Workshop 9

Principal component analysis

Low-dimensional data set

```
irispca<-princomp(iris[-5])
```

```
summary(irispca)
```

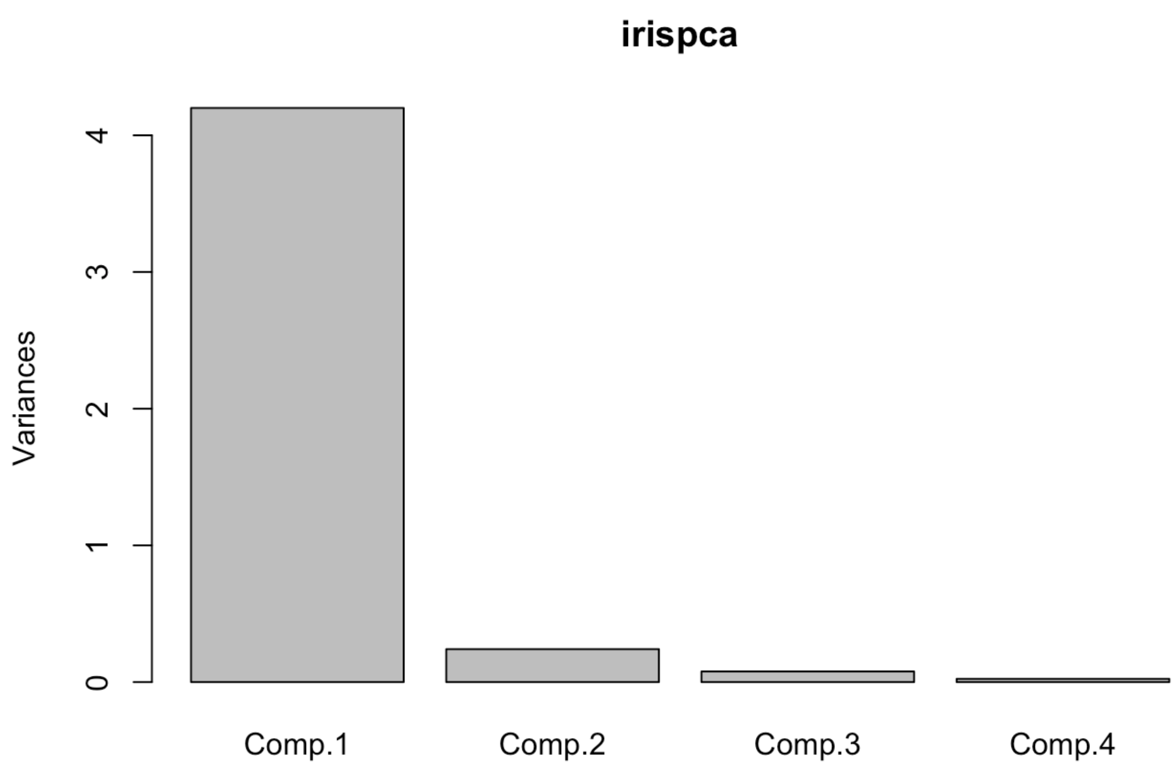
```
## Importance of components:
##                           Comp.1     Comp.2     Comp.3      Comp.4
## Standard deviation     2.0494032 0.49097143 0.27872586 0.153870700
## Proportion of Variance 0.9246187 0.05306648 0.01710261 0.005212184
## Cumulative Proportion  0.9246187 0.97768521 0.99478782 1.000000000
```

```
irispca$loadings
```

```
##
## Loadings:
##              Comp.1 Comp.2 Comp.3 Comp.4
## Sepal.Length  0.361 -0.657 -0.582  0.315
## Sepal.Width          -0.730  0.598 -0.320
## Petal.Length  0.857  0.173        -0.480
## Petal.Width   0.358         0.546  0.754
##
##                Comp.1 Comp.2 Comp.3 Comp.4
## SS loadings      1.00   1.00   1.00   1.00
## Proportion Var   0.25   0.25   0.25   0.25
## Cumulative Var   0.25   0.50   0.75   1.00
```

```
#irispca$scores
```

```
screeplot(irispca)
```



Higher-dimensional

You may need to install the `ElemStatLearn` package to get this data (`install.packages('ElemStatLearn')`).

```
library(ElemStatLearn)
data(phoneme)
dcl <- phoneme
```

This dataset has 258 dimensions but only the first 256 are numerical values so we will restrict our analysis to those dimensions.

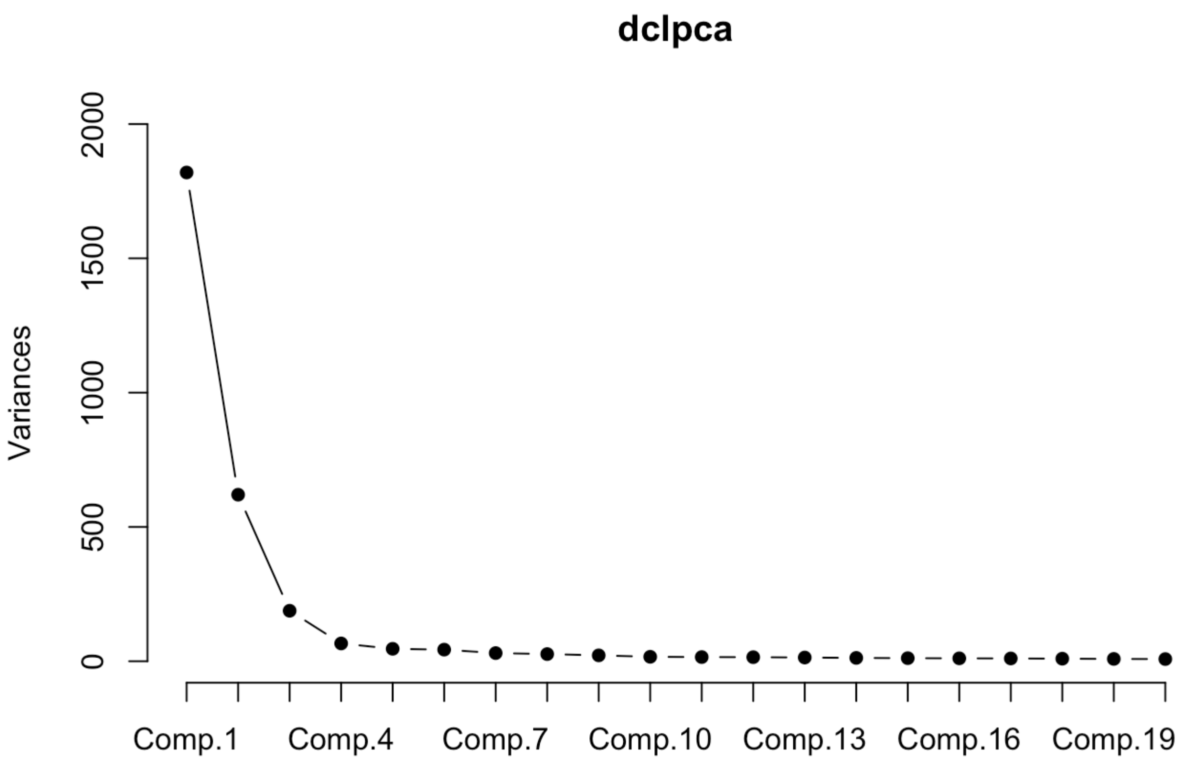
```
dcl <- dcl[,1:256]
```

Generate the principal components.

```
dclpca<-princomp(dcl)
```

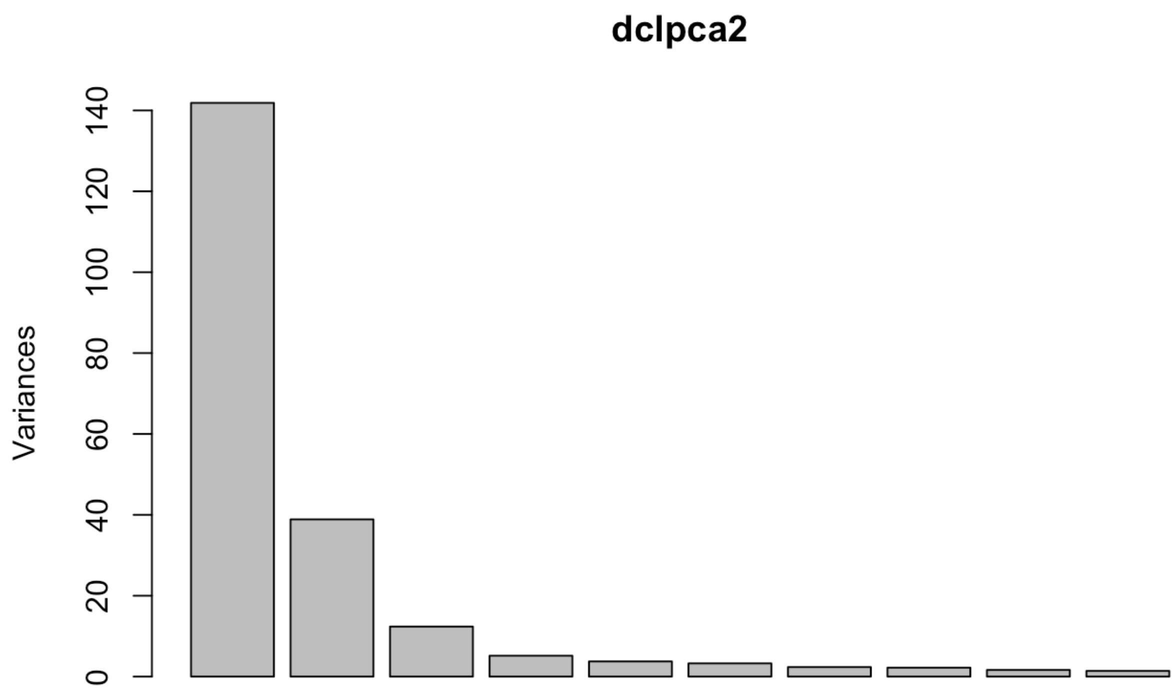
Do a screeplot of the first 20 eigenvalues.

```
screeplot(dclpca, 20, type="lines", pch=16, ylim=c(0,2000))
```

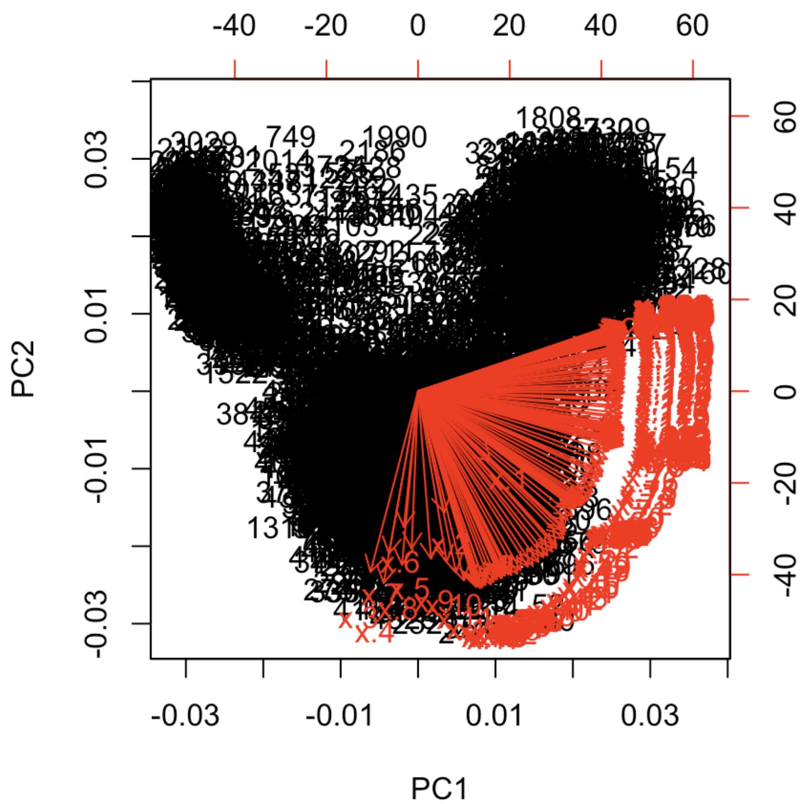



```
dclpca2 <- prcomp(dcl, scale=TRUE)
```

```
plot(dclpca2)
```




```
biplot(dclpca2)
```



Largest eigenvalue

Null case

```
n <- 100
p <- 50
```

```
y <- p/n
b <- (1+sqrt(y))^2
```

Eigenvalues given by `eigen` are in decreasing order, so we only need the first one.

```
maxeigen <- function(x) {
  eigen(x, only.values=T, symmetric = FALSE)$values[1]
}
```

```
Sigma <- diag(1, p, p)
```

```
lambda1 <- apply(rWishart(10000, n, Sigma), 3, maxeigen)
```

```
mu.np <- (sqrt(n-1/2)+sqrt(p-1/2))^2
sigma.np <- (sqrt(n)+sqrt(p))*(1/sqrt(n-0.5)+1/sqrt(p-0.5))^(1/3)
```

μ_{np} is close to b for large p, n.

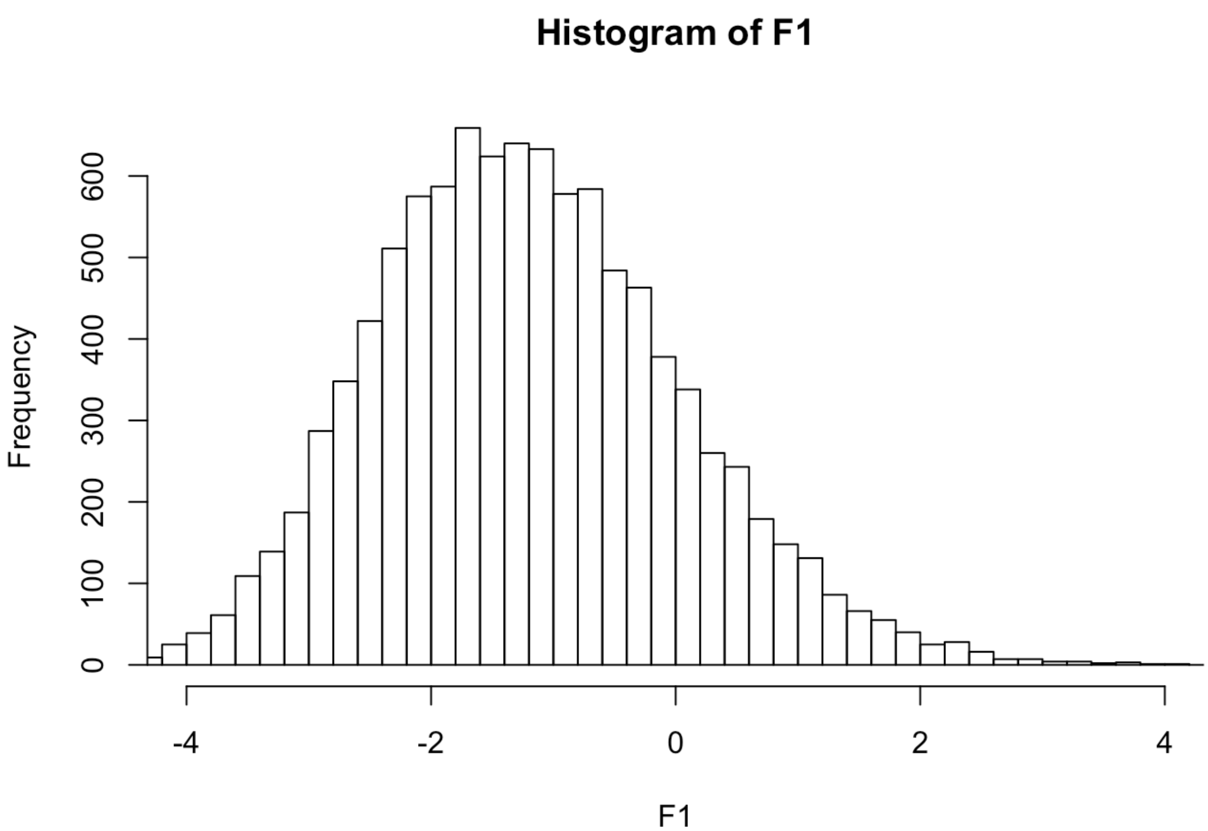
```
abs(mu.np - b)
```

```
## [1] 286.446
```

```
F1 <- (lambda1-mu.np)/sigma.np
```

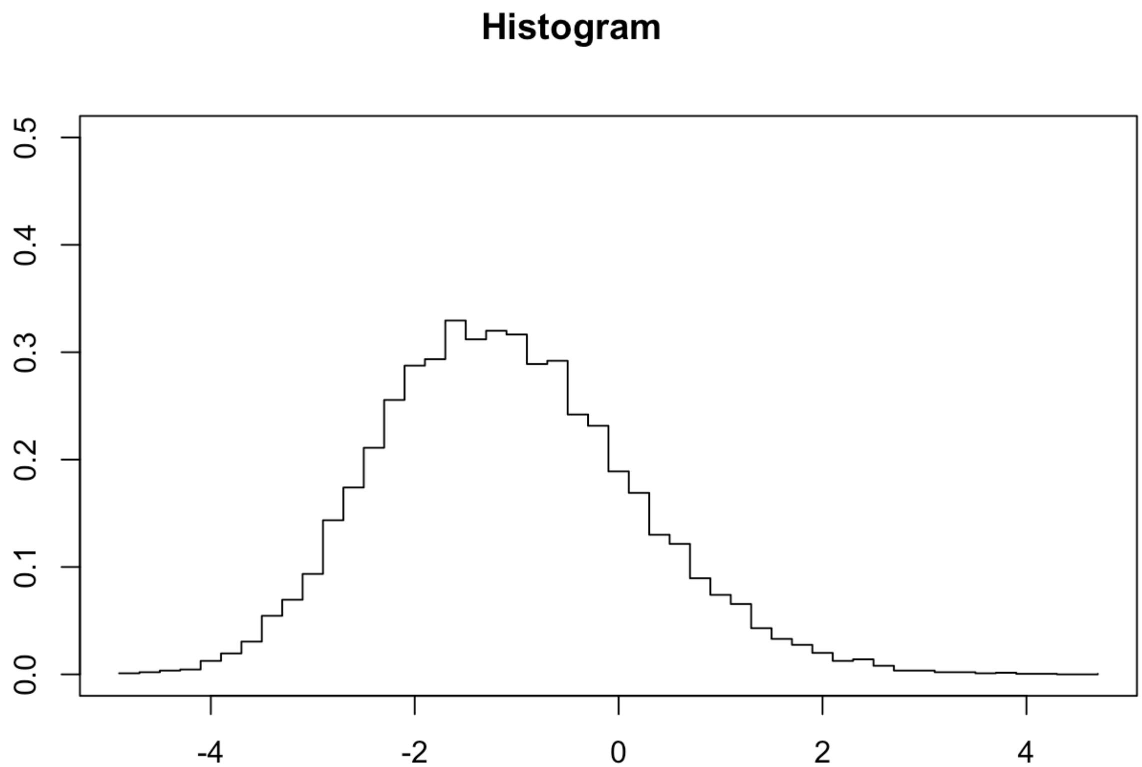
Generate histogram of MC simulations.

```
hist(F1, breaks=40, xlim=c(-4,4)) -> h
```



Plot the histogram.

```
plot(h$mids, h$density, type="s", xlab="", ylab="", ylim=c(0,0.5))
title(main="Histogram", outer=T, line=-2)
```

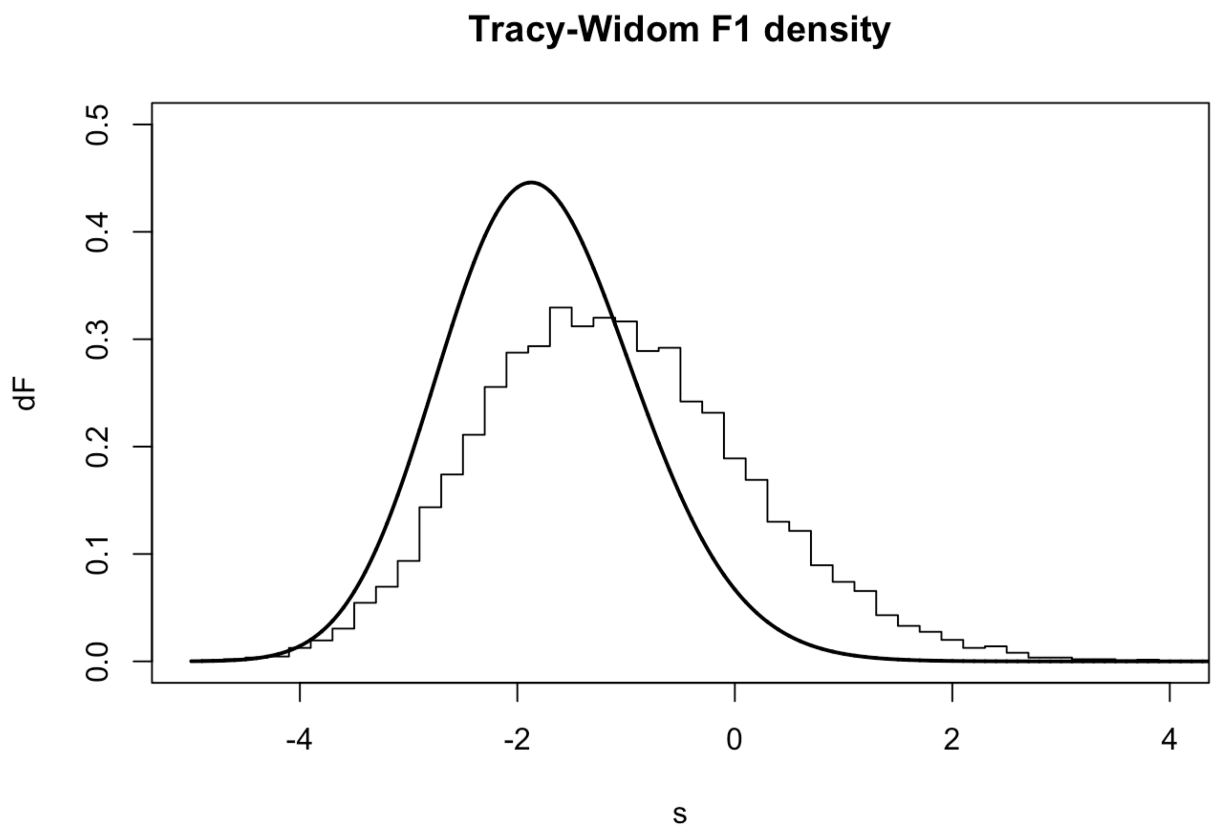


Solve for the Tracy-Widom F1 Distribution

For this next section, you'll need to install the `deSolve` and `gsl` packages. Do this with `install.packages(c('deSolve', 'gsl'))`.

Numerically solving to find the Tracy-Widom 1 density is not easy. We setup and solve the ODE as a system of first-order ODEs.

```r
library(deSolve)
library(gsl)

deq <- function(t, y, parms) {
    list(c(y[2],
           t * y[1] + 2 * y[1] ^ 3,
           y[4],
           y[1] ^ 2,
           y[1]))
}

t0 <- 5; tn <- -5
tspan <- seq(t0, tn, length.out=10000)
y0 <- c(airy_Ai(t0),
        airy_Ai_deriv(t0),
        0,
        airy_Ai(t0) ^ 2,
        0)

F.ode <- ode(y=y0, times=tspan, func=deq, parms=NULL, method="daspk")

s <- F.ode[, "time"]
F2 <- exp(-F.ode[, "3"])
dF <- -F2*F.ode[, "4"]
plot(s, dF, type='l', main="Tracy-Widom F1 density", xlim=c(-5, 4), ylim=c(0,0.
5), lwd=2)

lines(h$mids, h$density, type="s", xlab="", ylab="", ylim=c(0,0.5))
```



We try a bunch of numerical ODE integration methods.

```
tspan <- seq(t0, tn, length.out=1000)

plot(c(0,1), type='n', xlim=c(-5, 5), ylim=c(0,0.6))
methods = c("lsoda", "lsode", "lsodes", "lsodar", "vode", "daspk",
            "euler", "rk4", "ode23", "ode45", "radau",
            "bdf", "bdf_d", "adams", "impAdams", "impAdams_d")
for (m in methods) {
  F.ode <- ode(y=y0, times=tspan, func=deq, parms=NULL, method=m)
  s <- F.ode[, "time"]
  F2 <- exp(-F.ode[, "3"])
  dF <- -F2*F.ode[, "4"]
  lines(s, dF, type='l')
}
```

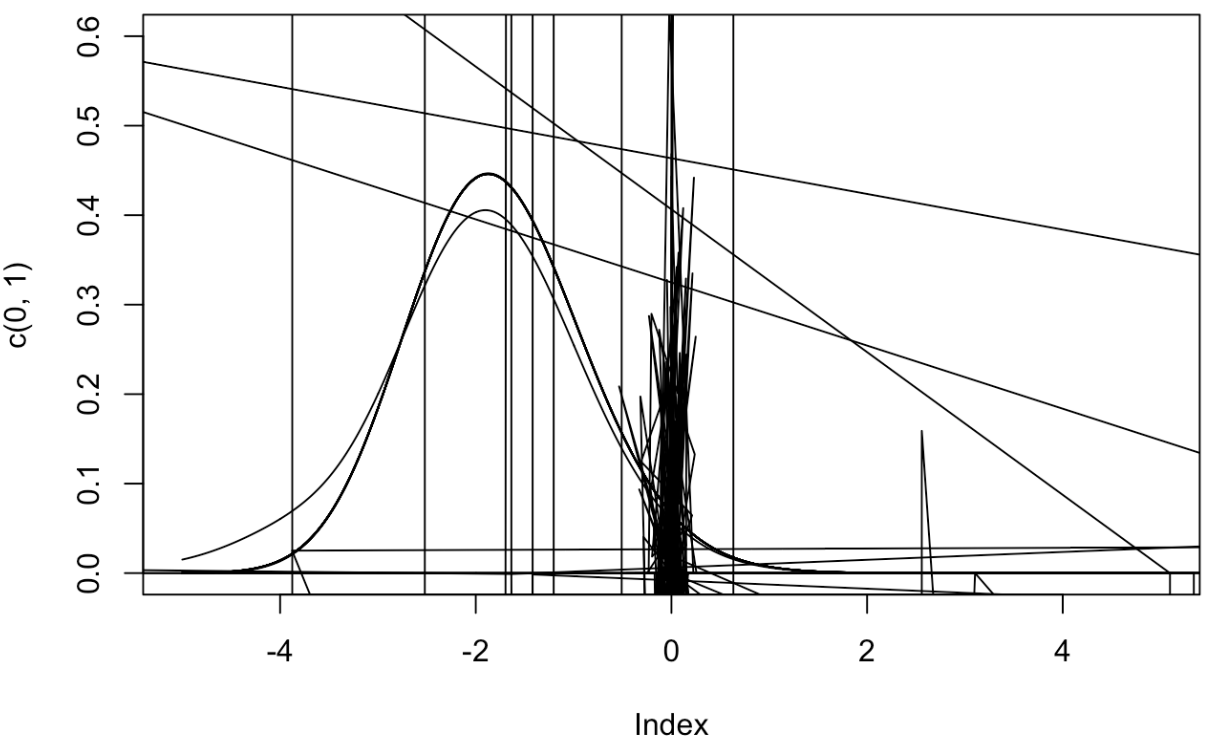
```
## dvode -- Warning.. Internal T (=R1) and H (=R2) are
##       such that in the machine, T + H = T on the next step
##       (H = step size). Solver will continue anyway.
## In above message, R1 = -4.95895, R2 = -3.28524e-16
## dvode -- Warning.. Internal T (=R1) and H (=R2) are
##       such that in the machine, T + H = T on the next step
##       (H = step size). Solver will continue anyway.
## In above message, R1 = -4.95895, R2 = -3.28524e-16
## dvode -- Warning.. Internal T (=R1) and H (=R2) are
##       such that in the machine, T + H = T on the next step
##       (H = step size). Solver will continue anyway.
## In above message, R1 = -4.95895, R2 = -3.28524e-16
## dvode -- Warning.. Internal T (=R1) and H (=R2) are
##       such that in the machine, T + H = T on the next step
##       (H = step size). Solver will continue anyway.
## In above message, R1 = -4.95895, R2 = -3.28524e-16
## dvode -- Warning.. Internal T (=R1) and H (=R2) are
##       such that in the machine, T + H = T on the next step
##       (H = step size). Solver will continue anyway.
## In above message, R1 = -4.95895, R2 = -3.28524e-16
## dvode -- Warning.. Internal T (=R1) and H (=R2) are
##       such that in the machine, T + H = T on the next step
##       (H = step size). Solver will continue anyway.
## In above message, R1 = -4.95895, R2 = -3.28524e-16
## dvode -- Warning.. Internal T (=R1) and H (=R2) are
##       such that in the machine, T + H = T on the next step
##       (H = step size). Solver will continue anyway.
## In above message, R1 = -4.95895, R2 = -3.28524e-16
## dvode -- Warning.. Internal T (=R1) and H (=R2) are
##       such that in the machine, T + H = T on the next step
##       (H = step size). Solver will continue anyway.
## In above message, R1 = -4.95895, R2 = -3.28524e-16
## dvode -- Warning.. Internal T (=R1) and H (=R2) are
##       such that in the machine, T + H = T on the next step
##       (H = step size). Solver will continue anyway.
## In above message, R1 = -4.95895, R2 = -2.35243e-16
## dvode -- Warning.. Internal T (=R1) and H (=R2) are
##       such that in the machine, T + H = T on the next step
##       (H = step size). Solver will continue anyway.
## In above message, R1 = -4.95895, R2 = -2.35243e-16
## dvode -- Above warning has been issued I1 times.            dvode -- ITASK
= I1 62
##       it will not be issued again for this problem.
## In above message, I1 = 10
## dvode -- At current T (=R1), MXSTEP (=I1) steps
##       taken on this call before reaching TOUT
```

```
##       with: R1 = -4.95895, I1=5000
```

```
## Warning in vode(y, times, func, parms, ...): an excessive amount of work (>
## maxsteps ) was done, but integration was not successful - increase maxsteps
```

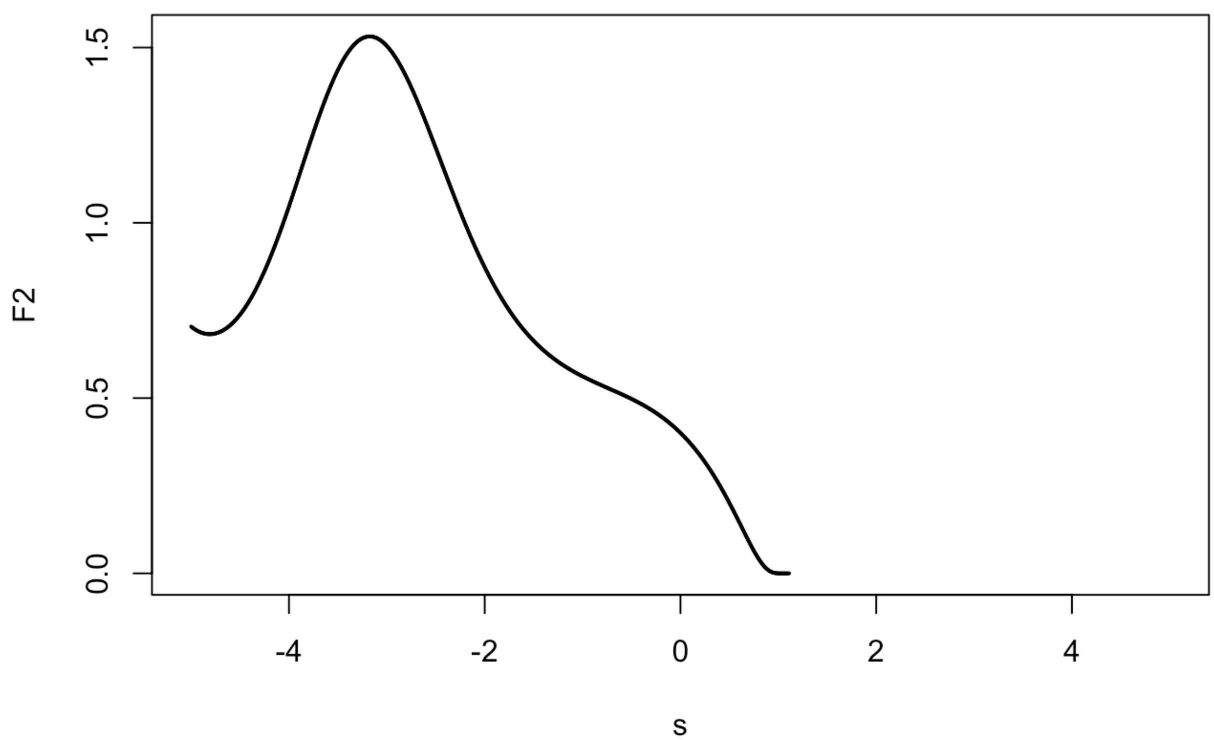
```
## Warning in vode(y, times, func, parms, ...): Returning early. Results are
## accurate, as far as they go
```

```
## Warning in daspk(y, times, func, parms, ...): Returning early. Results are
## accurate, as far as they go
```



The simpler system of ODEs doesn’t work very well.

```
library(deSolve)
library(gsl)

deq <- function(t, y, parms) {
    list(c(y[2],
           t * y[1] + 2 * y[1] ^ 3))
}

t0 <- -5; tn <- 5
tspan <- seq(t0, tn, length.out=100000)
y0 <- c(airy_Ai(t0),
        airy_Ai_deriv(t0))

F.ode <- ode(y=y0, times=tspan, func=deq, parms=NULL, method="ode23")

s <- F.ode[, "time"]
F2 <- exp(-F.ode[, "1"])
plot(s, F2, type='l', lwd=2)
```



There are better techniques for numerically solving these Painlevé transcendents but they are harder to implement in R. See the paper:

ON THE NUMERICAL EVALUATION OF DISTRIBUTIONS IN RANDOM MATRIX THEORY: A REVIEW. FOLKMAR BORNEMANN.