Solutions

immediate

Question 1

For $x \geq \alpha$, the cdf is given by,

$$\begin{aligned} F(x) &= \int_{-\infty}^{x} f(y)dy \\ &= \int_{\alpha}^{x} \beta\alpha^{\beta} y^{-(\beta+1)} dy \\ &= \alpha^{\beta} \left[-y^{-\beta}\right]_{\alpha}^{x} \\ &= 1 - \left(\frac{\alpha}{x}\right)^{\beta}. \end{aligned}$$

Thus the cdf is given by,

$$F(x) = \begin{cases} 0 & x < \alpha \\ 1 - \left(\frac{\alpha}{x}\right)^{\beta} & x \geq \alpha. \end{cases}$$

Now, find the inverse. Let

$$\begin{aligned} u &= 1 - \left(\frac{\alpha}{x}\right)^{\beta} \\ \Rightarrow \quad x &= \frac{\alpha}{(1-u)^{1/\beta}}. \end{aligned}$$

However, note that for $U \sim U[0,1]$, then $(1-U) \sim U[0,1]$. So that for $0 \leq u \leq 1$, $F^{-1}(u) = \alpha u^{-1/\beta}$. So in R:

```
rpareto <- function(n,alpha,beta) {
  u <- runif(n,0,1)
  return( alpha*u^(-1/beta))
}
```

Question 2

```
rexpInversion <- function(n,lambda) {
  u <- runif(n,0,1)
  x <- -(1/lambda) * log(1-u)
  return(x)
}


rexpRejection <- function(n,lambda) {
  samples <- numeric(n)
```

```
  numSoFar <- 0
  count <- 0
  for (i in 1:n) {
    while(TRUE) {
      count <- count+1
      x <- runif(1,0,5)
      u <- runif(1,0,1)
      if (u < dexp(x,lambda)/10*dunif(x,0,5)) {
        numSoFar <- numSoFar + 1
        samples[numSoFar] <- x
        break
      }
    }
  }
  print(n/count) #acceptance rate
  return(samples)
}

y1 <-rexpInversion(1000,2)
y2 <-rexpRejection(1000,2)
y3 <-rexp(1000,2)

plot(density(y1))
lines(density(y2),col='red')
lines(density(y3),col='blue')

ks.test(y1,y3)
ks.test(y2,y3)
```

The p-values will probably be above 0.05, since all the samples have an Exponential(2) distribution. However recall from last week that even when the null hypothesis is correct (i.e. when all samples have the same distribution) there is a 5% chance of making a Type I error and incorrectly rejecting the null when it is correct. So its possible that you will be 'unlucky', and your test will claim the distributions are different even if your code is correct. If this happens, try generating 1,000 different samples and doing the test again.

Question 3

The inverse distribution function is obtained by solving $F_X(x;\lambda,\gamma) = u$ with respect to u. This gives

$$F_X^{-1}(u;\lambda,\gamma) = \lambda(-\log(1-u))^{1/\gamma}$$

The density function is

$$f_X(x;\lambda,\gamma) = \frac{\mathrm{d}F_X(x;\lambda,\gamma)}{\mathrm{d}x} = (\gamma/\lambda)(x/\lambda)^{\gamma-1}e^{-(x/\lambda)^\gamma}$$

The below code plots the density of the Weibull distribution for several values of the parameter γ and fixed $\lambda = 1$.

```
dens.weibull <- function(x,lambda,gamma)
{
  res <- (gamma/lambda)*((x/lambda)^(gamma-1))*exp(-(x/lambda)^(gamma))
  return(res)
```

```
}

x      <- seq(0,15,len=100)
lambda <- 1
gamma  <- seq(0.1,3,len=30)

##-----------------------------------------
## plot density for first value of gamma
## and then use lines command to superpose
## the density plots for the remaining values
## of gamma
##-----------------------------------------
plot(x,dens.weibull(x,lambda,gamma[1]),type="l",ylim=c(0,1.5))
for(i in 2:length(gamma))
{
  lines(x,dens.weibull(x,lambda,gamma[i]),lty=i)
}
```

To simulate, we have:

```
sim.weibull <- function(n,lambda,gamma)
{
   u <- runif(n)
   X <- lambda*(-log(1-u))^(1/gamma)
   return(X)
}
```

Question 4

Following the hint, we compute M by maximising

$$h(x) = \frac{\phi(x)}{g(x;\lambda)} = \begin{cases} \frac{2}{(2\pi)^{1/2}\lambda} e^{\lambda x - x^2/2} & x \geq 0 \\ \\ \frac{2}{(2\pi)^{1/2}\lambda} e^{-\lambda x - x^2/2} & x < 0. \end{cases}$$

Differentiating with respect to x we get

$$h'(x) = \begin{cases} \frac{2}{(2\pi)^{1/2}\lambda} (\lambda - x) e^{\lambda x - x^2/2} & x \geq 0 \\ \\ \frac{2}{(2\pi)^{1/2}\lambda} (-1)(\lambda + x) e^{-\lambda x - x^2/2} & x < 0. \end{cases}$$

Therefore $h'(x) = 0 \Leftrightarrow x = \pm\lambda$. This gives the optimal bound as a function of λ

$$M_{\text{opt}}(\lambda) = h(\lambda) = h(-\lambda) = \frac{2}{(2\pi)^{1/2}\lambda} e^{\lambda^2/2},$$

which can be further improved by minimising $M_{\text{opt}}(\lambda)$ with respect to λ, i.e.,

$$M'_{\text{opt}}(\lambda) = \frac{\mathrm{d}\, M_{\text{opt}}(\lambda)}{\mathrm{d}\lambda} = \frac{2}{(2\pi)^{1/2}} e^{\lambda^2/2}(1 - 1/\lambda^2) = 0$$

so that $\lambda = 1$ ($M'_{\text{opt}}(\lambda) = 0$). Hence the optimal proposal density is a Laplace(1) density with probability of acceptance

$$\Pr(\text{accept}) = 1/M_{\text{opt}}(1) = (2\pi)^{1/2}/\{2\sqrt{e}\} \approx 76/100$$

The following code shows the density of the target distribution (N(0,1)), the optimal proposal density (Laplace(1)) and the corresponding optimal envelope ($M_{\text{opt}}(1)g(x;1)$)

```
dlaplace <- function(x,lambda)
{
    (lambda/2)*exp(-lambda*abs(x))
}

rlaplace <- function(n)
{
    B       <- rbinom(n,1,0.5)
    B[B==0] <- -1
    E       <- rexp(n)
    return(B*E)
}

x   <- seq(-7,7,len=1000)
M   <- ((2/sqrt(2*pi))*exp(1/2))
plot(x,M*dlaplace(x,1),type="l",lty=2)
legend("topleft",lty=c(2,3,1),c("envelope Mg","proposal g","target f"),cex=1.5)
lines(x,dnorm(x,0,1),lty=1)
lines(x,dlaplace(x,1),lty=3)
```

Note that this rlaplace function exploits the fact that the Laplace distribution is defined as two Exponential distributions stuck back to back; one positive and one negative. As such, we can simulate from the Laplace by simulating from an Exponential distribution and then setting the value to negative with probability 0.5.

Below is R code that implements the rejection sampling algorithm

```
rej.sampl.stnorm <- function(n)
{
    ## ------------------------------------------------------------
    ## Rejection sampling algorithm: simulate n standard normal
    ## variates with Laplace(1) proposal.
    ## ------------------------------------------------------------
    M   <- ((2/sqrt(2*pi))*exp(1/2))
    res <- NULL
    for(i in 1:n)
    {
        t <- -1
        while(t<0)
        {
            X <- rlaplace(1)
            Y <- runif(1,0,M*dlaplace(X,1))
            if(Y < dnorm(X,0,1))
            {
                res[i] <- X             ## save X to res object
                                        ## when Y < phi(X)
                t      <- 1             ## make t +ve so
                                        ## as to exit  while
            }
        }
    }
    return(res)
```

```
}
```

To verify code, simulate a large sample ($n = 2^{14}$) and check if histogram matches the density of the standard normal distribution.

```
X <- rej.sampl.stnorm(2^14)
hist(X,breaks=40,freq=FALSE)
lines(x,dnorm(x,0,1),lty=2)
```

Question 5

The rejection sampling algorithm extends to multivariate settings in a straightforward manner:

1. Simulate $X_A \sim \text{Poi}(\lambda)$ and $X_B \sim \text{Poi}(\mu)$.
2. Simulate $Y \sim \text{Unif}(0, M_{\text{opt}}\, g(X_A, X_B))$.
3. If $Y < f(X_A, X_B)$ accept (X_A, X_B). Otherwise reject (X_A, X_B).
4. Go back to step 1.

Similarly to the previous exercise, we compute

$$h(x,y) = \frac{f(x,y)}{g(x,y)} = \tau_{\lambda,\mu}(x,y)$$

and observe that

$$h(x,y) \leq M_{\text{opt}} = \begin{cases} \max(1+\lambda\rho, 1+\mu\rho) & \text{when } \rho \geq 0 \\ \\ \max(1-\lambda\mu\rho, 1-\rho) & \text{when } \rho < 0. \end{cases}$$

To implement the rejection sampling algorithm in R, computation of the proposal density $g(x,y)$ in step 2 is required. Since g depends on the dependence function $\tau_{\lambda,\mu}(x,y)$, we first write a function in R that returns the value of $\tau_{\lambda,\mu}(x,y)$ for given x, y, μ, λ and ρ.

```
tau <- function(x,y,lambda,mu,rho)
{
    if(x==0 & y ==0)
    {
        return(1-lambda*mu*rho)
    }
    if(x==0 & y ==1)
    {
        return(1+lambda*rho)
    }
    if(x==1 & y ==0)
    {
        return(1+mu*rho)
    }
    if(x==1 & y ==1)
    {
        return(1-rho)
    } else{
```

```
        return(1)
    }
}
```

Next, we write a function in R that returns the value of the density $g(x, y)$

```
dbivpois <- function(x,y,lambda,mu,rho)
{
    stopifnot(max(-1/lambda,-1/mu) < rho &
              rho < min(1/(lambda*mu),1))  # make sure you include
                                           # appropriate parameter
                                           # constraints in your code
    dens <- dpois(x,lambda)*dpois(y,mu)*tau(x,y,lambda,mu)
    return(dens)
}
```

and finally a function that simulates n samples from the given bivariate Poisson mass function f using rejection sampling with proposal g

```
rej.sampl.bivpois <- function(n,lambda,mu,rho)
{
    ## Rejection sampling algorithm: simulate n samples from
    ## given bivariate Poisson distribution with
    ## independent bivariate Poisson proposal.
    if(rho >= 0 )
    {
        M   <- 1+max(lambda,mu)*rho
    }else{
        M   <- max(1 - lambda*mu*rho,1-rho)
    }
    res <- matrix(nrow=n,ncol=2)
    for(i in 1:n)
    {
        t <- -1
        while(t<0)
        {
            XA <- rpois(1,lambda)
            XB <- rpois(1,mu)
            Y  <- runif(1,0,M*(dpois(XA,lambda)*dpois(XB,mu)))
            if(Y < dbivpois(XA,XB,lambda,mu,rho))
            {
                res[i,] <- c(XA,XB)
                t       <- 1             ## make t +ve so
                                         ## as to exit while loop
            }
        }
    }
    return(res)
}
```