CEE 204 2014 Final Exam

Question	Score
1	
2	
3	
4	
5	
6	
7	
8	
Total	

STANFORD UNIVERSITY
OFFICIAL EXAMINATION BOOK

(Coordinate Book – 8 pp.)

Name of student Steven Tsz Yiu Wong

Date of examination June 9th, 2014

Course ~~CEE~~ 204

THE STANFORD UNIVERSITY HONOR CODE

A. The Honor Code is an undertaking of the students, individually and collectively:
   (1) that they will not give or receive aid in examinations; that they will not give or receive unpermitted aid in class work, in the preparation of reports, or in any other work that is to be used by the instructor as the basis of grading;
   (2) that they will do their share and take an active part in seeing to it that others as well as themselves uphold the spirit and letter of the Honor Code.

B. The faculty on its part manifests its confidence in the honor of its students by refraining from proctoring examinations and from taking unusual and unreasonable precautions to prevent the forms of dishonesty mentioned above. The faculty will also avoid, as far as practicable, academic procedures that create temptations to violate the Honor Code.

C. While the faculty alone has the right and obligation to set academic requirements, the students and faculty will work together to establish optimal conditions for honorable academic work.

I acknowledge and accept the Honor Code.

(Signed) [signature]

Interpretations and applications of the Honor Code appear on the back cover of this examination book.

Page 1 of 7

[1] R-S ----------

Please find R codes attached

[Setup]

S and R are independent:

$$S \sim exp(\lambda = 1)$$

$$R \sim EV\ Type\ I\ (\alpha = 12, u = 7.5) \sim EV\ Type\ I\ (\mu = u + 0.5772/\alpha = 7.5481, \sigma = \pi/(\alpha\sqrt{6}) = 0.106879)$$

$$P_f = P[(g(x) = 2R - 3S) < 0]$$

[a] Numerical Integration

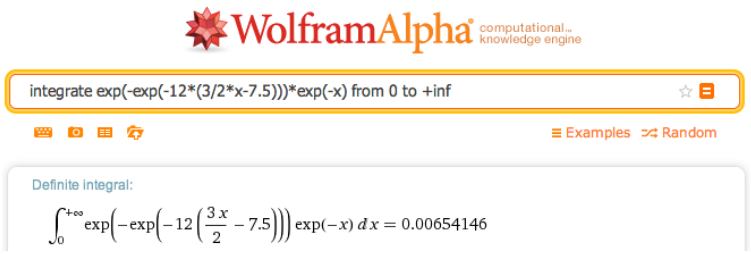
$$P_f = \iiint_{g(X)<0} f_X(x)dx = \iiint_{2R-3S<0} f_R(r:\alpha,u) f_S(s:\lambda) dr ds = \int_0^{\infty} \left[\int_{-\infty}^{r=\frac{3}{2}s} \alpha \exp\left[-\alpha(r-u) - \exp\left(-\alpha(r-u)\right)\right] dr \right] \exp(-s)\, ds$$

$$= \int_0^{\infty} \exp\left[-\exp\left[-12\left(\frac{3}{2}s - 7.5\right)\right]\right] \exp(-s)\, ds = \int_0^{\infty} \exp\left[-\exp\left[-12\left(\frac{3}{2}s - 7.5\right)\right] - s\right] ds$$

Results computed with R, and verified with WolframAlpha:

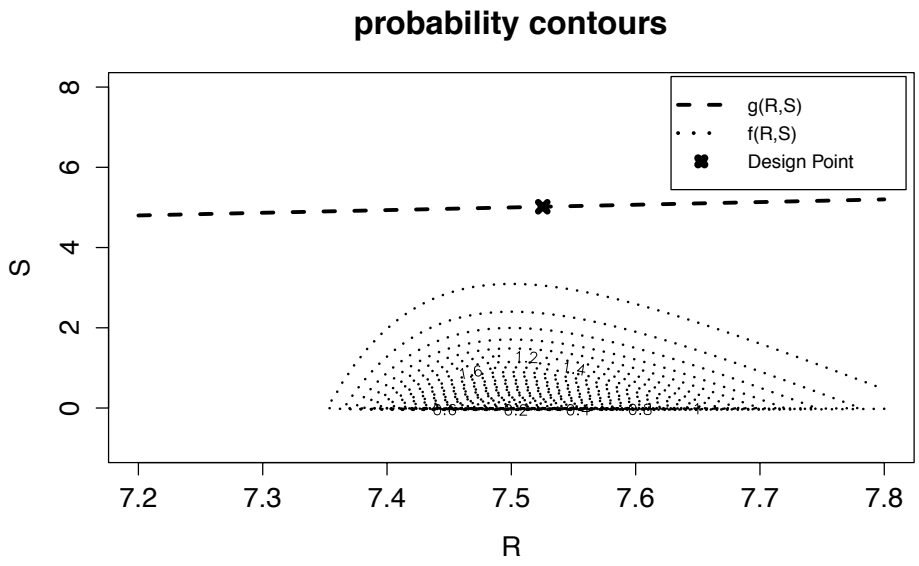
$$P_f = \mathbf{0.00654146}$$

$$\beta = -\Phi^{-1}(P_f) = \mathbf{2.4815043}$$



[b] FORM (**bolded** values are the ' * ' values)

	FORM						
Iteration	1	2	3	4	5	6	7
X1=R	7.548100	7.504770	7.527060	7.525720	7.525290	7.525280	**7.525280**
X2=S	1.000000	11.414690	5.993820	5.062610	5.016980	5.016850	**5.016850**
U1	0.177319	-0.282174	-0.036525	-0.050693	-0.055245	-0.055447	**-0.055444**
U2	0.337475	4.242906	2.807794	2.493250	2.477006	2.476959	**2.476959**
Alpha1	-0.066358	-0.013008	-0.020328	-0.022298	-0.022379	-0.022378	**-0.022378**
Alpha2	0.997796	0.999915	0.999793	0.999751	0.999750	0.999750	**0.999750**
Beta	0.324965	4.246220	2.807960	2.493760	2.477620	2.477580	**2.477580**
$P_f = \Phi(-Beta)$	**0.006614**						
Y[1,2]	-0.022378	0.999750					



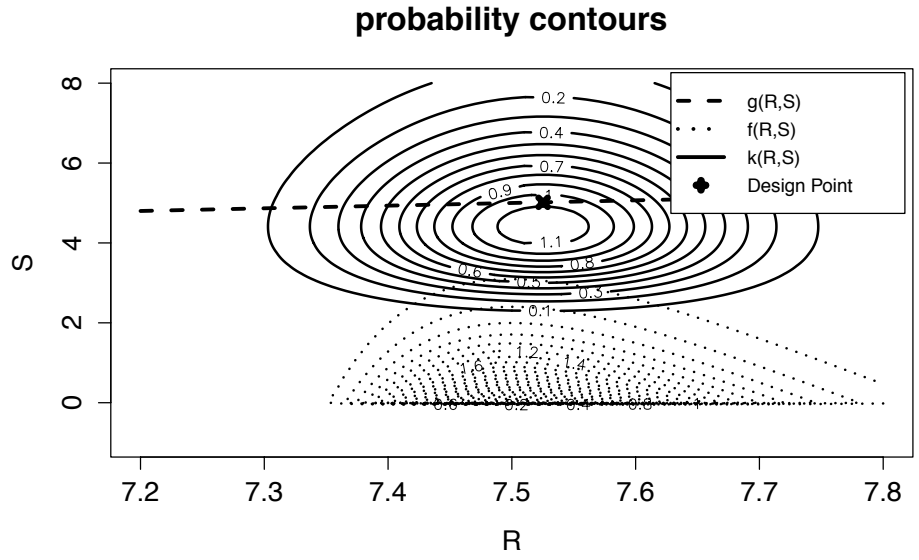
[c] Importance Sampling

The sampling distribution $k(S,R)$ is chosen such that S and R are independent, and thus are easily drawn from. The marginal sampling distribution for R is normal; that for S is lognormal, to ensure that negative values are not sampled, since S has [0, ∞) bound. In short, $k(S,R)$ has the **same support** as $f(S,R)$. Additionally, both marginal sampling distributions has their means chosen at the design point, and their standard deviations chosen by inspection, to ensure sufficient coverage. In particular, we want $k(S,R)$ to have **fatter tails** than $k(S,R)$, to prevent $f(s,r)/k(s,r)$ from ever blowing up to ∞. However, we don't want the tails to be too thick such that we are wasting simulations on parts of the original distribution that are relatively irrelevant to failure probability (i.e. far away from the limit state). This issue has largely been resolved by placing the sampling distribution closer to the design point.

In equation and graphical form:

$$S \sim lognormal(\mu = S^*, \sigma = 0.1)$$

$$R \sim N(\mu = R^*, \sigma = 0.3R^*)$$



The number of simulations is chosen to be $1e5$, which is about 1/6th of the estimated number of Crude Monte Carlo simulations to achieve 5% coefficient of variation (as was the target in HW5):

$$n = \frac{1 - P_f}{\delta^2 P_f} = \frac{1 - (0.00654146)}{0.05^2(0.00654146)} = 60748.5 \approx 6e5$$

The results show that we are better than 5% variation, and achieving so with only 1/6th of the crude counterpart:

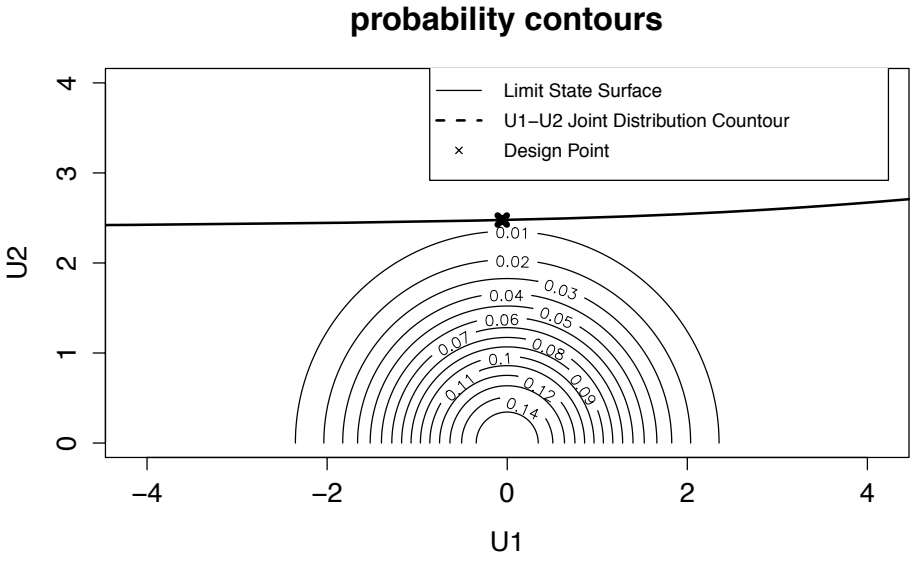
Seed	204	205	206	207	208	True Value (part a)
P_f	0.00653379	0.0065968	0.00654167	0.00648668	0.00657413	0.00654146
%Δ from True Value	-0.1173%	0.8460%	0.0032%	-0.8374%	0.4994%	/
Beta	2.481922	2.478500	2.481493	2.484500	2.479728	2.481504

[d] Comparisons

As expected, the values from parts (a), (b) and (c) are not the same, as they are different methods. Benchmarking against the true value (computed through numerical integration in part (a)), FORM gives less accurate results, than importance sampling with $1e5$ simulations. This observation is the same comparing to any of the 5 estimates. We note that while importance sampling approaches the true value asymptotically, FORM will always be a approximation, unless the g(X) is linear and X has a multivariate normal distribution. In our case, g(X) is linear, but X is jointly an exponential and an EV-Type-I. This results is a **convex limit state**, g(U)=0, **in the U space** away from the origin (shown below). Thus, by making a linear approximation to this limit state at the design point, FORM will provide a higher estimate on P_f and thus a lower estimate on beta, as shown in the results. SORM would have corrected some of this deficiency.

Part	**Beta**	%Δ	**P_f**	%Δ
[a] True Value	**2.481504**	/	**0.006541**	/
[b] FORM	**2.477580**	-0.1581%	**0.006614**	1.1064%
[c] Importance Sampling*	**2.481223**	-0.0113%	**0.006547**	0.0788%
[c] Importance Sampling (seed 204)	2.481922	0.0168%	0.00653379	-0.1173%
[c] Importance Sampling (seed 205)	2.478500	-0.1210%	0.0065968	0.8460%
[c] Importance Sampling (seed 206)	2.481493	-0.0005%	0.00654167	0.0032%
[c] Importance Sampling (seed 207)	2.484500	0.1207%	0.00648668	-0.8374%
[c] Importance Sampling (seed 208)	2.479728	-0.0716%	0.00657413	0.4994%

* This reliability index, beta, is computed from the average P_f of the fives estimates in part (c), which is 5*(1e5) draws.



```
################################################################################

# CEE 204 - 2014 Final
# Steven Wong | SUID: stywong
################################################################################


# setup ------------------------------------------------------------
rm(list=ls())
setwd("~/desktop/Machine_Learning/R"); source("library.r");
source("save_as_eps.r")
setwd("~/desktop/CEE 204 Final"); source("HL_RF_1.r")
options(digits=8)
set.seed(204)
par(mfrow=c(1,1))

################################################################################

# Problem 1
################################################################################


# variable joint distributions -------------------------------------------------
-----------

# actual distribution: joint-PDF in (X1, X2)
f_joint_compute = function(X,P,mu,sd,alpha,u,lambda) {
  PDF = matrix(rep(0,P),P,dim(X)[2])
  PDF[1,] = dgumbel(X[1,], location = u, scale = 1/alpha) # R
  PDF[2,] = dexp(X[2,],lambda) #S
  jpdf = PDF[1,]*PDF[2,]
  return(jpdf)
}

# sampling distribution: joint-PDF in (X1~normal, X2~lognormal)
k_joint_compute = function(X,P,mu,sd) {
  PDF = matrix(rep(0,P),P,dim(X)[2])
  PDF[1,] = dnorm(X[1,], mu[1], sd[1]) # sampling distribution for R
  PDF[2,] = dlnorm(X[2,], mu[2], sd[2]) # sampling distribution for S
  jpdf = PDF[1,]*PDF[2,]
  return(jpdf)
}

# function parameters ----------------------------------------------------------
-

# R
alpha = 12 # scale
u = 7.5 # location
euler = 0.5772
mu_R = u + euler/alpha
sd_R = pi/(alpha*sqrt(6))

# S
lambda = 1
mu_S = 1/lambda
sd_S = 1/lambda

# [a] numerical integration ----------------------------------------------------
---------
integrand = function(x) { exp(-exp(-12*(3/2*x-7.5))-x) }
integral = integrate(integrand, 0, Inf,abs.tol=1e-7)
pf_true = integral$value

# [b] FORM ---------------------------------------------------------------------
mu = matrix(c(mu_R, mu_S), 2, 1)
sd = matrix(c(sd_R, sd_S), 2, 1)
corr = matrix(c(1,0,0,1),2,2,byrow=TRUE)
p = dim(mu)[1]

# solve, return(list(X, U, a, b, beta, pf, gamma, designpts))
expK = 10
output = HL_RF_1(expK, mu, sd, corr, alpha, u, lambda)
designpts = output[[8]]
designptsU = output[[9]]

# [C] importance sampling ------------------------------------------------------
-------

(1-pf_true)/(0.05^2*pf_true)

set.seed(208)
n = 1e5

# sampling distribution for R: X1~normal
mu_X1 = designpts[1] # design point for R
sd_X1 = sd_R

# sampling distribution for S: X1~lognormal
mu_X2 = designpts[2] # design point for S
cv_X2 = 1
sd_log_X2 = sqrt(log(cv_X2^2 + 1))
mu_log_X2 = log(mu_X2) - 0.5*sd_log_X2^2

k_mu = matrix(c(mu_X1, mu_log_X2), 2, 1)
k_sd = matrix(c(sd_X1, sd_log_X2), 2, 1)

# simulation
sim_X1 = rnorm(n,k_mu[1],k_sd[1]) # R
sim_X2 = rlnorm(n,k_mu[2],k_sd[2]) # S
sim_G = 2*sim_X1 - 3*sim_X2
sim_X = cbind(sim_X1,sim_X2)
f_X1_X2 = f_joint_compute(t(sim_X),p,mu,sd,alpha,u,lambda)
k_X1_X2 = k_joint_compute(t(sim_X),p,k_mu,k_sd)

# pf
sum((sim_G <= 0)*f_X1_X2/k_X1_X2)/n

################################################################################
```

```
# Problem 1 Plots
################################################################################


# # contours plotting ------------------------------------------------------
-

# # computing the countour in (X1, X2)
# xlim = c(7.2, 7.8) # R
# ylim = c(-1, 8) # S
# x = as.matrix(seq(xlim[1],xlim[2], by=min(abs(xlim[2]-xlim[1])/abs(ylim[2]-
ylim[1]),0.01)))
# y = as.matrix(seq(ylim[1],ylim[2], by=min(abs(ylim[2]-ylim[1])/abs(xlim[2]-
xlim[1]),0.05)))
# x1 = matrix(rep(x,dim(y)[1]))
# x2 = matrix(matrix(rep(y,dim(x)[1]),dim(x)[1],dim(y)[1],byrow=TRUE),dim(x)
[1]*dim(y)[1])
# XX = cbind(x1,x2)
# z = t(matrix(f_joint_compute(t(XX),p,mu,sd,alpha,u,lambda),dim(y)[1],dim(x)
[1],byrow=TRUE))
# zk = t(matrix(k_joint_compute(t(XX),p,k_mu,k_sd),dim(y)[1],dim(x)
[1],byrow=TRUE))
#
# # plot
# contour(x = x, y = y, z = z,levels = pretty(range(z, finite = TRUE),20),
#         xlab="R",ylab="S", lwd=2, lty=3, main = "probability contours") #
f(X) countour
# contour(x = x, y = y, z = zk,levels = pretty(range(zk, finite = TRUE),10),
#         xlab="S",ylab="R",add=T, lwd=2, lty=1) # k(X) countour
# curve(2/3*x, from=xlim[1],to=xlim[2], add=TRUE,lwd=3,lty=2) # when g(X) = 0,
S = 2/3*R
# points(designpts[1],designpts[2], pch=4,lwd=5);# design point
# legend("topright",legend=c("g(R,S)","f(R,S)","k(R,S)","Design Point"),
#        pch=c(-1,-1,-1,3),lty=c(2,3,1,-1),lwd=c(3,3,3,6),cex=0.7, inset=0.01)
# legend("topright",legend=c("g(R,S)","f(R,S)","Design Point"),
#        pch=c(-1,-1,4),lty=c(2,3,-1),lwd=c(3,3,6),cex=0.7, inset=0.01)
# save_as_eps("1_contour_g_k.eps",5,8)
# save_as_eps("1_contour_g.eps",5,8)

# # contours plotting (U1, U2) space ---------------------------------------
----------------

# # actual distribution: joint-PDF in (U1, U2)
# f_U_joint_compute = function(X,P) {
#   PDF = matrix(rep(0,P),P,dim(X)[2])
#   PDF[1,] = dnorm(X[1,], 0, 1) # R
#   PDF[2,] = dnorm(X[2,], 0, 1) # S
#   pdf = PDF[1,]*PDF[2,]
#   return(pdf)
# }
# # CDF
# F_compute = function(X,P,mu,sd,alpha,u,lambda) {
#   CDF = matrix(rep(0,P),P,1)
#   CDF[1] = pgumbel(X[1], location = u, scale = 1/alpha) # R
#   CDF[2] = pexp(X[2],lambda) #S
#   return(CDF)
# }
#
# # getting the limit state surface
# xlim = c(0, 10)
# x = as.matrix(seq(xlim[1],xlim[2],0.1))
# y = 2/3*x
# X_sur = t(cbind(x,y))
# U_sur = X_sur
# L = t(chol(corr)) # corr = L%*%t(L)
# invL = solve(L)
# for (i in 1:dim(U_sur)[2]) {
#   U_sur[,i] = invL%*%qnorm(F_compute(X_sur[,i],p,mu,sd,alpha,u,lambda))
# }
#
# # computing the countour in (X1, X2)
# xlim = c(-4, 4)
# ylim = c(0, 4)
# x = as.matrix(seq(xlim[1],xlim[2],0.01))
# y = as.matrix(seq(ylim[1],ylim[2],0.01))
# x1 = matrix(rep(x,dim(y)[1]))
# x2 = matrix(matrix(rep(y,dim(x)[1]),dim(x)[1],dim(y)[1],byrow=TRUE),dim(x)
[1]*dim(y)[1])
# XX = cbind(x1,x2)
# z = t(matrix(f_U_joint_compute(t(XX),p),dim(y)[1],dim(x)[1],byrow=TRUE))
#
# # plot
# contour(x = x, y = y, z = z,levels = pretty(range(z, finite = TRUE),20),
#         xlab="U1",ylab="U2",asp=1, main = "probability contours") # f(X)
countour
# lines(U_sur[1,], U_sur[2,],lwd=2,lty=1) # when g(X) = 0
# points(designptsU[1],designptsU[2],pch=4,lwd=5);# design point
# legend("topright",legend=c("Limit State Surface","U1-U2 Joint Distribution
Countour","Design Point"),
#        pch=c(-1,-1,4),lty=c(1,2,-1),lwd=c(1,2,-1),cex=0.7, inset=0.01)
# save_as_eps("1_contour_U.eps",5,8)
```

```
################################################################################

# CEE 204 - 2014 Final
# Steven Wong | SUID: stywong
# ------------------------------------------------------------------------------
-
# HL-RF Algorithm
# ------------------------------------------------------------------------------
-
# inputs:
# mu = means (p by 1)
# sd = standard deviations (p by 1)
# expK = number of max iterations intended
# the rest = parameters for various distributions
################################################################################


HL_RF_1 = function(expK, mu, sd, corr, alpha, u, lambda){

  # function parameters ------------------------------------------------------
---
  p = dim(mu)[1]

  # correlation structure
  D = diag(1,p)*matrix(rep(sd,p),p,p) # standard deviations
  invD = solve(D)
  L = t(chol(corr)) # corr = L%*%t(L)
  invL = solve(L)

  # limit state functions ----------------------------------------------------
-----

  # G(X)
  gi_compute = function(X) {
    g = 2*X[1] - 3*X[2] # g(x) = 2R - 3S
    return(g)
  }

  # Delta_G(X)
  gi_d_compute = function(X,P) {
    g = matrix(rep(0,P),P,1)
    g[1] = 2
    g[2] = -3
    return(g)
  }

  # variable distributions ---------------------------------------------------
-------

  # PDF
  f_compute = function(X,P,mu,sd,alpha,u,lambda) {
    PDF = matrix(rep(0,P),P,1)
    PDF[1] = dgumbel(X[1], location = u, scale = 1/alpha) # R
    PDF[2] = dexp(X[2],lambda) #S
    return(PDF)
  }

  # CDF
  F_compute = function(X,P,mu,sd,alpha,u,lambda) {
    CDF = matrix(rep(0,P),P,1)
    CDF[1] = pgumbel(X[1], location = u, scale = 1/alpha) # R
    CDF[2] = pexp(X[2],lambda) #S
    return(CDF)
  }

  # Inverse CDF
  invF_compute = function(Z,P,mu,sd,alpha,u,lambda) {
    invCDF = matrix(rep(0,P),P,1)
    phi_Z = pnorm(Z)
    invCDF[1] = qgumbel(phi_Z[1],location = u, scale = 1/alpha) # R
    invCDF[2] = qexp(phi_Z[2],lambda) #S
    return(invCDF)
  }

  # HL-RF algorithm ----------------------------------------------------------
-----------
  k = 1 # starting count

  # tolerances
  err1_target = 1e-10
  err2_target = 1e-10
  err1 = 1 # initial value, to get the while loop going
  err2 = 1 # initial value, to get the while loop going

  # empty matrix setups
  X = matrix(rep(0,expK*p),p,expK)
  U = matrix(rep(0,expK*p),p,expK)
  Z = matrix(rep(0,expK*p),p,expK)
  a = matrix(rep(0,expK*p),p,expK)
  b = matrix(rep(1,expK*1),1,expK)
  hi = matrix(rep(0,expK*1),1,expK)

  # initial values
  X[,1] = matrix(c(mu[1],mu[2]),2,1)
  Z[,1] = qnorm(F_compute(X[,1],p,mu,sd,alpha,u,lambda)) # FORM
  U[,1] = invL%*%Z[,1]

  # the loop
  while ( (err1>err1_target)&&(err2>err2_target) ) {

    # jacobian for FORM (jacobian for HL is constant and thus outside the loop)
    JZX =
diag(as.vector(f_compute(X[,k],p,mu,sd,alpha,u,lambda)/as.matrix(dnorm(Z[,k]))))

    JUX = invL%*%JZX
    JXU = solve(JUX)

    # alpha
    gi_d = gi_d_compute(X[,k],p)
```

```
    hi_d = t(JXU)%*%gi_d
    hi_d_norm = as.numeric(sqrt(t(hi_d)%*%hi_d))
    a[,k] = -hi_d/hi_d_norm

    # beta
    b[,k] = t(a[,k])%*%U[,k]

    # updating U and X
    hi[k] = gi_compute(X[,k])
    U[,k+1] = a[,k]*(b[,k] + hi[k]/hi_d_norm)
    Z[,k+1] = L%*%U[,k+1] # FORM
    X[,k+1] = invF_compute(Z[,k+1],p,mu,sd,alpha,u,lambda) # FORM

    # updating error
    if (k != 1) {
      err1 = abs(b[,k]-b[,k-1])
      err2 = abs(hi[k])
    }
    k = k + 1
  }
  k = k-1 # steps to threshold reached

  # importance measure
  D = diag(sqrt(diag(JXU%*%t(JXU))),p)
  norm = as.numeric(sqrt(sum((D%*%t(JUX)%*%a[,k])^2)))
  gamma = D%*%t(JUX)%*%a[,k]/norm

  # return results
  beta = b[k]
  pf = pnorm(-beta)
  designpts = X[,k]
  designptsU = U[,k]
  return(list(X, U, a, b, beta, pf, gamma, designpts,designptsU))
}
```

[2] Vegas High Roller ----------

Please find R codes attached

[Setup]

D, α, T_h, T_c and L are mutually independent:

$$D \sim N(\mu = 0.8, \sigma = 0.1 \cdot 0.8 = 0.08)$$

$$\alpha \sim lognormal(\mu = 6 \cdot 10^{-6}, \sigma = 0.2 \cdot 6 \cdot 10^{-6} = 12 \cdot 10^{-7}) \sim lognormal(\mu_{ln} = -12.0434, \sigma_{ln} = 0.198042)$$

$$L \sim lognormal(\mu = 840, \sigma = 0.1 \cdot 840 = 84) \sim lognormal(\mu_{ln} = 6.72843, \sigma_{ln} = 0.0997513)$$

$$T_h \sim EV\ Type\ I\ (\mu = 106, \sigma = 0.2 \cdot 106 = 21.2) \sim EV\ Type\ I\ (\alpha = 0.0604976, u = 96.4591)$$

$$T_c \sim EV\ Type\ I\ (\mu = 69, \sigma = 0.2 \cdot 69 = 13.8) \sim EV\ Type\ I\ (\alpha = 0.0929384, u = 762.7894)$$

$$P_f = P[(g(x) = D - \alpha(T_h - T_c)L) < 0] = P[-\alpha T_h L + \alpha T_c L < 0]$$

[a] FORM

From the results below, we note that $\boldsymbol{T_h}$ **is the most important** for failure in g(X), and is considered as a load: it has the largest absolute gamma value, and is positive. The table also shows results for part (b).

			FORM
	g(X)	k(Xh)	k(Xc)
X1=D	**0.748865**	0.800000	0.800000
X2=α	**0.000007**	0.000006	0.000006
X3=T_h	**174.070000**	110.000000	102.517000
X4=T_c	**60.601300**	66.733000	72.000000
X5=L	**887.105000**	835.831000	835.831000
U1	**-0.639192**	0.000000	0.000000
U2	**1.184957**	0.000000	0.000000
U3	**2.361665**	0.367908	0.000000
U4	**-0.542876**	0.000000	0.395781
U5	**0.596848**	0.000000	0.000000
Alpha1	**-0.225410**	0.000000	0.000000
Alpha2	**0.417874**	0.000000	0.000000
Alpha3	**0.832838**	1.000000	0.000000
Alpha4	**-0.191445**	0.000000	-1.000000
Alpha5	**0.210478**	0.000000	0.000000
Beta	**2.835680**	**0.367908**	**-0.395781**
P_f = Φ(-Beta)	**0.002286**	**0.356471**	**0.653867**
g(X)'s ϒ[1,2,3,4,5]	-0.22541	0.417874	**0.832838**

Rzz	g(X)	k(Xh)	k(Xc)
g(X)	1.000000	0.832838	0.191445
k(Xh)	0.832838	1.000000	0.000000
k(Xc)	0.191445	0.000000	1.000000

-0.191445 0.210478

[b] Reliability Updating I

Our additional information is:

$$T_h > 110 \rightarrow k(x_h) = 110 - T_h < 0$$

$$T_c < 72 \rightarrow k(x_c) = T_h - 72 < 0$$

Our posterior failure probability is then:

$$P_f = P(g(x) \le 0 \mid k(x_h) < 0, k(x_c) < 0) = \frac{P(g(x) \le 0, (k(x_h) < 0,\ k(x_c) < 0))}{P(\ k(x_h) < 0, k(x_c) < 0)}$$

The results for each of the $P(g(x) \le 0\ )$, $P(k(x_h) < 0)$, and $P(\ k(x_c) < 0)$ are shown in the table in part (a). In order to combine them, we note that:

Since $[g(x) \le 0, k(x_h) < 0, k(x_c) < 0]$ is in parallel, $P[g(x) \le 0, k(x_h) < 0, k(x_c) < 0] = \Phi(-Beta, Rzz) = 0.00193034$
Since $[k(x_h) < 0, k(x_c) < 0]$ is in parallel, $P[k(x_h) < 0, k(x_c) < 0] = \Phi(-Beta, Rzz) = 0.233085$

**"Beta" and "Rzz" may be subsets of the Beta and Rzz in the table

Putting everything together:

$$P_f = \frac{0.00193034}{0.233085} = \mathbf{0.0082817}$$

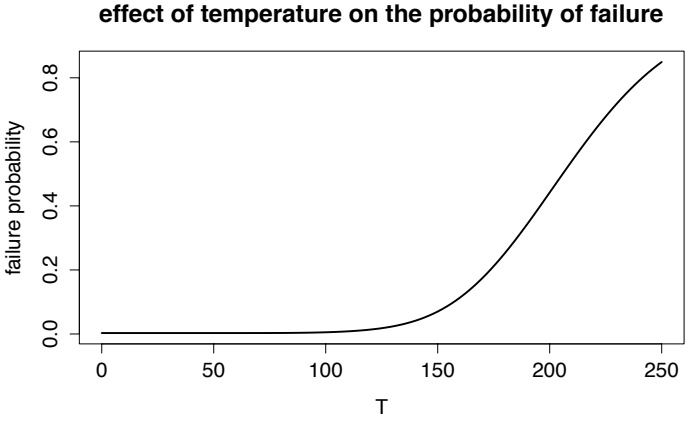
We double-checked with 1e7 Crude Monte Carlo simulations, and obtained $P_f = 0.0021417$ for part (a), and $P_f = 0.00801821$ for part (b). The results are very close to those by the FORM computation, and have discrepancies that can likely be attributed to the convexity of the limit state function in the U space, since FORM only linearly approximates the limit states in the U space (similar reasons as problem 1).

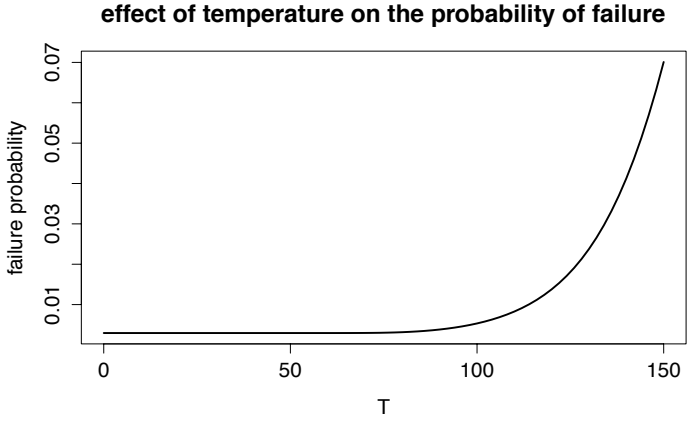
[c] Reliability Updating II

To plot the trend, we perform the same analysis as in part (b), except now exchanging '110' in $k(x_h)$ with the variable T:

$$k(x_h) = T - T_h < 0$$

We iterate the analysis a sufficient number of times to get a decent coverage on the T's range. The result is as follows (the left plot has a unearthly range for temperature, upward to 250°F; the right plot has a more reasonable bound on T):





```
################################################################################

# CEE 204 - 2014 Final
# Steven Wong | SUID: stywong
################################################################################


# setup ------------------------------------------------------------
rm(list=ls())
setwd("~/desktop/Machine_Learning/R"); source("library.r");
source("save_as_eps.r")
setwd("~/desktop/CEE 204 Final"); source("HL_RF_2.r")
options(digits=6)
set.seed(204)
par(mfrow=c(1,1))

################################################################################

# Problem 2
################################################################################


# function parameters ----------------------------------------------------------
-

# D ~ N
mu_D = 0.8
cv_D = 0.1
sd_D = cv_D*mu_D

# A ~ lognormal
mu_A = 6e-6
cv_A = 0.2
sd_A = cv_A*mu_A
sd_log_A = sqrt(log((sd_A/mu_A)^2 + 1))
mu_log_A = log(mu_A) - 0.5*sd_log_A^2

# TH ~ EV-Type-1
mu_TH = 106
cv_TH = 0.2
sd_TH = cv_TH*mu_TH
alpha_TH = pi/(sqrt(6)*sd_TH) # scale
u_TH = mu_TH - 0.5772/alpha_TH # location

# TC ~ EV-Type-1
mu_TC = 69
cv_TC = 0.2
sd_TC = cv_TC*mu_TC
alpha_TC = pi/(sqrt(6)*sd_TC) # scale
u_TC = mu_TC - 0.5772/alpha_TC # location

# L ~ lognormal
mu_L = 840
cv_L = 0.1
sd_L = cv_L*mu_L
sd_log_L = sqrt(log((sd_L/mu_L)^2 + 1))
mu_log_L = log(mu_L) - 0.5*sd_log_L^2

# [a] FORM ---------------------------------------------------------------------
mu = matrix(c(mu_D, mu_log_A, mu_TH, mu_TC, mu_log_L), 5, 1)
sd = matrix(c(sd_D, sd_log_A, sd_TH, sd_TC, sd_log_L), 5, 1)
corr = matrix(c(1,0,0,0,0,
                0,1,0,0,0,
                0,0,1,0,0,
                0,0,0,1,0,
                0,0,0,0,1),5,5,byrow=TRUE)
p = dim(mu)[1]

# solve, return(list(X, U, a, b, beta, alpha, pf, gamma, designpts,
designptsU))
expK = 20
Th_L = NaN
Tc_U = NaN

CASE = 1 # P(g(x) < 0)
output_a = HL_RF_2(expK, mu, sd, corr, alpha_TH, alpha_TC, u_TH, u_TC, CASE,
Th_L, Tc_U)
alpha_1 = output_a[[6]]
beta_1 = output_a[[5]]
pf_FORM = output_a[[7]]

# [b] reliability updating I ---------------------------------------------------
----------

# solve, return(list(X, U, a, b, beta, alpha, pf, gamma, designpts,
designptsU))
expK = 10
Th_L = 110
Tc_U = 72

CASE = 2 # P(k(xh) < 0)
output_b = HL_RF_2(expK, mu, sd, corr, alpha_TH, alpha_TC, u_TH, u_TC, CASE,
Th_L, Tc_U)
alpha_2 = output_b[[6]]
beta_2 = output_b[[5]]

CASE = 3 # P(k(xc) < 0)
output_c = HL_RF_2(expK, mu, sd, corr, alpha_TH, alpha_TC, u_TH, u_TC, CASE,
Th_L, Tc_U)
alpha_3 = output_c[[6]]
beta_3 = output_c[[5]]

# putting all the cases together
BETAS = rbind(beta_1,beta_2,beta_3)
ALPHA = rbind(alpha_1,alpha_2,alpha_3)
Rzz = ALPHA%*%t(ALPHA)

# P(g(x) < 0, k(xh) < 0, k(xc) < 0) => case 1, 2 & 3 => parallel
submatrix = c(1:3)
```

```
rzz = Rzz[submatrix,submatrix]
betas = BETAS[submatrix]
zeros = rep(0,length(submatrix))
pf_1  = pmnorm(-betas,zeros,rzz)

# P(k(xh) < 0, k(xc) < 0) => case 2 & 3 => parallel
submatrix = c(2:3)
rzz = Rzz[submatrix,submatrix]
betas = BETAS[submatrix]
zeros = rep(0,length(submatrix))
pf_2  = pmnorm(-betas,zeros,rzz)

pf_UPDATED = pf_1/pf_2

# [c] reliability updating II ------------------------------------------------
-----------

# computing the failure probability for a range of T's
expK = 50
CASE = 1
Ts = seq(0,150,by=2) # all T's considered
Tc_U = 72
p_f_all = rep(0,length(Ts))
for (i in 1:length(Ts)){

  CASE = 1 # P(g(x) < 0)
  output_a = HL_RF_2(expK, mu, sd, corr, alpha_TH, alpha_TC, u_TH, u_TC, CASE,
Ts[i], Tc_U)
  alpha_1 = output_a[[6]]
  beta_1 = output_a[[5]]
  pf_FORM = output_a[[7]]

  CASE = 2 # P(k(xh) < 0)
  output_b = HL_RF_2(expK, mu, sd, corr, alpha_TH, alpha_TC, u_TH, u_TC, CASE,
Ts[i], Tc_U)
  alpha_2 = output_b[[6]]
  beta_2 = output_b[[5]]

  CASE = 3 # P(k(xc) < 0)
  output_c = HL_RF_2(expK, mu, sd, corr, alpha_TH, alpha_TC, u_TH, u_TC, CASE,
Ts[i], Tc_U)
  alpha_3 = output_c[[6]]
  beta_3 = output_c[[5]]

  # putting all the cases together
  BETAS = rbind(beta_1,beta_2,beta_3)
  ALPHA = rbind(alpha_1,alpha_2,alpha_3)
  Rzz = ALPHA%*%t(ALPHA)

  # P(g(x) < 0, k(xh) < 0, k(xc) < 0) => case 1, 2 & 3 => parallel
  submatrix = c(1:3)
  rzz = Rzz[submatrix,submatrix]
  betas = BETAS[submatrix]
  zeros = rep(0,length(submatrix))
  pf_1  = pmnorm(-betas,zeros,rzz)

  # P(k(xh) < 0, k(xc) < 0) => case 2 & 3 => parallel
  submatrix = c(2:3)
  rzz = Rzz[submatrix,submatrix]
  betas = BETAS[submatrix]
  zeros = rep(0,length(submatrix))
  pf_2  = pmnorm(-betas,zeros,rzz)

  p_f_all[i] = pf_1/pf_2
}

# plot
plot(Ts,p_f_all,type="l",lwd=2, xlab="T", ylab="failure probability",
     main = "effect of temperature on the probability of failure")
save_as_eps("2_Ts_1.eps",5,8)

# check: simulation ----------------------------------------------------------
-
# (1-pf_FORM)/(0.05^2*pf_FORM)
#
# set.seed(204)
# n = 1e7
#
# # mu = matrix(c(mu_D, mu_log_A, mu_TH, mu_TC, mu_log_L), 5, 1)
# # simulation
# sim_D = rnorm(n,mu[1],sd[1])
# sim_A = rlnorm(n,mu[2],sd[2])
# sim_TH = rgumbel(n,location = u_TH, scale = 1/alpha_TH)
# sim_TC = rgumbel(n,location = u_TC, scale = 1/alpha_TC)
# sim_L = rlnorm(n,mu[5],sd[5])
#
# # pf
# sim_G = sim_D - sim_A*(sim_TH - sim_TC)*sim_L
# sim_KTH = Th_L - sim_TH
# sim_KTC = sim_TC - Tc_U
# sim_1 = (sim_G <= 0)
# sim_2 = (sim_KTH <= 0)
# sim_3 = (sim_KTC <= 0)
# # part (a): 0.0012352 with 1e7, seed 204
# sum(sim_1)/n
# # part (b): 0.00463571 with 1e7, seed 204
# f_t = sum(sim_1*sim_2*sim_3)/n
# f_b = sum(sim_2*sim_3)/n
# f_t/f_b
```

```
##############################################################################

# CEE 204 - 2014 Final
# Steven Wong | SUID: stywong
# ----------------------------------------------------------------------------
-
# HL-RF Algorithm
# ----------------------------------------------------------------------------
-
# inputs:
# mu = means (p by 1)
# sd = standard deviations (p by 1)
# expK = number of max iterations intended
# the rest = parameters for various distributions
##############################################################################


HL_RF_2 = function(expK, mu, sd, corr, alpha_TH, alpha_TC, u_TH, u_TC, CASE,
Th_L, Tc_U){

  # function parameters -------------------------------------------------------
---
  p = dim(mu)[1] # mu = matrix(c(mu_D, mu_log_A, mu_TH, mu_TC, mu_log_L), 5, 1)

  # correlation structure
  D = diag(1,p)*matrix(rep(sd,p),p,p) # standard deviations
  invD = solve(D)
  L = t(chol(corr)) # corr = L%*%t(L)
  invL = solve(L)

  # limit state functions -----------------------------------------------------
-----

  # G(X)
  gi_compute = function(X,Th_L,Tc_U) {
    if (CASE == 1) {
      g = X[1] - X[2]*(X[3]-X[4])*X[5]
    } else if (CASE == 2) {
      g = Th_L - X[3]
    } else if (CASE == 3) {
      g = X[4] - Tc_U
    }
    return(g)
  }

  # Delta_G(X)
  gi_d_compute = function(X,P) {
    g = matrix(rep(0,P),P,1)
    if (CASE == 1) {
      g[1] = 1
      g[2] = -(X[3]-X[4])*X[5]
      g[3] = -X[2]*X[5]
      g[4] = X[2]*X[5]
      g[5] = -X[2]*(X[3]-X[4])
    } else if (CASE == 2) {
      g[3] = -1
    } else if (CASE == 3) {
      g[4] = 1
    }
    return(g)
  }

  # variable distributions ----------------------------------------------------
-------
  # mu = matrix(c(mu_D, mu_log_A, mu_TH, mu_TC, mu_log_L), 5, 1)

  # PDF
  f_compute = function(X,P,mu,sd,alpha_TH, alpha_TC, u_TH, u_TC, CASE, Th_L,
Tc_U) {
    PDF = matrix(rep(0,P),P,1)
    PDF[1] = dnorm(X[1], mu[1], sd[1])
    PDF[2] = dlnorm(X[2], mu[2], sd[2])
    PDF[3] = dgumbel(X[3], location = u_TH, scale = 1/alpha_TH)
    PDF[4] = dgumbel(X[4], location = u_TC, scale = 1/alpha_TC)
    PDF[5] = dlnorm(X[5], mu[5], sd[5])
    return(PDF)
  }

  # CDF
  F_compute = function(X,P,mu,sd,alpha_TH, alpha_TC, u_TH, u_TC, CASE, Th_L,
Tc_U) {
    CDF = matrix(rep(0,P),P,1)
    CDF[1] = pnorm(X[1], mu[1], sd[1])
    CDF[2] = plnorm(X[2], mu[2], sd[2])
    CDF[3] = pgumbel(X[3], location = u_TH, scale = 1/alpha_TH)
    CDF[4] = pgumbel(X[4], location = u_TC, scale = 1/alpha_TC)
    CDF[5] = plnorm(X[5], mu[5], sd[5])
    return(CDF)
  }

  # Inverse CDF
  invF_compute = function(Z,P,mu,sd,alpha_TH, alpha_TC, u_TH, u_TC, CASE, Th_L,
Tc_U) {
    invCDF = matrix(rep(0,P),P,1)
    phi_Z = pnorm(Z)
    invCDF[1] = qnorm(phi_Z[1], mu[1], sd[1])
    invCDF[2] = qlnorm(phi_Z[2], mu[2], sd[2])
    invCDF[3] = qgumbel(phi_Z[3], location = u_TH, scale = 1/alpha_TH)
    invCDF[4] = qgumbel(phi_Z[4], location = u_TC, scale = 1/alpha_TC)
    invCDF[5] = qlnorm(phi_Z[5], mu[5], sd[5])
    return(invCDF)
  }

  # HL-RF algorithm -----------------------------------------------------------
-----------
  k = 1 # starting count

  # tolerances
  err1_target = 1e-10
```

```
  err2_target = 1e-10
  err1 = 1 # initial value, to get the while loop going
  err2 = 1 # initial value, to get the while loop going

  # empty matrix setups
  X = matrix(rep(0,expK*p),p,expK)
  U = matrix(rep(0,expK*p),p,expK)
  Z = matrix(rep(0,expK*p),p,expK)
  a = matrix(rep(0,expK*p),p,expK)
  b = matrix(rep(1,expK*1),1,expK)
  hi = matrix(rep(0,expK*1),1,expK)

  # initial values
  mu_adj =
rbind(mu[1],exp(mu[2]+0.5*sd[2]^2),mu[3],mu[4],exp(mu[5]+0.5*sd[5]^2))
  X[,1] = mu_adj
  Z[,1] = qnorm(F_compute(X[,1],p,mu,sd,alpha_TH, alpha_TC, u_TH, u_TC, CASE,
Th_L, Tc_U)) # FORM
  U[,1] = invL%*%Z[,1]

  # the loop
  while ( (err1>err1_target)&&(err2>err2_target) ) {

    # jacobian for FORM (jacobian for HL is constant and thus outside the loop)
    JZX = diag(as.vector(f_compute(X[,k],p,mu,sd,alpha_TH, alpha_TC, u_TH,
u_TC, CASE, Th_L, Tc_U)/as.matrix(dnorm(Z[,k]))))
    JUX = invL%*%JZX
    JXU = solve(JUX)

    # alpha
    gi_d = gi_d_compute(X[,k],p)
    hi_d = t(JXU)%*%gi_d
    hi_d_norm = as.numeric(sqrt(t(hi_d)%*%hi_d))
    a[,k] = -hi_d/hi_d_norm

    # beta
    b[,k] = t(a[,k])%*%U[,k]

    # updating U and X
    hi[k] = gi_compute(X[,k],Th_L,Tc_U)
    U[,k+1] = a[,k]*(b[,k] + hi[k]/hi_d_norm)
    Z[,k+1] = L%*%U[,k+1] # FORM
    X[,k+1] = invF_compute(Z[,k+1],p,mu,sd,alpha_TH, alpha_TC, u_TH, u_TC,
CASE, Th_L, Tc_U) # FORM

    # updating error
    if (k != 1) {
      err1 = abs(b[,k]-b[,k-1])
      err2 = abs(hi[k])
    }
    k = k + 1
  }
  k = k-1 # steps to threshold reached

  # importance measure
  D = diag(sqrt(diag(JXU%*%t(JXU))),p)
  norm = as.numeric(sqrt(sum((D%*%t(JUX)%*%a[,k])^2)))
  gamma = D%*%t(JUX)%*%a[,k]/norm

  # return results
  beta = b[k]
  alpha = a[,k]
  pf = pnorm(-beta)
  designpts = X[,k]
  designptsU = U[,k]
  return(list(X, U, a, b, beta, alpha, pf, gamma, designpts,designptsU))
}
```

[3] Highway Network ----------

Please find R codes attached

[Setup]

NOTE: for this problem, I am using a truncated normal distribution on T with a lower bound of zero, instead of a normal distribution. The change should not alter the following results greatly, since the T's mean sufficiently far enough from zero, relative to its spread. Nonetheless, I made the choice for two main reasons:

- Having nonzero values for T results in $\sqrt{T}$ having imaginary values, and consequently invalidated a number of Monte Carlo draws.
- Toll cost in practice should be greater than zero.

C, T and the D's are mutually independent, but there is a correlation structure amongst all the D values:

$$C \sim lognormal(\mu = 840, \sigma = 0.1 \cdot 840 = 84) \sim lognormal(\mu_{ln} = 6.72843, \sigma_{ln} = 0.0997513)$$

$$\begin{bmatrix} D_{101} \\ D_{80} \\ D_{880} \end{bmatrix} \sim lognormal(\mu = \begin{bmatrix} 73 \\ 73 \\ 73 \end{bmatrix}, \Sigma = (0.3^2) \begin{bmatrix} 73 & 0 & 0 \\ 0 & 73 & 0 \\ 0 & 0 & 73 \end{bmatrix} \begin{bmatrix} 1 & 0.5 & 0.5 \\ 0.5 & 1 & 0.5 \\ 0.5 & 0.5 & 1 \end{bmatrix} \begin{bmatrix} 73 & 0 & 0 \\ 0 & 73 & 0 \\ 0 & 0 & 73 \end{bmatrix})$$

$$T \sim TN(\mu = ?, \sigma = 0.3 \cdot ? = 84, lower\ bound\ = 0)$$

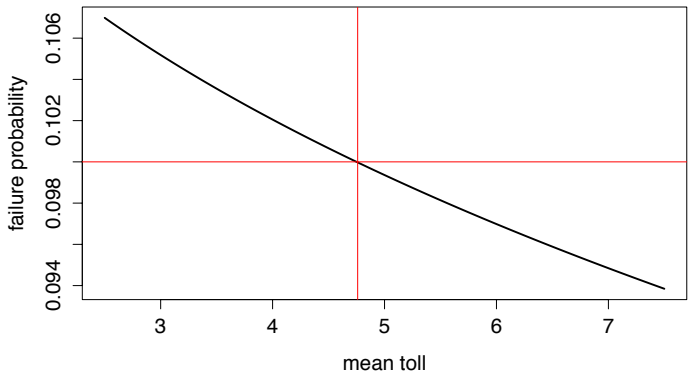
$$P_f = P[(g(x) = C - D_{101} < 0]$$

$$P_f = P[(k(x) = C - D_{80} + 2\sqrt{T} < 0]$$

$$P_f = P[(h(x) = C - D_{880} < 0]$$

[a] Optimal Mean Toll

From iterating FORM computations, we found the minimum mean toll to be **$4.76**, in order to bring the probability of traffic jam for I-80 to a value just below 10% (the probability of traffic jam for I-80 is 9.9974%, for $E[T]$ = $4.76). From the results below (in part b), we note that **D_{80} is more important** for traffic jam on I-80 than toll, and is also considered as a load: it has the largest absolute gamma value, and is positive. The table also shows results for part (b).



[b] FORM: System Failure

	FORM		
	g(X)	h(X)	k(X)
X1=C	**95.935270**	**95.648850**	**95.935270**
X2=D_{101}	**95.935270**	**83.616800**	**81.901900**
X3=D_{80}	**81.901900**	**99.994810**	**81.901900**

$X4=D_{880}$	**81.901900**	**83.616800**	**95.935270**				
X5=T	**4.760770**	**4.721840**	**4.760770**				
U1	**-0.366123**	**-0.396098**	**-0.366123**				
U2	**1.077472**	**0.609326**	**0.538736**				
U3	**0.000000**	**1.055383**	**0.311039**				
U4	**0.000000**	**0.000000**	**0.879752**				
U5	**0.000000**	**-0.027270**	**0.000000**				
Alpha1	**-0.321732**	**-0.309042**	**-0.321732**	Rzz	g(X)	h(X)	k(X)
Alpha2	**0.946831**	**0.475405**	**0.473415**	g(X)	1.000000	0.549557	0.551756
Alpha3	**0.000000**	**0.823426**	**0.273327**	h(X)	0.549557	1.000000	0.549557
Alpha4	**0.000000**	**0.000000**	**0.773084**	k(X)	0.551756	0.549557	1.000000
Alpha5	**0.000000**	**-0.021276**	**0.000000**				
Beta	**1.137980**	**1.281700**	**1.137980**				
$P_f = \Phi(-Beta)$	**0.127564**	**0.099974**	**0.127564**				
h(X)'s ϒ[1,2,3,4,5]	-0.309042	0.000000	**0.950810**	0.000000	-0.021276		

The probability of the entire system failing is the probability that 101 and (80 or 880) have a traffic jam:

$$P_f = P(g(x) < 0\,, [h(x) < 0 \cup\ k(x) < 0])$$
$$= P(g(x) < 0\,, h(x) < 0) + P(g(x) < 0\,, k(x) < 0) - P(g(x) < 0\,, h(x) < 0, k(x) < 0)$$

The results for each of the $P(g(x) \leq 0\ )$, $P(h(x) < 0)$, and $P(\ k(x) < 0)$ are shown in the table above. In order to combine them, we note that:

Since $[g(x) < 0\,, h(x) < 0]$ is in parallel, $P[g(x) < 0\,, h(x) < 0] = \Phi(-Beta, Rzz) = \ 0.041887$
Since $[g(x) < 0\,, k(x) < 0]$ is in parallel, $P[g(x) < 0\,, k(x) < 0] = \Phi(-Beta, Rzz) = 0.0499379$
Since $[g(x) < 0\,, h(x) < 0, k(x) < 0]$ is in parallel, $P[g(x) < 0\,, h(x) < 0, k(x) < 0] = \Phi(-Beta, Rzz) = 0.0239108$

**"Beta" and "Rzz" may be subsets of the Beta and Rzz in the table

Putting everything together:

$$P_f = \ 0.041887 + 0.0499379 - 0.0239108 = \mathbf{0.067914}$$

[d] Monte Carlo: System Failure

The find the probability for a parallel system understand some criteria, we run Monte Carlo for each of the individual components, count the number of instances when all constituent components satisfy their respective criteria, and divide that number by the number of simulations. Aiming for 0.2% coefficient of variation, the number of simulations is chosen to be $1e6$, rounding up:

$$n = \frac{1 - P_f}{\delta^2 P_f} = \frac{1 - (0.067914)}{0.002^2(0.067914)} = 548980 \approx 5e5 \rightarrow 1e6$$

Seed	204	205	206	207	208	FORM Value (part b)
P_f	0.068043	0.067838	0.067585	0.068457	0.067943	0.067914
%Δ from 204	/	-0.3013%	-0.6731%	0.6084%	-0.1470%	-0.1896%
%Δ from FORM	0.1899%	-0.1119%	-0.4844%	0.7995%	0.0427%	/

The 5 estimates are within 1% from the FORM result, and are relatively stable, around our coefficient of variation target. They are not unequivocally larger or smaller than the FORM result, and thus it is difficult to comment on their differences. We do know that FORM's results are not the true value, being which would require the limit state function to be linear and the X having a multivariate normal distribution. We have neither in this problem. On the other hand, Monte Carlo would asymptotically approach the true value.

```
################################################################################

# CEE 204 - 2014 Final
# Steven Wong | SUID: stywong
################################################################################


# setup ------------------------------------------------------------
rm(list=ls())
setwd("~/desktop/Machine_Learning/R"); source("library.r");
source("save_as_eps.r")
setwd("~/desktop/CEE 204 Final"); source("HL_RF_3.r")
options(digits=6)
set.seed(204)
par(mfrow=c(1,1))

################################################################################

# Problem 3
################################################################################


# function parameters ----------------------------------------------------------
-

# C ~ lognormal
mu_C = 100
cv_C = 0.1
sd_C = cv_C*mu_C
sd_log_C = sqrt(log((sd_C/mu_C)^2 + 1))
mu_log_C = log(mu_C) - 0.5*sd_log_C^2
# pnorm(log(110),mu_log_C,sd_log_C) = plnorm(log110,mu_log_C,sd_log_C)

# D101, D80, D880 ~ jointly-lognormal
mu_D = 73
cv_D = 0.3
sd_D = cv_D*mu_D
sd_log_D = sqrt(log((sd_D/mu_D)^2 + 1))
mu_log_D = log(mu_D) - 0.5*sd_log_D^2

# T ~ N
mu_T = NaN
cv_T = 0.3
sd_T = cv_T*mu_T

# [a] FORM: I-80 ---------------------------------------------------------------

corr = matrix(c(1.0,0.0,0.0,0.0,0.0,
                0.0,1.0,0.5,0.5,0.0,
                0.0,0.5,1.0,0.5,0.0,
                0.0,0.5,0.5,1.0,0.0,
                0.0,0.0,0.0,0.0,1.0),5,5,byrow=TRUE)
p = dim(sd)[1]

# solve, return(list(X, U, a, b, beta, alpha, pf, gamma, designpts,
designptsU))
# mu_T_tests = seq(2.5,7.5,0.01)
# expK = 50
# CASE = 2 # P(h(x) < 0)
# pf_tests = rep(0,length(mu_T_tests))
# for (i in 1:length(mu_T_tests)) {
#   mu = matrix(c(mu_log_C, mu_log_D, mu_log_D, mu_log_D, mu_T_tests[i]), 5, 1)
#   sd = matrix(c(sd_log_C, sd_log_D, sd_log_D, sd_log_D, cv_T*mu_T_tests[i]),
5, 1)
#   output_a_all = HL_RF_3(expK, mu, sd, corr, CASE)
#   pf_tests[i] = output_a_all[[7]]
# }
#
# target = 0.1
# max(pf_tests[pf_tests < target])
# mu_T_opt = mu_T_tests[pf_tests==max(pf_tests[pf_tests < target])]
# mu_T_opt
#
# plot(mu_T_tests,pf_tests,type="l",lwd="2",xlab="mean toll", ylab="failure
probability")
# abline(h=0.1,col="red")
# abline(v=mu_T_opt,col="red")
# save_as_eps("3_mu_T.eps",5,8)

# [b] FORM: System -------------------------------------------------------------

# T ~ N
# mu_T = mu_T_opt # from part [a]
mu_T = 4.76 # from part [a]
cv_T = 0.3
sd_T = cv_T*mu_T

mu = matrix(c(mu_log_C, mu_log_D, mu_log_D, mu_log_D, mu_T), 5, 1)
sd = matrix(c(sd_log_C, sd_log_D, sd_log_D, sd_log_D, sd_T), 5, 1)

CASE = 1 # P(g(x) < 0)
output_a = HL_RF_3(expK, mu, sd, corr, CASE)
alpha_1 = output_a[[6]]
beta_1 = output_a[[5]]

CASE = 2 # P(h(x) < 0)
output_b = HL_RF_3(expK, mu, sd, corr, CASE)
alpha_2 = output_b[[6]]
beta_2 = output_b[[5]]
designptsCase2 = output_b[[9]]

CASE = 3 # P(k(x) < 0)
output_c = HL_RF_3(expK, mu, sd, corr, CASE)
alpha_3 = output_c[[6]]
beta_3 = output_c[[5]]

# putting all the cases together
BETAS = rbind(beta_1,beta_2,beta_3)
ALPHA = rbind(alpha_1,alpha_2,alpha_3)
```

```
Rzz = ALPHA%*%t(ALPHA)

# P(g(x) < 0, h(x) < 0, k(x) < 0) => case 1, 2 & 3 => parallel
submatrix = c(1:3)
rzz = Rzz[submatrix,submatrix]
betas = BETAS[submatrix]
zeros = rep(0,length(submatrix))
pf_1  = pmnorm(-betas,zeros,rzz)

# P(g(x) < 0, h(x) < 0) => case 1 & 2 => parallel
submatrix = c(1,2)
rzz = Rzz[submatrix,submatrix]
betas = BETAS[submatrix]
zeros = rep(0,length(submatrix))
pf_2  = pmnorm(-betas,zeros,rzz)

# P(g(x) < 0, k(x) < 0) => case 1 & 3 => parallel
submatrix = c(1,3)
rzz = Rzz[submatrix,submatrix]
betas = BETAS[submatrix]
zeros = rep(0,length(submatrix))
pf_3  = pmnorm(-betas,zeros,rzz)

pf_UPDATED = pf_2 + pf_3 - pf_1

# 0.0678

# [c] check: simulation ------------------------------------------------------
-----
(1-pf_1)/(0.01^2*pf_1)
(1-pf_2)/(0.01^2*pf_2)
(1-pf_3)/(0.01^2*pf_3)

(1-pf_UPDATED)/(0.001^2*pf_UPDATED)

# Crude Monte Carlo ----------------------------------------------------------
-
set.seed(208)
n = 1e6

# simulation
cov = diag(as.vector(sd))%*%corr%*%diag(as.vector(sd))
cov_D = cov[2:4,2:4]
m = mu_T_opt
sim_C = rlnorm(n,mu[1],sd[1])
sim_D = exp(rmnorm(n,mu[2:4],cov_D))
sim_T = rtruncnorm(n, a=0, b=Inf, m, m*cv_T)

sim_g = sim_C - sim_D[,1]
sim_h = sim_C - sim_D[,2] + 2*sqrt(sim_T)
sim_k = sim_C - sim_D[,3]

sim_1 = (sim_g <= 0)
sim_2 = (sim_h <= 0)
sim_3 = (sim_k <= 0)

# part [a]
sum(sim_2)/n

# part [b]
sum(sim_1*sim_2)/n + sum(sim_1*sim_3)/n -sum(sim_1*sim_2*sim_3)/n
```

```
##############################################################################

# CEE 204 - 2014 Final
# Steven Wong | SUID: stywong
# ----------------------------------------------------------------------------
-
# HL-RF Algorithm
# ----------------------------------------------------------------------------
-
# inputs:
# mu = means (p by 1)
# sd = standard deviations (p by 1)
# expK = number of max iterations intended
# the rest = parameters for various distributions
##############################################################################


HL_RF_3 = function(expK, mu, sd, corr, CASE){

  # function parameters -------------------------------------------------------
---
  p = dim(mu)[1] # mu = matrix(c(mu_D, mu_log_A, mu_TH, mu_TC, mu_log_L), 5, 1)

  # correlation structure
  D = diag(1,p)*matrix(rep(sd,p),p,p) # standard deviations
  invD = solve(D)
  L = t(chol(corr)) # corr = L%*%t(L)
  invL = solve(L)

  # limit state functions -----------------------------------------------------
-----

  # G(X)
  gi_compute = function(X,Th_L,Tc_U) {
    if (CASE == 1) {
      g = X[1] - X[2]
    } else if (CASE == 2) {
      g = X[1] - X[3] + 2*sqrt(X[5])
    } else if (CASE == 3) {
      g = X[1] - X[4]
    }
    return(g)
  }

  # Delta_G(X)
  gi_d_compute = function(X,P) {
    g = matrix(rep(0,P),P,1)
    if (CASE == 1) {
      g[1] = 1
      g[2] = -1
    } else if (CASE == 2) {
      g[1] = 1
      g[3] = -1
      g[5] = 1/sqrt(X[5])
    } else if (CASE == 3) {
      g[1] = 1
      g[4] = -1
    }
    return(g)
  }

  # variable distributions ----------------------------------------------------
-------
  # mu = matrix(c(mu_D, mu_log_A, mu_TH, mu_TC, mu_log_L), 5, 1)

  # PDF
  f_compute = function(X,P,mu,sd) {
    PDF = matrix(rep(0,P),P,1)
    PDF[1] = dlnorm(X[1], mu[1], sd[1])
    PDF[2] = dlnorm(X[2], mu[2], sd[2])
    PDF[3] = dlnorm(X[3], mu[3], sd[3])
    PDF[4] = dlnorm(X[4], mu[4], sd[4])
    PDF[5] = dtruncnorm(X[5], a=0, b=Inf, mu[5], sd[5])
    return(PDF)
  }

  # CDF
  F_compute = function(X,P,mu,sd) {
    CDF = matrix(rep(0,P),P,1)
    CDF[1] = plnorm(X[1], mu[1], sd[1])
    CDF[2] = plnorm(X[2], mu[2], sd[2])
    CDF[3] = plnorm(X[3], mu[3], sd[3])
    CDF[4] = plnorm(X[4], mu[4], sd[4])
    CDF[5] = ptruncnorm(X[5], a=0, b=Inf, mu[5], sd[5])
    return(CDF)
  }

  # Inverse CDF
  invF_compute = function(Z,P,mu,sd) {
    invCDF = matrix(rep(0,P),P,1)
    phi_Z = pnorm(Z)
    invCDF[1] = qlnorm(phi_Z[1], mu[1], sd[1])
    invCDF[2] = qlnorm(phi_Z[2], mu[2], sd[2])
    invCDF[3] = qlnorm(phi_Z[3], mu[3], sd[3])
    invCDF[4] = qlnorm(phi_Z[4], mu[4], sd[4])
    invCDF[5] = qtruncnorm(phi_Z[5], a=0, b=Inf, mu[5], sd[5])
    return(invCDF)
  }

  # HL-RF algorithm -----------------------------------------------------------
-----------
  k = 1 # starting count

  # tolerances
  err1_target = 1e-10
  err2_target = 1e-10
  err1 = 1 # initial value, to get the while loop going
  err2 = 1 # initial value, to get the while loop going
```

```
# empty matrix setups
X = matrix(rep(0,expK*p),p,expK)
U = matrix(rep(0,expK*p),p,expK)
Z = matrix(rep(0,expK*p),p,expK)
a = matrix(rep(0,expK*p),p,expK)
b = matrix(rep(1,expK*1),1,expK)
hi = matrix(rep(0,expK*1),1,expK)

# initial values
mu_adj = rbind(exp(mu[1]+0.5*sd[1]^2),
               exp(mu[2]+0.5*sd[2]^2),
               exp(mu[3]+0.5*sd[3]^2),
               exp(mu[4]+0.5*sd[4]^2),
               mu[5])
X[,1] = mu_adj
Z[,1] = qnorm(F_compute(X[,1],p,mu,sd)) # FORM
U[,1] = invL%*%Z[,1]

# the loop
while ( (err1>err1_target)&&(err2>err2_target) ) {

  # jacobian for FORM (jacobian for HL is constant and thus outside the loop)
  JZX = diag(as.vector(f_compute(X[,k],p,mu,sd)/as.matrix(dnorm(Z[,k]))))
  JUX = invL%*%JZX
  JXU = solve(JUX)

  # alpha
  gi_d = gi_d_compute(X[,k],p)
  hi_d = t(JXU)%*%gi_d
  hi_d_norm = as.numeric(sqrt(t(hi_d)%*%hi_d))
  a[,k] = -hi_d/hi_d_norm

  # beta
  b[,k] = t(a[,k])%*%U[,k]

  # updating U and X
  hi[k] = gi_compute(X[,k],Th_L,Tc_U)
  U[,k+1] = a[,k]*(b[,k] + hi[k]/hi_d_norm)
  Z[,k+1] = L%*%U[,k+1] # FORM
  X[,k+1] = invF_compute(Z[,k+1],p,mu,sd) # FORM

  # updating error
  if (k != 1) {
    err1 = abs(b[,k]-b[,k-1])
    err2 = abs(hi[k])
  }
  k = k + 1
}
k = k-1 # steps to threshold reached

# importance measure
D = diag(sqrt(diag(JXU%*%t(JXU))),p)
norm = as.numeric(sqrt(sum((D%*%t(JUX)%*%a[,k])^2)))
gamma = D%*%t(JUX)%*%a[,k]/norm

  # return results
  beta = b[k]
  alpha = a[,k]
  pf = pnorm(-beta)
  designpts = X[,k]
  designptsU = U[,k]
  return(list(X, U, a, b, beta, alpha, pf, gamma, designpts,designptsU))
}
```