Data and Decision Making AT1-A

Joshua McCarthy #13448550

16/08/2020

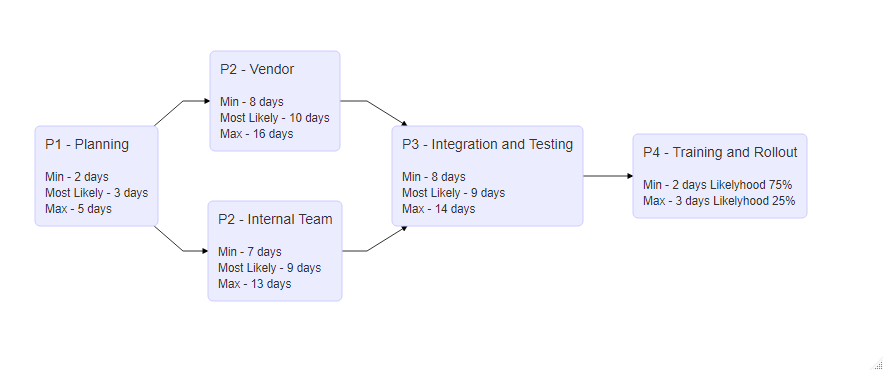
### Libraries

library(tidyverse)  
library(DiagrammeR)  
library(GoFKernel)  
library(scales)  
library(SimJoint)  
library(Matrix)  
  
# remmember

# Project Simulation Using Monte Carlo

## Sequence Diagram

# Create sequence graph  
  
mermaid("  
graph LR  
 A(P1 - Planning<br><br><small>Min - 2 days<br>Most Likely - 3 days<br>Max - 5 days</small>)  
 B(P2 - Vendor<br><br><small>Min - 8 days<br>Most Likely - 10 days<br>Max - 16 days</small>)  
 C(P2 - Internal Team<br><br><small>Min - 7 days<br>Most Likely - 9 days<br>Max - 13 days</small>)  
 D(P3 - Integration and Testing<br><br><small>Min - 8 days<br>Most Likely - 9 days<br>Max - 14 days</small>)  
 E(P4 - Training and Rollout<br><br><small>Min - 2 days Likelyhood 75%<br>Max - 3 days Likelyhood 25%</small>)  
   
 A-->B  
 A-->C  
 B-->D  
 C-->D  
 D-->E  
 ")

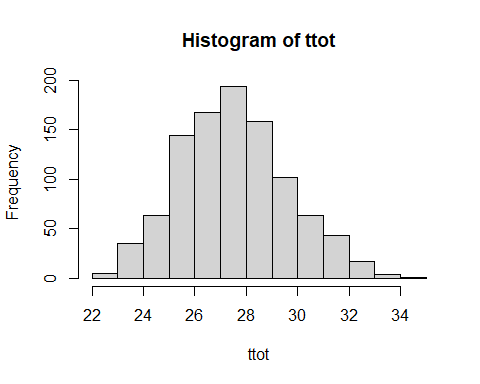


Sequence Diagram

## Simulation Using Triangular Distribution

# create data frame of tasks and times for the first 4 cases  
  
task\_durations <- data.frame(task = c("p1","p2v","p2i","p3"),  
 tmin = c(2,8,7,8),  
 tml = c(3,10,9,9),  
 tmax = c(5,16,13,14))

## Monte Carlo Simulation with Triangle Distribution  
  
# inverse triangle function  
  
inv\_triangle\_cdf <- function(P, vmin, vml, vmax){  
  
 Pvml <- (vml-vmin)/(vmax-vmin)  
   
 return(ifelse(P < Pvml,  
 vmin + sqrt(P\*(vml-vmin)\*(vmax-vmin)),  
 vmax - sqrt((1-P)\*(vmax-vml)\*(vmax-vmin))))  
}  
  
# number of trials  
  
n=1000  
  
set.seed(98)  
  
# trials df  
  
tsim <- as.data.frame(matrix(nrow=n,ncol=nrow(task\_durations)+1))  
  
# for each task  
for (i in 1:nrow(task\_durations)){  
 #set task durations  
 vmin <- task\_durations$tmin[i]  
 vml <- task\_durations$tml[i]  
 vmax <- task\_durations$tmax[i]  
   
 #generate n random numbers (one per trial)  
 psim <- runif(n)  
 #simulate n instances of task  
 tsim[,i] <- inv\_triangle\_cdf(psim,vmin,vml,vmax)   
}  
  
# calc phase 4, likelyhood of being completed in 2 days 0.75 in 3 days 0.25  
# Very naive, no interpolated distribution   
  
tsim[,5] <- runif(nrow(tsim))  
tsim[tsim[,5] < 0.25, 5] <- 3  
tsim[tsim[,5] != 3, 5] <- 2  
  
#sum costs for each trial  
ttot <- tsim[,1] + pmax(tsim[,2], tsim[,3]) + tsim[,4] + tsim[,5]  
  
#time distribution  
hist(ttot)



#mean, max, min and median time  
mean(ttot)

## [1] 27.51006

max(ttot)

## [1] 34.82005

min(ttot)

## [1] 22.40172

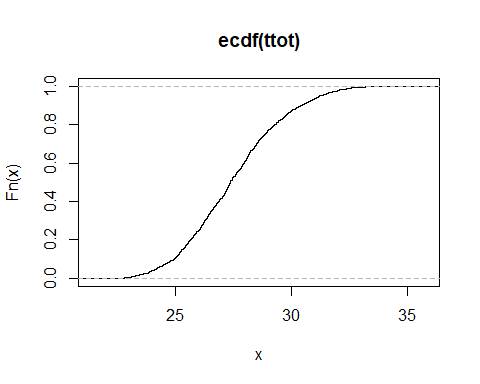
median(ttot)

## [1] 27.37217

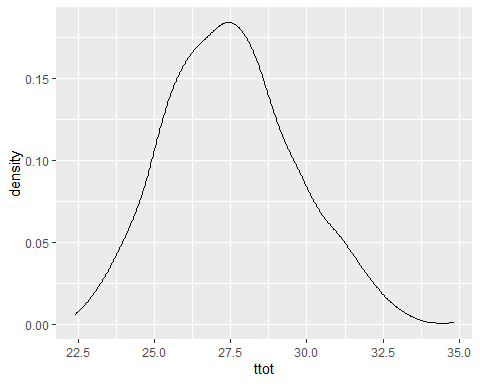
#standard deviation  
sd(ttot)

## [1] 2.107572

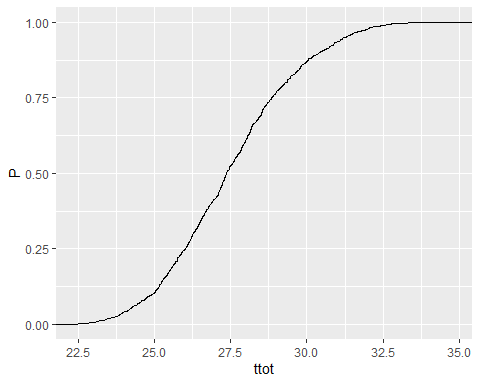
#plot cdf  
plot(ecdf(ttot))



ggplot() + geom\_density(aes(x = ttot))



ggplot() + stat\_ecdf(aes(x = ttot), geom = "line") + ylab("P")



ecdf(ttot)(27)

## [1] 0.416

quantile(ecdf(ttot),0.9,type=7)

## 90%   
## 30.41336

# why this type?

Distribution and assumption of indipendence are the model

Same probability for each time? No

Late / later

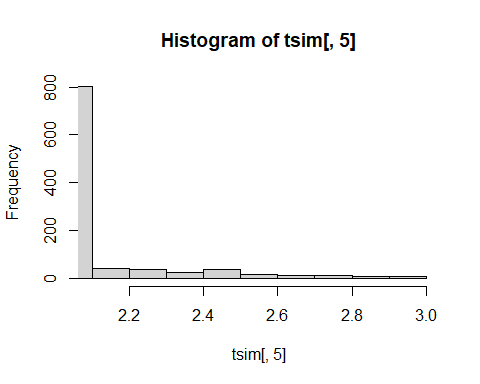
# Improvements

## Impliment PERT Distribution

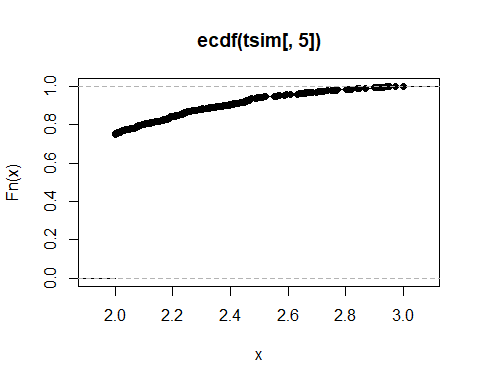
# https://www.riskamp.com/beta-pert  
  
rpert <- function( n, x.min, x.max, x.mode, lambda = 4 ){  
  
 if( x.min > x.max || x.mode > x.max || x.mode < x.min ) stop( "invalid parameters" );  
  
 x.range <- x.max - x.min;  
 if( x.range == 0 ) return( rep( x.min, n ));  
  
 mu <- ( x.min + x.max + lambda \* x.mode ) / ( lambda + 2 );  
  
 # special case if mu == mode  
 if( mu == x.mode ){  
 v <- ( lambda / 2 ) + 1  
 }  
 else {  
 v <- (( mu - x.min ) \* ( 2 \* x.mode - x.min - x.max )) /  
 (( x.mode - mu ) \* ( x.max - x.min ));  
 }  
  
 w <- ( v \* ( x.max - mu )) / ( mu - x.min );  
 return ( rbeta( n, v, w ) \* x.range + x.min );  
}  
  
pert <- rpert(1000, 7, 13, 9, lambda = 4)  
  
# number of trials  
  
n=1000  
  
set.seed(98)  
  
# trials df  
  
tsim <- as.data.frame(matrix(nrow=n,ncol=nrow(task\_durations)+1))  
  
# for each task  
for (i in 1:nrow(task\_durations)){  
 #set task durations  
 vmin <- task\_durations$tmin[i]  
 vml <- task\_durations$tml[i]  
 vmax <- task\_durations$tmax[i]  
   
 #simulate n instances of task  
 tsim[,i] <- rpert(n,vmin,vmax,vml)   
}

## Improve phase 4 distribution

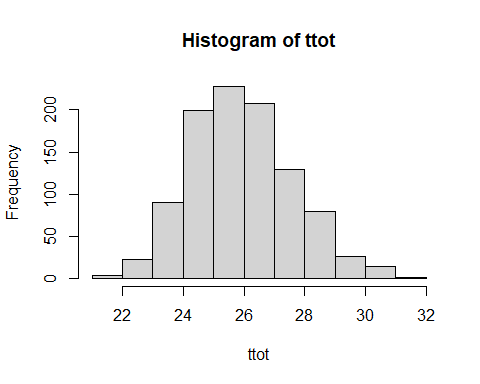
# calc phase 4, likelyhood of being completed in 2 days 0.75 in 3 days 0.25  
# Very naive, no interpolated distribution   
  
tsim[,5] <- rpert(nrow(tsim), 0, 1, 0.5, lambda = 4)  
sfq <- quantile(ecdf(tsim[,5]),0.75,type=7)  
  
tsim[tsim[,5] <= sfq, 5] <- 2  
tsim[tsim[,5] != 2, 5] <- rescale(tsim[tsim[,5] != 2, 5], to = c(2:3))  
hist(tsim[,5], xlim = c(2.1,3))



plot(ecdf(tsim[,5]))



#sum costs for each trial  
ttot <- tsim[,1] + pmax(tsim[,2], tsim[,3]) + tsim[,4] + tsim[,5]  
  
#time distribution  
hist(ttot)



#mean, max, min and median time  
mean(ttot)

## [1] 25.92015

max(ttot)

## [1] 31.10815

min(ttot)

## [1] 21.25254

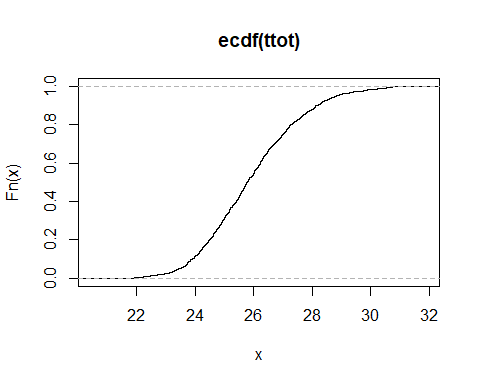
median(ttot)

## [1] 25.76896

#standard deviation  
sd(ttot)

## [1] 1.67448

#plot cdf  
plot(ecdf(ttot))



#late-later correlation

cor(tsim)

## V1 V2 V3 V4 V5  
## V1 1.000000000 -0.049728980 -0.08467255 -0.009272618 0.055813332  
## V2 -0.049728980 1.000000000 0.00371570 -0.037924524 0.005817681  
## V3 -0.084672548 0.003715700 1.00000000 -0.011781751 -0.012800199  
## V4 -0.009272618 -0.037924524 -0.01178175 1.000000000 -0.023429134  
## V5 0.055813332 0.005817681 -0.01280020 -0.023429134 1.000000000

set.seed(98)  
cormat <- matrix(runif(25, min = 0.4, max = 0.6), nrow = 5, ncol = 5)  
for (i in 1:nrow(cormat)) {  
cormat[i,i] <- 1  
}  
  
cormat <- as.matrix(forceSymmetric(cormat))  
  
cormat

## [,1] [,2] [,3] [,4] [,5]  
## [1,] 1.0000000 0.4621269 0.5718670 0.5787514 0.5419149  
## [2,] 0.4621269 1.0000000 0.5548362 0.4568522 0.4744907  
## [3,] 0.5718670 0.5548362 1.0000000 0.4538582 0.5309327  
## [4,] 0.5787514 0.4568522 0.4538582 1.0000000 0.5985139  
## [5,] 0.5419149 0.4744907 0.5309327 0.5985139 1.0000000

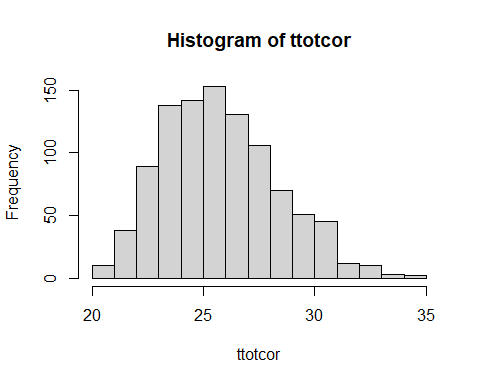
tsimcor <- tsim %>%   
 map\_df(sort) %>%   
 as.matrix()  
  
tsimcor <- SJpearson(tsimcor, cormat)$X

## Iteration = 1: square root of mean squared error in cor = 0.161425  
## Cholesky decomposition failed. Perform eigen decomposition.  
## Iteration = 2: square root of mean squared error in cor = 0.181728  
## Iteration = 3: square root of mean squared error in cor = 0.0437117  
## Iteration = 4: square root of mean squared error in cor = 0.0134497  
## Iteration = 5: square root of mean squared error in cor = 0.00407298  
## Iteration = 6: square root of mean squared error in cor = 0.00137905  
## Iteration = 7: square root of mean squared error in cor = 0.000570315  
## Iteration = 8: square root of mean squared error in cor = 8.90303e-05  
## Iteration = 9: square root of mean squared error in cor = 9.0728e-05  
## Iteration = 10: square root of mean squared error in cor = 0.000101371  
## Iteration = 11: square root of mean squared error in cor = 8.45253e-05  
## Iteration = 12: square root of mean squared error in cor = 9.12067e-05  
## Iteration = 13: square root of mean squared error in cor = 8.41472e-05  
## Iteration = 14: square root of mean squared error in cor = 0.000118192  
## Iteration = 15: square root of mean squared error in cor = 8.95425e-05  
## Iteration = 16: square root of mean squared error in cor = 5.64741e-05  
## Iteration = 17: square root of mean squared error in cor = 4.0099e-05  
## Iteration = 18: square root of mean squared error in cor = 9.10892e-05  
## Iteration = 19: square root of mean squared error in cor = 4.46935e-05  
## Iteration = 20: square root of mean squared error in cor = 3.58392e-05  
## Iteration = 21: square root of mean squared error in cor = 4.61948e-05  
## Iteration = 22: square root of mean squared error in cor = 2.18622e-05  
## Iteration = 23: square root of mean squared error in cor = 4.86942e-05  
## Iteration = 24: square root of mean squared error in cor = 3.48945e-05  
## Iteration = 25: square root of mean squared error in cor = 3.48945e-05  
## Iteration = 26: square root of mean squared error in cor = 2.18622e-05  
## Iteration = 27: square root of mean squared error in cor = 2.18622e-05  
## Iteration = 28: square root of mean squared error in cor = 2.18622e-05  
## Iteration = 29: square root of mean squared error in cor = 2.18622e-05  
## Iteration = 30: square root of mean squared error in cor = 2.18622e-05  
## Iteration = 31: square root of mean squared error in cor = 2.18622e-05  
## Iteration = 32: square root of mean squared error in cor = 2.18622e-05  
## Iteration = 33: square root of mean squared error in cor = 2.18622e-05

tsimcor <- as.data.frame(tsimcor)  
cor(tsimcor)

## V1 V2 V3 V4 V5  
## V1 1.0000000 0.4621277 0.5718675 0.5787463 0.5419289  
## V2 0.4621277 1.0000000 0.5548265 0.4568448 0.4744661  
## V3 0.5718675 0.5548265 1.0000000 0.4538334 0.5308765  
## V4 0.5787463 0.4568448 0.4538334 1.0000000 0.5985199  
## V5 0.5419289 0.4744661 0.5308765 0.5985199 1.0000000

#sum costs for each trial  
ttotcor <- tsimcor[,1] + pmax(tsimcor[,2], tsimcor[,3]) + tsimcor[,4] + tsimcor[,5]  
  
#time distribution  
hist(ttotcor)



#mean, max, min and median time  
mean(ttotcor)

## [1] 25.76437

max(ttotcor)

## [1] 34.15506

min(ttotcor)

## [1] 20.35091

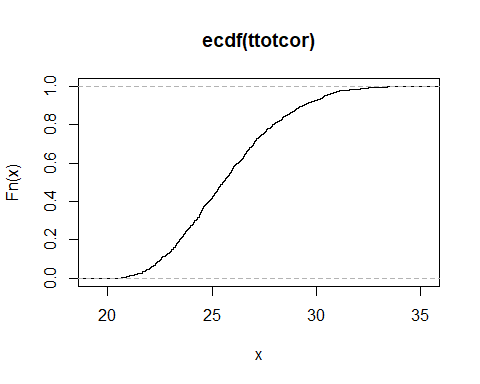
median(ttotcor)

## [1] 25.54177

#standard deviation  
sd(ttotcor)

## [1] 2.575265

#plot cdf  
plot(ecdf(ttotcor))



ecdf(ttotcor)(27)

## [1] 0.701

quantile(ecdf(ttotcor),0.9,type=7)

## 90%   
## 29.36257

pairs(tsimcor)

