```
# DFT of the data from Stull, 1988, "An Intro to Boundary Layer Meteorology"
# Roland Stull,  23 June 2015, modified 29 Feb 2020, modified 20 Jan 2025

# Modified to Read in, or set, a time series to analyze.
# Then find the DFT of it.  Next, use the DFT to reconstruct the time series.
# Finally, compute and display the spectrum

# set up graphs on each page
par(mfrow = c(1, 1))               # graphs per page (1 rows, 1 columns)
#par(mfrow = c(2, 1))               # graphs per page (2 rows, 1 columns)
# par(tcl = 0.5)                    # tic marks on inside, and slightly larger label size
par(mar = c(4, 4, 1.5, 1.5),
    mex = 0.8,
    mgp = c(2, 0.5, 0))            # set graphics parameters
par(cex = 1.2 )

# Prepare to read in a data file.
setwd("~/Desktop")                 # set the working directory on my iMac
print(getwd())                          # display the working directory, to help debug

T2col = read.csv(filename<-file.choose(),header=TRUE)   # read file of t and T (degC)
T = T2col$T        # extract the column of dependent variables
t = T2col$t        # extract the column of independent variables

#=== Or Create the time series to be analyzed ===
# Create small time series from Stull, Chapter 8, p 304
# T <- c(8, 9, 9, 6, 10, 3, 5, 6)    # Create time series of Temperature (degC)

# === Plot the input data ===
plot(t ,T ,type="l", col="darkred", main="Input Data", xlab="t (s)",
     ylab="T (degC)", xlim=c(range(t, na.rm=TRUE)), ylim = c(range(T, na.rm=TRUE)) )
# abline(h=0.)      # draw horizontal line indicated bottom boundary (at y = 0)


# === Preliminary calculations
N <- length(T)    # size of data set
k <- c(0:(N-1))   # create vector of indices for each time-series element
N                 # display the length of the vector
k                 # display the indices
T                 # display this time series

#===Find the mean and population variance for the original time series
print(T.mean <- mean(T), digits=4)               # find and display the mean
print(T.var <-  mean((T - T.mean)^2), digits=4)   # find the biased (population) variance

#=== Compute the DFT ===
Fr <- rep(0, N)      # Initialize to zero a vector for real values of the spectrum
Fi <- rep(0, N)      # Initialize to zero a vector for imaginary values of the spectrum

# loop over each frequency (need to use the explicit loop to get at the index n)
for (n in 0:(N-1)) {                # for each harmonic in the DFT
  cosv <- T*cos(2*pi*n*k/N)         # vector of cosine terms for any one n
  Fr[n+1] <- sum(cosv)/N            # real component of spectrum
  sinv <- T*sin(2*pi*n*k/N)         # vector of sine terms for any one n
  Fi[n+1] <- -1 * sum(sinv)/N       # imaginary component of DFT
}

n <- c(0:(N-1))          # create vector of harmonic indices
n                        # display the harmonic indices
#round(Fr, digits = 5)    # display vector of real values of the DFT
#round(Fi, digits = 5)    # display vector of imaginary values of the DFT
zapsmall(Fr, digits = 5)    # display vector of real values of the DFT & set small values to 0
zapsmall(Fi, digits = 5)    # display vector of imaginary values of the DFT & set small values to 0

#=== Do an Inverse DFT to check if we get back the original time series ===
# Check to see if we get back the original time series

Tnew <- rep(0, N)    # initialize the new spec humidity vector

for (k in 0:(N-1)) {              # for each data point in the time series
  cos.ts <- Fr*cos(2*pi*n*k/N)     # vector of cosine terms in the timeseries for any one k
  Tnew[k+1] <- sum(cos.ts)         # real contribution to the time series
  sin.ts <- Fi*sin(2*pi*n*k/N)     # vector of sine terms in the timeseries for any one k
  Tnew[k+1] <- Tnew[k+1] -1 * sum(sin.ts)    # add imaginary to the real contribution
```

```
}

k <- c(0:(N-1))            # recreate vector of time series indices (because for-loop killed it)
k                          # display the indices of the time series
round(Tnew, digits = 5)    # Display the re-constructed time series of humidities
round(T, digits = 5)       # Compare with the original time series

# === Plot the re-created data ===
plot(t ,Tnew ,type="l", col="blue", main="Inverse DFT to Re-create Data", xlab="t (s)",
     ylab="T_new (degC)", xlim=c(range(t, na.rm=TRUE)), ylim = c(range(Tnew, na.rm=TRUE)) )
# abline(h=0.)      # draw horizontal line indicated bottom boundary (at y = 0)


#=== Compute the spectrum ===

nyquist <- as.numeric(N)/2     # Nyquist frequency
nyquist                        # Display the nyquist frequency

Fsq <- (Fr*Fr) + (Fi*Fi)       # compute the spectral energies
Fsq                            # display the energies

the.mean <- Fr[1]              # the mean of the time series is just the first real DFT component
the.variance <- sum(Fsq[2:N])  # the variance is the sum of all the non-mean energies of the DFT

the.mean                       # display the mean of the time series
the.variance                   # display the variance of the time series

#Fold the energies from above the nyquist down to below the nyquist
#nyquist.index = 1 + as.integer(nyquist)   #remember that vector indices start at 1 in R
#E <- 2*Fsq[2:nyquist.index]   #computes spectral energy up to truncated value of nyquist

# But if the nyquist happens to = an integer, then unfold it from the previous calculation.
#if ((N %% 2) == 0 ) E[nyquist.index-1] <- E[nyquist.index-1] - Fsq[nyquist.index] # for N = even
#E                              # display the results for E

# alternative method to do folding, start at beginning and end
E <- rep(0, floor(N/2))      # initialize spectrum to zero
i <- 2            # start at beginning to work forward
j <- N            # start at end and work backward
while (i < j) {
  E[i-1] <- Fsq[i] + Fsq[j]
  i <- i + 1     # increment
  j <- j - 1     # start at end and work backward
}
if (i == j)  E[i-1] = Fsq[i]
E                # display E


# Check of total variance of E, should equal the population variance of the time series
the.variance <- sum(E)
the.variance

# Plot the resulting discrete spectrum
barplot(E, axis.lty=1, names.arg=c(1:floor(N/2)), xlab = "n (wavenumber = waves per domain)",
        ylab = "E (contrib.to total var.)",
        main="Energy Spectrum", col="palegreen")

# Convert from wavenumbers (n) to multiples of delt (s).
m <- N/c(1:nyquist)     # a vector of m values.  Example: m = 2 is a 2 delt waveperiod

# Plot the resulting discrete spectrum as semi-log bar plot vs m
barplot(E, axis.lty=1, names.arg= round( N/c(1:floor(N/2)), digits=1  ),
        xlab = "m (where m*del_t = waveperiod)",
        ylab = "E (contrib.to total var.)",
        main="Energy Spectrum", col="palegreen")

# ======================================
# can stop here.  The next plot is semi-log, for full turb spectra
# Plot the resulting discrete spectrum as semi-log bar plot
#barplot(E, log="y", axis.lty=1, names.arg=c(1:floor(N/2)), xlab = "n (wavenumber = waves per domain)",
#        ylab = "E (contrib. as semi-log)",
#        main="Energy Spectrum", col="palegreen")
```