Supplement to "Ranking Forecasts by Stochastic Error Distance, Information, and Reliability Measures": Online Appendix

R Codes and Instructions

Omid M. Ardakani[a]
Nader Ebrahimi[b]
Ehsan S. Soofi[c]

[a] *Department of Economics, Armstrong State University, Savannah, Georgia, USA*
[b] *Division of Statistics, Northern Illinois University, DeKalb, Illinois, USA*
[c] *Sheldon B. Lubar School of Business and Center for Research on International Economics, University of Wisconsin-Milwaukee, Milwaukee, USA*

14 September 2016

Contents

Preliminaries

The codes are written in `R` and are sufficiently self-explanatory. You need to install the package `ggplot2` to generate the plots. You can install it from `CRAN` with `install.packages("x")` and use it in `R` with `library(x)`. The `e` and `z` sequences are defined as the forecast error and absolute error vectors.

The `stat` package provides the functions `dnorm` and `pnorm`, `dt` and `pt`, `dlogis` and `plogis`, which compute densities and distributions of normal, Student-t, and logistic distributions. We have developed the functions listed in Table 1 to compute densities, distributions, and survivals of generalized error, double generalized Pareto, generalized gamma, and Laplace distributions.

Table 1: Functions for computing densities, distributions, and survivals

Function	Description
`dge`	Generalized error density
`ddgp`	Double generalized Pareto density
`dlaplace`	Laplace (double exponential) density
`pge`	Generalized error distribution
`pgg`	Generalized gamma distribution
`sdgp`	Double generalized Pareto survival
`slaplace`	Laplace survival

PDF Plots

This section includes the code and instructions for generating Figures 1 and 2.

Figure 1 presents the PDFs of members of the families of distributions with various shapes and tail thickness models for the forecast error.

Generalized Error

The density function for generalized error, $f_\beta(e)$, is defined as

$$f_\beta(e) = \frac{\beta}{2\sigma\Gamma(1/\beta)} e^{-(|e-\mu|/\sigma)^\beta}.$$

The function `dge` computes this density.

```
# Error vector
e <- seq(-5,5, length = 99)
n <- length(e)

beta <- c(2,1,10)
nbeta <- length(beta)
```

```
# The function dge generates the PDF of generalized error distribution.
dge <- function (e, beta, mu, sigma)
{
    pdf = (beta/(2*sigma*gamma(1/beta))) * exp(-(abs(e - mu)/sigma)^beta)
    print(pdf)
}

generrpdf <- matrix(rep(0,n*nbeta), n, nbeta)

for (i in 1:nbeta){
    generrpdf[,i] <- dge(e, beta[i], mu = 0, sigma = 1)
}

data <- data.frame(e, generrpdf[,1], generrpdf[,2], generrpdf[,3])
colnames(data) <- c("e", "GE(2) [Normal(0,.5)]", "GE(1) [Laplace]", "GE(10)")

data <- melt(data, id = c("e"))

gedens <- ggplot(data) +  geom_line(aes(x=e, y=value, colour=variable,
                                  linetype = variable), size = 1) +
    ylab("PDF") +
    xlab("Error") +
                    theme_bw() +
        ggtitle("Generalized Error") +
                            theme(
                        legend.position="top", legend.title=element_blank(),
                        legend.key.width = unit(1, "cm"),
                        axis.line = element_line(colour = "black"),
                        panel.grid.major = element_blank(),
                        panel.grid.minor = element_blank(),
                        panel.border = element_blank(),
                        panel.background = element_blank(),
                        axis.text.x = element_text(size=18),
                        axis.text.y = element_text(size=18),
                        text = element_text(size=20)
                     )
```

Bell Shaped (Normal, Student-t, and Logistic)

```
# Error vector
e <- seq(-5,5, length = 100)

# Student-t density with df = 2
tpdf <- dt(e, df = 2, log = FALSE)
```

```
# Standard normal density
normpdf <- dnorm(e, mean = 0, sd = 1, log = FALSE)

# Logistic density
logispdf <- dlogis(e, location = 0, scale = 1)

data <- data.frame(e, normpdf, tpdf, logispdf)
colnames(data) <- c("e", "Normal(0,1)", "t(2)", "Logistic(0,1)")

data <- melt(data, id = c("e"))

normtlogisdens <- ggplot(data) +
    geom_line(aes(x=e, y=value, colour=variable,
                  linetype = variable), size = 1 ) +
    ylab("PDF") +
    xlab("Error") +
                     theme_bw() +
        ggtitle("Bell Shaped") +
                               theme(
                          legend.position="top", legend.title=element_blank(),
                          legend.key.width = unit(1, "cm"),
                          axis.line = element_line(colour = "black"),
                          panel.grid.major = element_blank(),
                          panel.grid.minor = element_blank(),
                          panel.border = element_blank(),
                          panel.background = element_blank(),
                          axis.text.x = element_text(size=18),
                          axis.text.y = element_text(size=18),
                          text = element_text(size=20)
                       )
```

Double Generalized Pareto

The density function for double generalized Pareto is written as

$$f_\alpha(e) = \frac{1}{2\left(1+\frac{|e|}{\alpha}\right)^{1+\alpha}}.$$

The function `ddgp` gives the density of the double generalized Pareto. The function `dlaplace` generates the Laplace density. The Laplace distribution is the limiting case in the double generalized Pareto ($DGP(\infty)$).

```
# Error vector
e <- seq(-5,5, length = 99)
n <- length(e)
```

```r
alpha <- c(1.1, 2)
nalpha <- length(alpha)

# The density of double generalized Pareto.
ddgp <- function(e, alpha){
    pdf <- 1/(2*((1 + abs(e)/alpha)^(alpha + 1)))
    print(pdf)
}

dgppdf <- matrix(rep(0, n), n, nalpha)

for (i in 1:nalpha){
    dgppdf[,i] <- ddgp(e, alpha[i])
}

# The density of Laplace (double exponential)
dlaplace <- function(e){
    pdf <- exp(-abs(e))/2
    print(pdf)
}

laplacepdf <- dlaplace(e)

data <- data.frame(e, dgppdf[,1], dgppdf[,2], laplacepdf)
colnames(data) <- c("e", "DGP(1.1)", "DGP(2)", "DGP(Infinity) [Laplace]")

data <- melt(data, id = c("e"))

dgpdens <- ggplot(data) +
    geom_line(aes(x=e, y=value, colour=variable,
                  linetype = variable), size = 1) +
    ylab("PDF") +
    xlab("Error") +
    theme_bw() +
            ggtitle("Double Generalized Pareto") +
                            theme(
                        legend.position="top", legend.title=element_blank(),
                        legend.key.width = unit(1, "cm"),
                        axis.line = element_line(colour = "black"),
                        panel.grid.major = element_blank(),
                        panel.grid.minor = element_blank(),
                        panel.border = element_blank(),
                        panel.background = element_blank(),
                        axis.text.x = element_text(size=18),
                        axis.text.y = element_text(size=18),
                        text = element_text(size=20)
                        )
```

CDF and Survival Plots

Figure 2 illustrates SED visualizations of the Student-t and generalized error families.

CDFs

Generalized Error

The cumulative distribution function of the generalized error family is given by

$$F(e) = \frac{1}{2} + sgn(e-\mu)\frac{\gamma\left[1/\beta, \left(\frac{|e-\mu|}{\sigma}\right)^{\beta}\right]}{2\Gamma(1/\beta)},$$

where γ denotes the lower incomplete gamma function. We create the function pge to compute this distribution.

```
# Error vector
e <- seq(-5,5, length = 100)
n <- length(e)

beta <- c(1,1.5,2)
nbeta <- length(beta)

pge <- function (e, beta, mu, sigma)
{
    e = (e - mu)/sigma
    s = abs(e)^beta
    g = 1/(2 * gamma(1/beta))
    igamma <- pgamma(s,1/beta)*gamma(1/beta) # incompelete gamma function
    cdf = 0.5 + sign(e - mu) * g * igamma
    print(cdf)
}

gedist <- matrix(rep(0,n*nbeta),n,nbeta)

for (i in 1:nbeta){
    gedist[,i] <- pge(e, beta[i], mu = 0, sigma = 1)
}

# Ideal forecast error distribution
f0 <- c(rep(0,n/2), rep(1,n/2))

data <- data.frame(e, gedist[,3], gedist[,2], gedist[,1], f0)
colnames(data) <- c("e", "GE(2)", "GE(1.5)", "GE(1)", "Fo")

data <- melt(data, id = c("e"))
```

```
gedist <- ggplot(data) +
    geom_line(aes(x=e, y=value, colour=variable,
                  linetype = variable), size = 1) +
    ylab("CDF") +
    xlab("Error") +
    ggtitle("Generalized Error Family") +
                    theme_bw() +
                            theme(
                         legend.position="top", legend.title=element_blank(),
                         legend.key.width = unit(1, "cm"),
                         axis.line = element_line(colour = "black"),
                         panel.grid.major = element_blank(),
                         panel.grid.minor = element_blank(),
                         panel.border = element_blank(),
                         panel.background = element_blank(),
                          axis.text.x = element_text(size=18),
                         axis.text.y = element_text(size=18),
                         text = element_text(size=20)
                     )
```

Normal and Student-t

```
# Error vector
e <- seq(-5,5, length = 100)
n <- length(e)

# Absolute error vector
z <- seq(0,5, length = 100)
m <- length(z)

# f0 is the ideal forcast error distribution.
f0 <- c(rep(0, n/2), rep(1, n/2))

# Studnet-t distribution with df = 2, 5, and 30.
df <- c(2,5,30)
ndf <- length(df)

t <- matrix(rep(0,n*ndf),n,ndf)

for (i in 1:ndf){
    t[,i] <- pt(e, df = df[i], lower.tail = TRUE, log.p = FALSE)
}

data <- data.frame(e,t[,3], t[,2],  t[,1], f0)
colnames(data) <- c("e", "Normal(0,1)","t(5)", "t(2)", "Fo")
```

```
data <- melt(data, id = c("e"))

tdist <- ggplot(data) +
    geom_line(aes(x=e, y=value, colour=variable,
                  linetype = variable), size = 1 ) +
    ylab("CDF") +
    xlab("Error") +
        ggtitle("Student-t Family") +
                    theme_bw() +
                            theme(
                         legend.position="top", legend.title=element_blank(),
                         legend.key.width = unit(1, "cm"),
                         axis.line = element_line(colour = "black"),
                         panel.grid.major = element_blank(),
                         panel.grid.minor = element_blank(),
                         panel.border = element_blank(),
                         panel.background = element_blank(),
                         axis.text.x = element_text(size=18),
                         axis.text.y = element_text(size=18),
                         text = element_text(size=20)
                     )
```

Survivals

Generalized Gamma

For the generalized error, the distribution of $|e|$ is the generalized gamma with the gamma shape parameter of $1/\beta$ and Weibull shape parameter β,

$$F(|e|) = \frac{\gamma\left[1/\beta, \left(\frac{|e|-\mu}{\sigma}\right)^{\beta}\right]}{\Gamma(1/\beta)}.$$

```
# Absolute error vector
z <- seq(0,5, length = 100)
m <- length(z)

beta <- c(1,1.5,2)
nbeta <- length(beta)

pgg <- function (abse, beta, mu, sigma)
{
    abse = (abse - mu)/sigma
    s = abs(abse)^beta
    g = 1/gamma(1/beta)
    igamma <- pgamma(s,1/beta)*gamma(1/beta) # incompelete gamma function
```

```
    cdf = g * igamma
    print(cdf)
}

gedist <- matrix(rep(0,m*nbeta), m, nbeta)

for (i in 1:nbeta){
    gedist[,i] <- pgg(z, beta[i], mu = 0, sigma = 1)
}

gesurv <- 1 - gedist

data <- data.frame(z, gesurv[,3], gesurv[,2], gesurv[,1])
colnames(data) <- c("z","GE(2)","GE(1.5)",  "GE(1)")

data <- melt(data, id = c("z"))

gesurv <- ggplot(data) +  geom_line(aes(x=z, y=value, colour=variable,
                                        linetype = variable), size = 1) +
    ylab("Survival") +
    xlab("Absolute Error") +
                    theme_bw() +
                            theme(
                        legend.position="top", legend.title=element_blank(),
                        legend.key.width = unit(1, "cm"),
                        axis.line = element_line(colour = "black"),
                        panel.grid.major = element_blank(),
                        panel.grid.minor = element_blank(),
                        panel.border = element_blank(),
                        panel.background = element_blank(),
                        axis.text.x = element_text(size=18),
                        axis.text.y = element_text(size=18),
                        text = element_text(size=20)
                     )
```

Folded normal and Folded-t

For normal and t distributed errors, the distribution of |e| are folded normal and folded-t. The following code generates the lower right panel in Figure 2.

```
tdist <- matrix(rep(0,m*ndf),m,ndf)

for (i in 1:ndf){
    tdist[,i] <- 2*pt(z, df = df[i], lower.tail = TRUE, log.p = FALSE)
}
```

```
# Survival function of absolute values
tsurv <- matrix(rep(0,m*ndf),m,ndf)

for (i in 1:ndf){
    tsurv[,i] <- 2 - tdist[,i]
}

data <- data.frame(z, tsurv[,3], tsurv[,2], tsurv[,1])
colnames(data) <- c("z", "Normal(0,1)" , "t(5)","t(2)")

data <- melt(data, id = c("z"))

tsurv <- ggplot(data) +
    geom_line(aes(x=z, y=value, colour=variable,
                  linetype = variable), size = 1) +
    ylab("Survival") +
    xlab("Absolute Error") +
                     theme_bw() +
                             theme(
                          legend.position="top", legend.title=element_blank(),
                          legend.key.width = unit(1, "cm"),
                          axis.line = element_line(colour = "black"),
                          panel.grid.major = element_blank(),
                          panel.grid.minor = element_blank(),
                          panel.border = element_blank(),
                          panel.background = element_blank(),
                          axis.text.x = element_text(size=18),
                          axis.text.y = element_text(size=18),
                          text = element_text(size=20)
                       )
```

Mean Excess Error and Survival Plots

This section provides the code for generating Figures 3, 4, and 5. We, first, present the mean excess errors for various distributions followed by their survivals. Figures 3 and 4 show the mean excess error and survival plots of normal, logistic, and double generalized Pareto error models.

Mean Excess Errors

Normal and Logistic Error Models

`normmee` and `logismee` functions compute the MEEs for normal and logistic error models. The $MEE(\tau)$ of normal is

$$MEE(\tau) = \frac{\phi(\tau)}{1-\Phi(\tau)} - \tau,$$

where $\Phi(\cdot)$ is the standard normal CDF and $\phi(\cdot)$ is the standard normal PDF. The $MEE(\tau)$ of logistic is

$$MEE(\tau) = \frac{2\log(1+e^{-\tau})}{S(\tau)},$$

where $S(\cdot)$ is the survival function.

```
# Error vector
e <- seq(-5,5, length = 100)
n <- length(e)

# Thresholds
tau <- seq(0,5,.1)
l <- length(tau)

# Standard deviations for the normal example
sd <- c(1,2)
nsd <- length(sd)

# Scale parameter for the logistic example
lambda <- 1

# Mean excess error
normmee <- matrix(rep(0, l), l, nsd)

# MEE of two normal models: N(0,1) and N(0,4)
for (i in 1:length(sd)){
    num <- sd[i]*dnorm(tau/sd[i], mean = 0, sd = sd[i], log = FALSE)
    den <- 1 - pnorm(tau/sd[i])
    normmee[,i] <- num/den - tau
}

logismee <- lambda*(1+exp(tau/lambda))*log(1+exp(-tau/lambda))

data <- data.frame(tau, normmee[,1], logismee)
colnames(data) <- c("tau", "Normal(0,1)", "Logistic(0,1)")

data <- melt(data, id = c("tau"))

normlogismee <- ggplot(data) +  geom_line(aes(x=tau, color=variable, y=value,
                                              linetype = variable), size = 1) +
    ylab("MEE") +
        xlab("Error Tolerance Threshold") +
```

```
                        theme_bw() +
                    theme(
                        legend.position="top", legend.title=element_blank(),
                        legend.key.width = unit(1, "cm"),
                        axis.line = element_line(colour = "black"),
                        panel.grid.major = element_blank(),
                        panel.grid.minor = element_blank(),
                        panel.border = element_blank(),
                        panel.background = element_blank(),
                        axis.text.x = element_text(size=18),
                        axis.text.y = element_text(size=18),
                        text = element_text(size=20)
                    )
```

Double Generalized Pareto Error Models

```
# dgp: double generalized pareto
alpha <- c(4, 2)
nalpha <- length(alpha)

dgpmee <- matrix(rep(0,l), l,nalpha)

tau <- seq(0,5,.1)
l <- length(tau)

for (i in 1:nalpha){
    dgpmee[,i] <- (tau + alpha[i])/(alpha[i] - 1)
}

# Laplace (double exponential)
laplacemee <- rep(1, l)
data <- data.frame(tau,laplacemee, dgpmee[,1], dgpmee[,2])
colnames(data) <- c("tau", "DGP(Infinity) [Laplace]",  "DGP(4)", "DGP(2)")

data <- melt(data, id = c("tau"))

dgpmee <- ggplot(data) +
    geom_line(aes(x=tau, y=value, colour=variable,
                  linetype = variable), size = 1) +
    ylab("MEE") +
        xlab("Error Tolerance Threshold") +
            ylim(0,8) +
        geom_hline(yintercept=0) +
    scale_x_continuous(expand = c(0, 0)) +
    scale_y_continuous(expand = c(0, 0)) +
```

```
                    theme_bw() +

                       theme(
                           legend.position="top", legend.title=element_blank(),
                           legend.key.width = unit(1, "cm"),
                           axis.line = element_line(colour = "black"),
                           panel.grid.major = element_blank(),
                           panel.grid.minor = element_blank(),
                           panel.border = element_blank(),
                           panel.background = element_blank(),
                           axis.text.x = element_text(size=18),
                           axis.text.y = element_text(size=18),
                           text = element_text(size=20)
                       )
```

Survivals

Normal and Logistic Error Models

The survival function of normal is given by

$$S(\tau) = 2[1 - \Phi(\tau)]$$

and the logistic survival is defined as

$$S(\tau) = 2e^{-\tau}/(1 + e^{-\tau}).$$

```
# Absolute error vector
z <- seq(0,5, length = 100)
m <- length(z)

normsurv <- 2 - 2*pnorm(z, mean = 0, sd = sd[1], log = FALSE)
logissurv <- 2 - 2*plogis(z, location = 0, scale = lambda)

data <- data.frame(z, normsurv, logissurv)
colnames(data) <- c("z", "Normal(0,1)", "Logistic(0,1)")

data <- melt(data, id = c("z"))

normlogissurv <- ggplot(data) +
    geom_line(aes(x=z, y=value, colour=variable,
                  linetype = variable), size = 1) +
    ylab("Survival") +
    xlab("Absolute Error") +
                     theme_bw() +
```

```
                        theme(
                    legend.position="top", legend.title=element_blank(),
                    legend.key.width = unit(1, "cm"),
                    axis.line = element_line(colour = "black"),
                    panel.grid.major = element_blank(),
                    panel.grid.minor = element_blank(),
                    panel.border = element_blank(),
                    panel.background = element_blank(),
                    axis.text.x = element_text(size=18),
                    axis.text.y = element_text(size=18),
                    text = element_text(size=20)
                )
```

Double Generalized Pareto Error Models

The functions `sdgp` and `slaplace` give the survivals of the double generalized Pareto and Laplace error models.

```
# Absolute error vector
z <- seq(0,5, length = 100)
m <- length(z)

# Thresholds
tau <- seq(0,5,.1)
l <- length(tau)

alpha <- c(4,2)
nlapha <- length(alpha)

# Survival of the double generalized Pareto.
sdgp <- function(abse, alpha){
    survival <- 1/(1 + abse/alpha)^alpha
    print(survival)
}

dsurv <- matrix(rep(0, m), m, nalpha)

for (i in 1:nalpha){
    dsurv[,i] <- sdgp(z, alpha[i])
}

# Survival of the Laplace distribution
slaplace <- function(abse){
    survival <-  exp(-abse)
    print(survival)
}
```

```
lsurv <- slaplace(z)

data <- data.frame(z, lsurv, dsurv[,1], dsurv[,2])
colnames(data) <- c("z", "DGP(Infinity) [Laplace]","DGP(4)", "DGP(2)")

data <- melt(data, id = c("z"))

dgpsurv <- ggplot(data) +
    geom_line(aes(x=z, y=value, colour=variable,
                  linetype = variable), size = 1) +
    ylab("Survival") +
    xlab("Absolute Error") +
                     theme_bw() +

                           theme(
                        legend.position="top", legend.title=element_blank(),
                        legend.key.width = unit(1, "cm"),
                        axis.line = element_line(colour = "black"),
                        panel.grid.major = element_blank(),
                        panel.grid.minor = element_blank(),
                        panel.border = element_blank(),
                        panel.background = element_blank(),
                        axis.text.x = element_text(size=18),
                        axis.text.y = element_text(size=18),
                        text = element_text(size=20)
                     )
```

MEEs of Mixture Models

Figure 5 presents the mean excess error of two mixture models.

The mean excess error of the mixture is written as

$$MEE(\tau) = w(\tau)MEE_1(\tau) + (1 - w(\tau))MEE_2(\tau),$$

where $w(\tau) = \frac{pS_1(\tau)}{S(\tau)}$.

Laplace Error Mixture

```
tau <- seq(0,10,.1)
l <- length(tau)

sigma <- c(1, 2)
nsigma <- length(sigma)
```

```
p <- .9

laplacemee <- matrix(rep(0, l), l, nsigma)

for (i in 1:nsigma){
    laplacemee[,i] <- rep(sigma[i], l)
}

lsurv <- matrix(rep(0,l), l, nsigma)

for (i in 1:nsigma){
    lsurv[,i] <- exp(-tau/sigma[i])
}

mixsurve <- p*lsurv[,1] + (1 - p)*lsurv[,2]

w <- p*(lsurv[,1]/mixsurve)

meemix <- w*1 + (1 - w)*2

data <- data.frame(tau,laplacemee[,1], laplacemee[,2], meemix)
colnames(data) <- c("tau", "Laplace(1)", "Laplace(2)", "Mixture")
data <- melt(data, id = c("tau"))

laplacemixmee <- ggplot(data) +  geom_line(aes(x=tau, color=variable, y=value,
                                               linetype = variable), size = 1) +
    ylab("MEE") +
        xlab("Error Tolerance Threshold") +
    theme_bw() +
    expand_limits(y = 0) +
                      theme(
                          legend.position="top", legend.title=element_blank(),
                          legend.key.width = unit(1, "cm"),
                          axis.line = element_line(colour = "black"),
                          panel.grid.major = element_blank(),
                          panel.grid.minor = element_blank(),
                          panel.border = element_blank(),
                          panel.background = element_blank(),
                          axis.text.x = element_text(size=18),
                          axis.text.y = element_text(size=18),
                          text = element_text(size=20)
                      )
```

Normal Error Mixture

```
tau <- seq(0,5,.1)
l <- length(tau)

sd <- c(1,2)
nsd <- length(sd)

p <- .9

normmee <- matrix(rep(0, l), l, nsd)

normcdf <- matrix(rep(0,l), l,nsd)

for (i in 1:nsd){
    normcdf[,i] <- pnorm(tau, mean = 0, sd = sd[i], log = FALSE)
}

normmixcdf <- p*normcdf[,1] + (1-p)*normcdf[,2]

normsurv <- matrix(rep(0,l), l, nsd)

for (i in 1:2){
    normsurv[,i] <- 2 - 2*normcdf[,i]
}

normmixsurv <- 2 - 2*normmixcdf

for (i in 1:length(sd)){
    num <- sd[i]*dnorm(tau/sd[i], mean = 0)
    den <- 1 - pnorm(tau/sd[i], mean = 0)
    normmee[,i] <- num/den - tau
}

w <- p*normsurv[,1]/normmixsurv

mixmee <- w*normmee[,1] + (1 - w)*normmee[,2]

data <- data.frame(tau, normmee[,1], normmee[,2], mixmee)
colnames(data) <- c("tau", "Normal(0,1)", "Normal(0,4)", "Mixture")
data <- melt(data, id = c("tau"))

normmixmee <- ggplot(data) +  geom_line(aes(x=tau, color=variable, y=value,
                                            linetype = variable), size = 1) +
    ylab("MEE") +
        xlab("Error Tolerance Threshold") +
                            theme_bw() +
```

```
                    theme(
                        legend.position="top", legend.title=element_blank(),
                        legend.key.width = unit(1, "cm"),
                        axis.line = element_line(colour = "black"),
                        panel.grid.major = element_blank(),
                        panel.grid.minor = element_blank(),
                        panel.border = element_blank(),
                        panel.background = element_blank(),
                        axis.text.x = element_text(size=18),
                        axis.text.y = element_text(size=18),
                        text = element_text(size=20)
                    )
```

Mean Excess Error Risk Plots

Figure 6 shows bounds for the excess error risks of the Student-t, GE, and DGP error distributions. The bounds are written as

$$e^{H(e)-H_\gamma} \leq MEER(\tau) \leq \frac{V(e)}{2MAE(e)},$$

where $\gamma \approx .5772$ is the Euler constant and $H_\gamma = \gamma + 1 + \log 2$ is the entropy of the Laplace distribution. In order to find the lower and upper bounds, we need to find the variance, mean absolute error, and entropy.

MEER bounds: Generalized Error

```
beta <- seq(.5,5, length = 100)
var <- gamma(3/beta)/gamma(1/beta) # variance
sdev <- sqrt(var)

entropy <- log(2*gamma(1/beta)/beta)+(1/beta) # entropy
mae <- gamma(2/beta)/gamma(1/beta) # mean absolute error

lbound <- 1/2*exp(-1.5772+entropy)
ubound <- var/(2*mae)

data <- data.frame(sdev, lbound, ubound)

colnames(data) <- c("sdev", "Lower Bounds", "Upper Bounds")
data <- melt(data, id = c("sdev"))

gebounds <- ggplot(data) +  geom_line(aes(x=sdev, color="red",
```

```
                                    y=value, linetype = variable), size = 1) +
    ylab("") +
              ggtitle("Generalized Error") +
    xlab("Error standard deviation") +
        scale_linetype_manual(values=c("twodash", "twodash"))+
                            theme_bw() +
                      theme(
                          legend.position="", legend.title=element_blank(),
                          legend.key.width = unit(1, "cm"),
                          axis.line = element_line(colour = "black"),
                          panel.grid.major = element_blank(),
                          panel.grid.minor = element_blank(),
                          panel.border = element_blank(),
                          panel.background = element_blank(),
                          axis.text.x = element_text(size=18),
                          axis.text.y = element_text(size=18),
                          text = element_text(size=20)
                      )
```

MEER bounds: Student-t

```
df <- seq(3,100)
var <- df/(df - 2) # variance
sdev <- sqrt(var)

c <-gamma(df/2 + .5)/(sqrt(df*pi)*gamma(df/2))
d <- digamma((df+1)/2) - digamma(df/2)
entropy <- -log(c) + (df+1)*d/2 # entropy
mae <- (2*df*c)/(df - 1) # mean absolute error

lbound <- 1/2*exp(-1.5772+entropy)
ubound <- var/(2*mae)

data <- data.frame(sdev, lbound, ubound)

colnames(data) <- c("sdev", "Lower Bound", "Upper Bound")
data <- melt(data, id = c("sdev"))

tbounds <- ggplot(data) +  geom_line(aes(x=sdev, color="red",
                                    y=value, linetype = variable), size = 1) +
    ylab("") +
        xlab("Error standard deviation") +
    theme_bw() +
    scale_linetype_manual(values=c("twodash", "twodash"))+
           ggtitle("Student-t") +
```

```
                        theme(
                            legend.position="", legend.title=element_blank(),
                            legend.key.width = unit(1, "cm"),
                            axis.line = element_line(colour = "black"),
                            panel.grid.major = element_blank(),
                            panel.grid.minor = element_blank(),
                            panel.border = element_blank(),
                            panel.background = element_blank(),
                            axis.text.x = element_text(size=18),
                            axis.text.y = element_text(size=18),
                            text = element_text(size=20)
                        )
```

MEER bounds: Double Generalized Pareto

```
alpha <- seq(2.1,6.1, length = 100)
var <- (2*alpha^2)/((alpha-1)*(alpha-2)) # variance
sdev <- sqrt(var)

entropy <- log(2/alpha) + (1/alpha) + 1 # entropy
mae <- alpha/(alpha-1) # mean absolute error

lbound <- 1/2*exp(-1.5772+entropy)
meer <- alpha^2/(alpha - 1)^2 # mean excess error risk
ubound <- var/(2*mae)

data <- data.frame(sdev, lbound, meer, ubound)

colnames(data) <- c("sdev", "l", "MEER", "u")
data <- melt(data, id = c("sdev"))

dgpbounds <- ggplot(data) +  geom_line(aes(x=sdev, color=variable,
                                    y=value, linetype = variable), size = 1) +
    ylab("") +
    xlab("Error standard deviation") +
    scale_linetype_manual(values=c("twodash", "solid", "twodash"))+
     scale_color_manual(values=c("palevioletred2","royalblue","palevioletred2"))+
              ggtitle("Double Generalized Pareto") +
                        theme_bw() +
                        theme(
                            legend.position="", legend.title=element_blank(),
                            legend.key.width = unit(1, "cm"),
                            axis.line = element_line(colour = "black"),
                            panel.grid.major = element_blank(),
                            panel.grid.minor = element_blank(),
```

```
                        panel.border = element_blank(),
                        panel.background = element_blank(),
                        axis.text.x = element_text(size=18),
                        axis.text.y = element_text(size=18),
                        text = element_text(size=20)
                    )
```

Empirical SED, MEE, and MEER

In this section we provide the code for the results in Section 4 (Bond Risk Premia) containing Figure 7 and Tables 1 and 2.

R Functions

We use `zoo` and `forecast` packages to define time series class and to produce rolling forecasting results. We define the functions `mee` and `meer` to produce mean excess error and mean excess error risk. Table 2 lists the new functions for generating our findings.

Table 2: New R functions for the MEE, MEER, and rolling forecasting

Function	Description
`mee`	the mean excess error
`meer`	the mean excess error risk
`arimafcst`	the forecast for Arima(1,1,0)
`rolfcst`	forecasts based on rolling regression

```
# the mean excess error risk
meer <- function(err) {
    value <- diff(sort(abs(err)))
    no <- c(1:length(err))
    surv <- 1 - (no/length(err))
    surv <- surv[1:(length(err) - 1)]
    lsurv <- log(surv)
    meer <- -sum(value*surv*lsurv)
    return(meer)
}

# the mean excess error
mee <- function(err, tau) {
    pabserr <- abs(err) - tau
    value <- pabserr[pabserr>0]
    mee <- mean(value, na.rm = TRUE)
    print(mee)
```

```
}

# the forecast for Arima(1,1,0).
arimafcst <- function(x){
    fit <- Arima(x, order = c(1,1,0))
    fcst <- forecast(fit, h = 1)$mean[1]
    return(fcst)
}
```

Data

In the folder "application," you will find the folder "data" including the data for the bond risk premia application. This data set is used in Bauer and Hamilton (2015) and Cochrane and Piazzesi (2005). The functions `calcReturns` and `getFBdata` are written by Bauer and Hamilton (2015) to apply CRSP Fama-Bliss yields and prices, to calculate yields, and to produce excess returns.

```
calcReturns <- function(yields, h=12) {
    # construct annual excess returns
    T <- nrow(yields)
    Y <- as.matrix(yields[attr(yields, "yield.cols")])
    mats <- attr(yields, "mats")
    for (n in mats[mats>h])
        if (any(mats==n))
            if (any(mats==n-h)) {
                yields[paste0('xr', n/12)] <- c(-(n-h)*Y[(h+1):T, mats==n-h]
                                                + n*Y[1:(T-h), mats==n]
                                                - h*Y[1:(T-h), mats==h],
                                                rep(NA, h))/h
            } else {
                # approximate: y(t+h)^(n-h) = y(t+h)^(n)
                yields[paste0('xr', n/12)] <- c(-(n-h)*Y[(h+1):T, mats==n]
                                                + n*Y[1:(T-h),
                                                      mats==n] - h*Y[1:(T-h),
                                                                     mats==h],
                                                rep(NA, h))/h
            }
    yields$xr.avg <- rowMeans(as.matrix(yields[,grep("xr", names(yields))]))
    return(yields)
}

getFBdata <- function(beg=NULL, end=NULL, use.yields = FALSE) {
    if (use.yields) {
        # CRSP Fama-Bliss yields
        data <- read.table("~/data/yields.txt", header=TRUE, skip=3)
        yield.cols <- paste0('y', 1:5)
        names(data) <- c("dates", yield.cols)
```

```
    } else {
        # CRSP Fama-Bliss discount bond prices, calculate yields
        data <- read.table("~/data/prices.txt", header=TRUE, skip=3)
        price.cols <- paste0('p', 1:5)
        names(data) <- c("dates", price.cols)
        p <- log(data[price.cols]/100)  # log prices
        Y <- as.matrix(-p * rep(100/(1:5), each=nrow(p)))
        yield.cols <- sub("p", "y", price.cols)
        colnames(Y) <- yield.cols
        data <- cbind(data, Y)
    }
    mats <- 12*(1:5)
    data <- structure(data, mats=mats, yield.cols=yield.cols)
    # sample period
    data$yyyymm <- floor(data$dates/100)

    data$date <- as.Date(as.character(data$dates), format="%Y%m%d")
    if (!is.null(beg)) data <- data[data$yyyymm >= beg,]
    if (!is.null(end)) data <- data[data$yyyymm <= end,]
    data <- calcReturns(data)
    data
}

cpdata <- getFBdata(196401, 201312)

#  PCs up to 2003-12
Y <- as.matrix(cpdata[attr(cpdata, "yield.cols")])
W <- eigen(cov(Y[cpdata$yyyymm<=200312,]))$vectors[,1:5]
attr(cpdata, "W") <- W
cpdata[paste0("PC", 1:5)] <- Y %*% W

attach(cpdata)
```

Results

Table 1 reports One-step-ahead forecast error measures of predicting excess returns by five subsets of return-forecasting factors. The following code provide the results of the average excess return (The last panel in Table 1). To produce two-year, three-year, four-year, and five-year results, change the dependant variable to `xr2`, `xr3`, `xr4`, and `xr5`, respectively. For rolling forecasting, we consider the in-sample of 2002–2012.

```
nrol <- 468

# Intercept
int <- 1:600
data <- data.frame(int=int,y = xr.avg, x = 1)
```

```
zoodata <- zoo(data)
fmla <- y ~ x

rolfcst <- function(series)
    {
        mod <- lm(formula = fmla, data=as.zoo(series)) # the model
        nextobs <- max(series[,'int'])+1 # the first row that follows the window
        if (nextobs<=nrow(zoodata))
            {
                predicted=predict(mod,newdata=data.frame(x = zoodata[nextobs,'x'],
        y=zoodata[nextobs,'y']))
                attributes(predicted) <- NULL
                c(predicted=predicted)
            }
    }
rolwindow <- 468
rolling <- rollapply(zoodata,width=rolwindow, by =1 ,
FUN=rolfcst,by.column=FALSE,align='right')
fcst0 <- rolling

## PC1
data <- data.frame(int=int,y = xr.avg, x = PC1)
zoodata <- zoo(data)
fmla <- y ~ x
rolfcst <- function(series)
    {
        mod <- lm(formula = fmla, data=as.zoo(series))
        nextobs <- max(series[,'int'])+1
        if (nextobs<=nrow(zoodata))
            {
                predicted=predict(mod,
                                  newdata=data.frame(x = zoodata[nextobs,'x'],
                                                     y=zoodata[nextobs,'y']))
                attributes(predicted) <- NULL
                c(predicted=predicted)
            }
    }
rolwindow <- 468
rolling <- rollapply(zoodata,width=rolwindow, by =1 ,
                     FUN=rolfcst,by.column=FALSE,align='right')
fcst1 <- rolling

## PC2
data <- data.frame(int=int,y = xr.avg, x = PC2)
zoodata <- zoo(data)
fmla <- y ~ x
```

```
rolfcst <- function(series)
    {
        mod <- lm(formula = fmla, data=as.zoo(series))
        nextobs <- max(series[,'int'])+1
        if (nextobs<=nrow(zoodata))
            {
                predicted=predict(mod,
                                  newdata=data.frame(x = zoodata[nextobs,'x'],
                                                     y=zoodata[nextobs,'y']))
                attributes(predicted) <- NULL
                c(predicted=predicted)
            }
    }
rolwindow <- 468
rolling <- rollapply(zoodata,width=rolwindow, by =1 ,
                     FUN=rolfcst,by.column=FALSE,
                     align='right')
fcst2 <- rolling

## PC3
data <- data.frame(int=int,y = xr.avg, x = PC3)
zoodata <- zoo(data)
fmla <- y ~ x

rolfcst <- function(series)
    {
        mod <- lm(formula = fmla, data=as.zoo(series))
        nextobs <- max(series[,'int'])+1
        if (nextobs<=nrow(zoodata))
            {
                predicted=predict(mod,
                                  newdata=data.frame(x = zoodata[nextobs,'x'],
                                                     y=zoodata[nextobs,'y']))
                attributes(predicted) <- NULL
                c(predicted=predicted)
            }
    }
rolwindow <- 468
rolling <- rollapply(zoodata,
                     width=rolwindow, by =1 ,
                     FUN=rolfcst,by.column=FALSE,align='right')
fcst3 <- rolling

## PC1, PC2, PC3
data <- data.frame(int=int,y = xr.avg, cbind(PC1,PC2,PC3))
zoodata <- zoo(data)
fmla <- y ~ cbind(PC1,PC2,PC3)
```

```
rolfcst <- function(series)
    {
        mod <- lm(formula = fmla, data=as.zoo(series))
        nextobs <- max(series[,'int'])+1
        if (nextobs<=nrow(zoodata))
            {
                predicted=predict(mod,
                                  newdata=data.frame(zoodata[nextobs,3:5],
                                                     y=zoodata[nextobs,'y']))
                attributes(predicted) <- NULL
                c(predicted=predicted)
            }
    }
rolwindow <- 468  # to replicate CP and BH papers.
rolling <- rollapply(zoodata,width=rolwindow, by =1 ,
                     FUN=rolfcst,by.column=FALSE,align='right')
fcst123 <- rolling

## PC1, PC2, PC3, PC4, PC5
data <- data.frame(int=int,y = xr.avg, cbind(PC1,PC2,PC3, PC4, PC5))
zoodata <- zoo(data)
fmla <- y ~ cbind(PC1,PC2,PC3, PC4, PC5)

rolfcst <- function(series)
    {
        mod <- lm(formula = fmla, data=as.zoo(series))
        nextobs <- max(series[,'int'])+1
        if (nextobs<=nrow(zoodata))
            {
                predicted=predict(mod,
                                  newdata=data.frame(zoodata[nextobs,3:7],
                                                     y=zoodata[nextobs,'y']))
                attributes(predicted) <- NULL
                c(predicted=predicted)
            }
    }
rolwindow <- 468   # to replicate CP and BH papers.
rolling <- rollapply(zoodata,width=rolwindow, by =1 ,
                     FUN=rolfcst,by.column=FALSE,align='right')
fcst12345 <- rolling
fcsts <- cbind(fcst0, fcst1, fcst2, fcst3, fcst123, fcst12345)

actual <- xr.avg[468:599]

e0 <- actual - fcst0
e1 <-actual - fcst1
e2 <- actual - fcst2
```

```
e3 <-actual - fcst3
e123 <-actual - fcst123
e12345 <- actual - fcst12345

fe <- data.frame(cbind(e0,e1,e2,e3,e123,e12345))
fe <- fe[1:121,]

measures <- c("$RMSE$", "$MAE$", "MEER")
models <- c("$Intercept$", "$W_1$", "$W_2$", "$W_3$",
            "$W_1, W_2, W_3$", "$W_1, W_2, W_3, W_4, W_5$")

RMSE <- matrix(NA, length(models))

for (i in 1:ncol(fe)){
    RMSE[i] <- sqrt(mean(fe[,i]^2))
}

MAE <- matrix(NA, length(models))

for (i in 1:ncol(fe)){
    MAE[i] <- mean(abs(fe[,i]))
}


MEER <- matrix(NA, length(models))

for (i in 1:ncol(fe)){
    MEER[i] <- meer(fe[,i])
}

tbl <- matrix(NA, length(models), length(measures))
colnames(tbl) <- measures
rownames(tbl) <- models

for(i in 1:length(models)){
    tbl[i,1] <- RMSE[i,1]
    tbl[i,2] <- MAE[i,1]
    tbl[i,3] <- MEER[i,1]
}

xtbl <- xtable(tbl, digi=3)
print(xtbl, include.rownames=TRUE, include.colnames=TRUE,
      only.contents=TRUE, hline.after=nrow(xtbl),
      sanitize.text.function=function(x){x}, align = "lrrrrr")
```

Figure 7: Mean excess error of excess returns using return-forecasting factors.

```
tau <- seq(0,.5, by = .1)

mee0 <- matrix(0,length(tau),1)

for(i in 1:length(tau)){
        mee0[i,1] <- mee(e0,tau[i])
    }

mee1 <- matrix(0,length(tau),1)

for(i in 1:length(tau)){
        mee1[i,1] <- mee(e1,tau[i])
    }

mee2 <- matrix(0,length(tau),1)

for(i in 1:length(tau)){
        mee2[i,1] <- mee(e2,tau[i])
    }

mee3 <- matrix(0,length(tau),1)

for(i in 1:length(tau)){
        mee3[i,1] <- mee(e3,tau[i])
    }

mee123 <- matrix(0,length(tau),1)

for(i in 1:length(tau)){
        mee123[i,1] <- mee(e123,tau[i])
    }

mee12345 <- matrix(0,length(tau),1)

for(i in 1:length(tau)){
        mee12345[i,1] <- mee(e12345,tau[i])
    }


data <- data.frame(tau, mee0, mee1, mee2, mee3,
                   mee123, mee12345)

colnames(data) <- c("tau", "intercept", "1", "2", "3",
                    "1,2,3", "1,2,3,4,5")

data <- melt(data, id = c("tau"))
```

```
meeplot <- ggplot(data) +
    geom_line(aes(x=tau, color=variable, y=value,
                  linetype=variable), size = 1) +
    ylab("MEE") +
    xlab("Error Tolerance Threshold") +
    ggtitle("Average") +
                            theme_bw() +
                      theme(
                          legend.position="top", legend.title=element_blank(),
                          legend.key.width = unit(.8, "cm"),
                          axis.line = element_line(colour = "black"),
                          panel.grid.major = element_blank(),
                          panel.grid.minor = element_blank(),
                          panel.border = element_blank(),
                          panel.background = element_blank(),
                          axis.text.x = element_text(size=18),
                          axis.text.y = element_text(size=18),
                          text = element_text(size=20)
                      )
```

Table 2: One-step-ahead forecast error measures of predicting excess returns by arima(1,1,0) models.

```
depvars <- c("xr2", "xr3", "xr4", "xr5", "xr.avg")

tbl <- matrix(NA, length(depvars), 3)
rownames(tbl) <- depvars
colnames(tbl) <- c("$RMSE$", "$MAE$", "MEER")
for (i in seq_along(depvars)) {
    depvar <- depvars[i]
    arima110fit <- arima(get(depvar), order = c(1,1,0))
    arima110fcst <- rollapply(get(depvar), width = nrol, Arima110Fcst, by = 1)
    actual <- get(depvar)[468:588]
    fcst <- arima110fcst[1:121]
    arima110fe <- fcst - actual
    tbl[i, 1] <-  sqrt(mean(arima110fe^2))
    tbl[i, 2] <- mean(abs(arima110fe))
    tbl[i, 3] <- meer(arima110fe)
}

print(round(tbl, digi=3))
xtbl <- xtable(tbl, digi=3)
print(xtbl, include.rownames=TRUE, include.colnames=TRUE,
      only.contents=TRUE, hline.after=nrow(xtbl),
      sanitize.text.function=function(x){x})
```