R Notebook

$$X \sim U(0,1), \qquad Y = f(X) + \epsilon\sigma, \sigma \sim N(0,1)$$

where $var\{f(X)\} = 1$

thus

$$G^2_{Y|X} = (1+\sigma^2)^{-1}$$

generate dataset

```
library(energy) # for distance correlation
n = 225 # number of data points per simulation
n1 = n2 = 100 # replications
num.noise = 20 # number of noise level
num.type = 8 # number of function type
noise = sqrt(1/seq(0.01, 0.2, length.out = num.noise)-1)
value.cor = value.dcor = value.g2m = value.g2t = numeric(n1)
value.cor2 = value.dcor2 = value.g2m2 = value.g2t2 = numeric(n2)
power.cor = power.dcor = power.g2m = power.g2t = array(NA, c(num.type, num.noise))
genXY <- function(n = 225, epsi = 1, type = 1, resimulate = FALSE)
{
  x = runif(n, 0, 1)
  if (type == 1){
    # linear
    fx = x
  }
  else if (type == 2){
    # quadratic
    fx = x^2
  }
  else if (type == 3){
    # cubic
    fx = x^3
  }
  else if (type == 4){
    # radical
    fx = sqrt(x)
  }
  else if (type == 5){
    # low freq sine
    # freq = 1
    fx = sin(2*pi*x)
  }
  else if (type == 6){
    # triangle
    # max(1-\vert t\vert, 0)
    fx = sapply(x, function(xi) max(c(0, 1-abs(x))))
  }
  else if (type == 7){
    # high freq sine
    fx = sin(20*pi*x)
  }
```

```r
  else if (type == 8){
    # piecewise constant
    fx = ceiling(x/0.2)
  }
  y = fx + epsi*rnorm(n, 0, 1)
  if (resimulate)
    x = runif(n, 0, 1)
  res = list(X = x, Y = y)
  return(res)
}
```

estimate G_m^2 and G_t^2

```r
## fix lambda0 = 3
g2 <- function(X, Y){
  ## step 1: data preparation
  idx = order(X)
  x = X[idx]
  y = Y[idx]
  # normalize
  y = y - mean(y)
  y = sqrt(n)*y/sqrt(sum(y^2))
  ## step 2: main algorithm
  n = length(X)
  m = ceiling(sqrt(n))
  lambda = -3*log(n)/2
  alpha = exp(lambda)
  # initialize three sequences
  Mi = numeric(n)
  Bi = numeric(n)
  Ti = numeric(n)
  Mi[1] = 0
  Bi[1] = Ti[1] = 1
  for (i in m:n){
    bi = 0
    ti = 0
    if (i < 2*m)
    { # do not ignore
      Mi[i] = Mi[1]
      Bi[i] = Bi[1]
      Ti[i] = Ti[1]
      next
    }
    #mi = numeric(i-2*m+1)
    mi = rep(-Inf, i-2*m+1)
    kk = 0
    for (k in c(1, seq(m+1, i-m+1))){
      kk = kk + 1
      # regression y on x for k:i
      xx = x[k:i]
      yy = y[k:i]
      xx2 = xx-mean(xx)
      yy.hat = sum(xx2*yy)/sum(xx2^2)*xx2 + mean(yy)
      sigma2.hat = var(yy - yy.hat)
```

```
      l.ki = -(i-k)*log(sigma2.hat)/2
      mi[kk] = lambda + Mi[k] + l.ki
      L.ki = exp(l.ki)
      bi = bi + Bi[k] # no need to multiple alpha
      ti = ti + Ti[k]*L.ki # no need to multiple alpha
    }
    Mi[i] = max(mi)
    Bi[i] = bi
    Ti[i] = ti
  }
  ## step 3: final result
  res = list(g2m = 1-exp(-2/n*(Mi[n]-lambda)),
             g2t = 1-(Ti[n]/Bi[n])^{-2/n})
  return(res)
}
```

```
library(foreach)
library(doParallel)
```

```
## Loading required package: iterators
```

```
## Loading required package: parallel
```

```
cl<-makeCluster(4)
registerDoParallel(cl)
output = foreach(i=1:num.noise, .combine = list, .packages = "energy", .export = c("genXY", "g2", "n",
  value.cor = value.dcor = value.g2m = value.g2t = numeric(n1)
  value.cor2 = value.dcor2 = value.g2m2 = value.g2t2 = numeric(n2)
  power.cor = power.dcor = power.g2m = power.g2t = numeric(num.type)
  for (j in 1:num.type){
    for (k in 1:n1){
      res = genXY(epsi = noise[i], type = j, resimulate = TRUE)
      X = res$X
      Y = res$Y

      value.cor[k] = (cor(X, Y))^2
      value.dcor[k] = dcor(X, Y)
      value.g2 = g2(X, Y)
      value.g2m[k] = value.g2$g2m
      value.g2t[k] = value.g2$g2t
    }
    ## faster way
    # cl<-makeCluster(4)
    # clusterExport(cl, c("genXY", "g2", "i", "j", "n", "dcor"))
    # parSapply(cl, as.character(1:num.type), function(k){
    #   ki <- as.numeric(k)
    #   res = genXY(epsi = i, type = j, resimulate = TRUE)
    #   X = res$X
    #   Y = res$Y
    #   value.g2 = g2(X, Y)
    #   c(cor = (cor(X, Y))^2, dcor = dcor(X, Y), g2m = value.g2$g2m, value.g2$g2t)
    # })
    # stopCluster(cl)
    # calculate the rejection cutoffs
    cut.cor = quantile(value.cor, .95)
```

```r
    cut.dcor = quantile(value.dcor, .95)
    cut.g2m = quantile(value.g2m, .95)
    cut.g2t = quantile(value.g2t, .95)

    for (k in 1:n2){
      res = genXY(epsi = noise[i], type = j)
      X = res$X
      Y = res$Y
      # calculate the value
      value.cor2[k] = (cor(X, Y))^2
      value.dcor2[k] = dcor(X, Y)
      value.g22 = g2(X, Y)
      value.g2m2[k] = value.g22$g2m
      value.g2t2[k] = value.g22$g2t
    }

    # calculate the power
    # power.cor[j, i] = sum(value.cor2 > cut.cor)/n2
    # power.dcor[j, i] = sum(value.dcor > cut.dcor)/n2
    # power.g2m[j, i] = sum(value.g2m2 > cut.g2m)/n2
    # power.g2t[j, i] = sum(value.g2t2 > cut.g2t)/n2
    power.cor[j] = sum(value.cor2 > cut.cor)/n2
    power.dcor[j] = sum(value.dcor > cut.dcor)/n2
    power.g2m[j] = sum(value.g2m2 > cut.g2m)/n2
    power.g2t[j] = sum(value.g2t2 > cut.g2t)/n2
  }
  data.frame(power.cor, power.dcor, power.g2m, power.g2t)
}
```

```
## Warning in e$fun(obj, substitute(ex), parent.frame(), e$data): already
## exporting variable(s): genXY, g2, n1, n2, num.type
```

```r
stopCluster(cl)
```