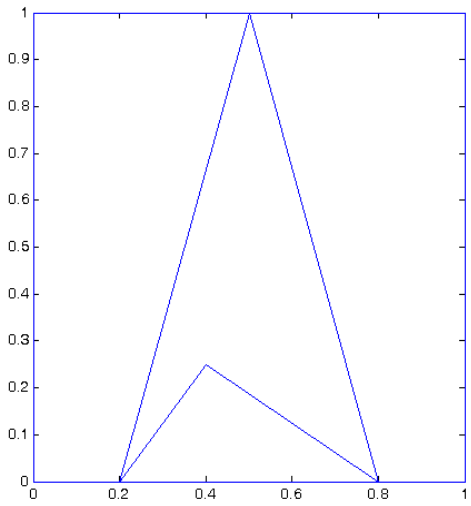
Exercise 1 – Monte-Carlo simulation and Random number generation

1.1

Using Monte Carlo simulation, define the area of the polygon P specified by the vertices {(0.2; 0), (0.4;0.25), (0.8;0), (0.5;1)}.



a) Verify the sampling experiment by constructing a plot of the result.

b) What is your estimate for the area of P? Define a 1-δ confidence interval for the estimate with δ=0.05.

c) Compare your solution to the actual area of the polygon.

d) How does the sample size of your experiment affect your estimate and the confidence interval? Repeat the experiment using different sample sizes. Create a plot of the length of the confidence interval against sample size.

e) What is the worst-case sample size for absolute error ε=0.05 and confidence level 1-δ=0.95

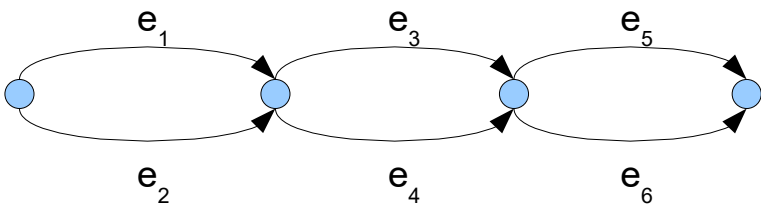
f) Repeat the entire simulation experiment 1000 times using the worst-case sample size and define the proportion of cases where the absolute error is less than ε=0.05.

g) Construct 1000 confidence intervals for the estimate of the area and define their coverage, i.e. the proportion of CI's that contain the actual value of the area.

In Matlab, you may find the functions *rand, inpolygon,* and *polyarea* helpful.

1.2

Determine the expected duration of a project described by the following activity network:



Each arc represents an acitivity and each node a milestone. An activity can not be started until all activities leading to the milestone that represents the starting point of the activity are completed .

The duration of each activity e_i is exponentially distributed with mean μ_i. The total length of the project with given realizations of activity durations is the maximum duration of the paths of the network. The mean durations are $(\mu_1,\ldots,\mu_6)=(0.5,1,1.5,2,0.25,1)$.

To simulate exponentially distributed random variables, you can use the command `exprnd` in Matlab.

1.3

Write codes that generate triangularly distributed random variates utilizing both the the inverse transform method and the acceptance-rejection method. The probability density function of the triangular distribution is

$$f(x)=\begin{cases}\dfrac{2(x-a)}{(b-a)(c-a)}, & \text{if } a\le x\le c\\ \dfrac{2(b-x)}{(b-a)(b-c)}, & \text{if } c<x\le b\\ 0, & \text{otherwise}\end{cases}$$

where *a* and *b* define the minimum and maximum values and *c* is the mode; *a*<*c*<*b*. The cumulative distribution function is

$$F(x)=\begin{cases}0 & \text{if } x< a\\ \dfrac{(x-a)^2}{(b-a)(c-a)}, & \text{if } a\le x\le c\\ 1-\dfrac{(b-x)^2}{(b-a)(b-c)}, & \text{if } c< x\le b\\ 1, & \text{if } x> b\end{cases}$$

Verify your codes by plotting the frequency distribution (histogram) of samples of the generated random variates.

Then compare, which method is faster. Notice, for instance, that '`a+rand*(b-a)`' may be considerably faster than 'unifrnd(a,b)'.

In Matlab, you might find the functions *rand*, *unifrnd*, *hist,* and *tic/toc* helpful.

1.4 (Demo)

Examine the basic properties of the built-in random number generator of the software you are using. In Matlab, for instance, write in the command window: *help rand*.

Generate a sample of 50 random numbers with the generator and use the chi-square test to assess whether the numbers are uniformly distributed.

Repeat the tests for 100 independent sequences of random numbers and define the proportion of cases in which the hypothesis of uniformity was rejected.

The cumulative distribution function of the chi-square distribution is given by *chi2cdf*. A code for the distribution is as follows, in case the Statistics toolbox of Matlab is not available, see page 4 (this was once an issue in the computer rooms...)

Cumulative distribution function for the chi-square distribution

The chi-square distribution with k degrees of freedom is the same as the gamma distribution with parameters $\alpha=k/2$ and $\beta=2$. The cumulative distribution function for the gamma distribution, assuming that α is a positive integer is

$$F(x)=1-e^{-x/\beta}\frac{\sum_{j=0}^{a-1}(x/\beta)^j}{j!}, x>0.$$

If α is not a positive integer, a closed form does not exist. Therefore, set up the chi-square test in the exercise so that k-1 is even or use the closest possible integer to (k-1)/2 as approximate degrees of freedom. The code for the distribution might look something like

```
function y = x2cdf(x,k)
%Evaluates the cumulative distribution function of the chi-
%square distribution at x with k df.

alpha = k/2;
beta = 2;

for j = 0:alpha-1

      if j == 0
            a(j+1) = ((x/beta)^j);
      else
            a(j+1) = ((x/beta)^j)/prod(1:j);
      end
end

y = 1-exp(-x/beta)*sum(a);
```