\sigma-1)/\rho} = 0$$

- We can use an iterative method and the "fsolve(.)" function in Matlab®:

cover.m

```
r=0.5;
save dati r
sigma=fsolve('dm1' , 0)
```

Equation in σ (here s variable) of the form F(σ) = 0

dm1.m

```
function dm1 = dm1(s)
load dati
dm1 = s-exp((s-1)/r);
return
```

Numerical Solution of Equations: a First Example

- To improve the accuracy of the solution (up to 10^{-6}) we need to use 'optimset' in the "fsolve(.)" command as follows:

```
sigma=fsolve('dm1' , 0, optimset('TolFun',1e-6))
```

Numerical Solution of Equations: WiFi Analysis

- Let us consider another example related to the saturated analysis of the WiFi access as shown in Lesson No. 10.

- τ is the probability that a station transmits in a randomly-chosen slot time. A transmission occurs when the backoff time counter is equal to zero, regardless of the backoff stage:

$$\tau = \sum_{i=0}^{m} b_{i,0}$$

- p is the collision probability for a general transmission attempt: the probability that a transmitted packet encounters a collision, is the probability that, in the same time slot, at least one station of the n– 1 remaining ones transmits:

$$p = 1 - (1 - \tau)^{n-1}$$

Numerical Solution of Equat.: WiFi Analysis (cont'd)

- p and τ equations form a system of non-linear equations that can be expressed as follows:

$$\begin{cases} \tau = \dfrac{2(1-2p)}{(1-2p)(W+1)+pW\left[1-(2p)^m\right]} \\ p = 1-(1-\tau)^{n-1} \end{cases}$$

- This system admits a single solution that depends on parameters:
  - Initial contention window, W
  - Number of WiFi stations, n
  - Number of backoff stages, m.

Numerical Solution of Equat.: WiFi Analysis (cont'd)

- We can use the "fsolve(.)" function in Matlab® (iterative method) to determine the solution for the system in p and τ:

cover2.m

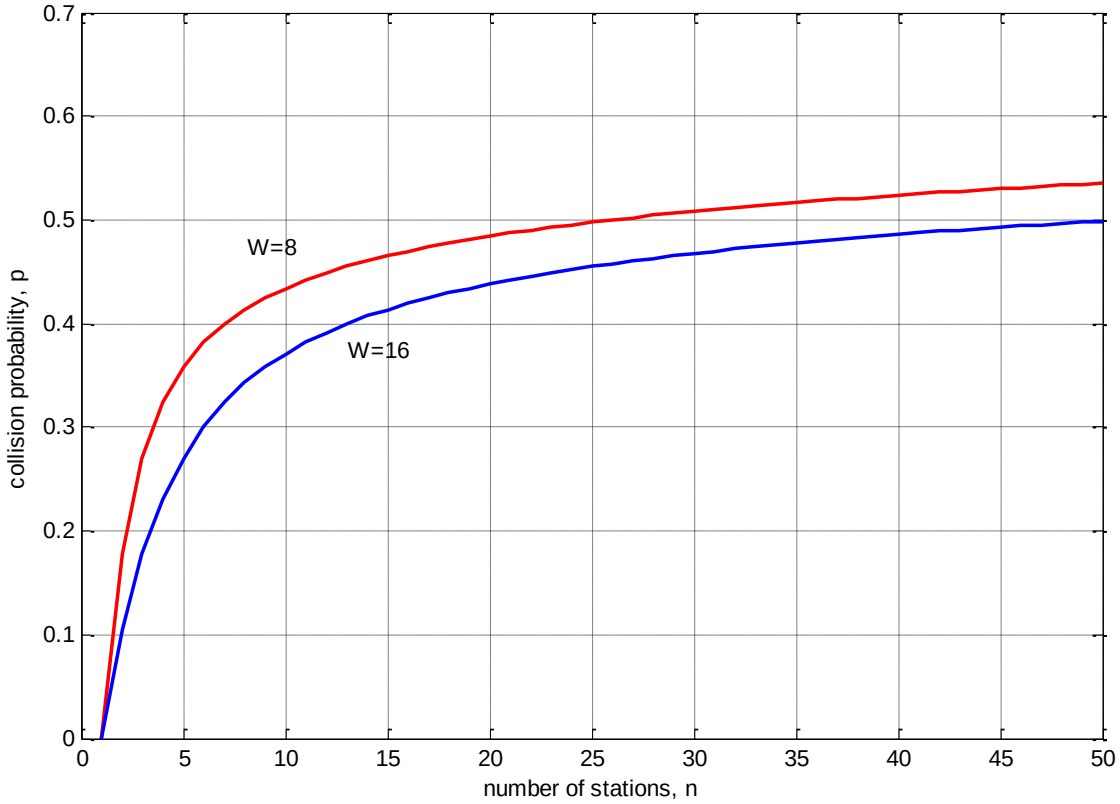
```
W=32;
m=16; % a lower value is used in the standard
enne=1:1:50;
l=length(enne);
for i=1:l
    n=enne(i);
    save dati W m n
    collision(i)=fsolve('wifi' , 0);
end
plot(enne,collision,'-b')
xlabel('number of stations, n')
ylabel('collision probability, p')
grid
```

wifi.m

```
function wifi = wifi(p)
load dati

tau=2*(1-2*p)/((1-
2*p)*(W+1)+p*W*(1-(2*p)^m));

wifi = p-1+(1-tau)^(n-1);

return
```

Numerical Solution of Equat.: WiFi Analysis (cont'd)

- The graph below shows the resulting behavior of p as a function of n:



Thank you!

giovanni.giambene@gmail.com