Data Overview S2

Phil Shea

9 September 2024

```r
library( parallel)
# Calculate the number of cores
num_cores <- detectCores() - 1
 source("~/DSP/MD/R/runcombine.r")
```

Data Creation

This is the second run, which has the Nelson Halperin Order Parameter calculated, and also has the initial angular momentum zero'd out (this was not done with the S1 data). Note that angular momentum is not conserved in periodic boundary conditions, but the random initialization could give the . The data was generated by the following Julia script executed on 21 August 2024. Note that the `Threads` section was executed by hand as there was an error in the original script.

```
include("C:\\Users\\phils\\Documents\\DSP\\MD\\dev\\MD2\\src\\MD2.jl")
using .MD2
using Printf
using Base.Threads
using Dates
using FLoops


origin = pwd()

td = "H:\\Data\\DSP\\MD\\Data\\1980"
cd( td)

pwd()

function run3(td, t, initsteps, steps)
    cd( td)
    temp = t/1000
    basename = @sprintf "S2.%.4d" t
    io = open( basename * ".txt", "a")
    println( io, "Starting Run $basename: $(now())")
    MD2.InitSystem( basename * ".0", temp=temp, io=io)
    MD2.RunSystem( basename * ".0", basename * ".1", initsteps, io=io)
    println( io, "Init steps finished: $(now())")
    @time MD2.RunSystem( basename * ".1", basename * ".2", steps, io=io)
    println( io, "First Run finished: $(now())")
    @time MD2.RunSystem( basename * ".2", basename * ".3", steps, io=io)
    println( io, "Second Run finished: $(now())")
    close( io)
    return (initsteps + 2 * steps)
end

function scriptrun_serial()
    steps = 5000
    initsteps = 1000
    for t in 1:10
        run3( td, t, initsteps, steps)
    end
end

function scriptrun_floop()
    steps = 5000
    initsteps = 1000
    @floop for t in 11:20
        run3( td, t, initsteps, steps)
    end
end

@time scriptrun_serial()

@time scriptrun_floop()

function scriptrun_thread()
    steps = 5000
    initsteps = 1000
    Threads.@threads for t in 21:30
        run3( td, t, initsteps, steps)
    end
end

@time scriptrun_thread()

cd( origin)
```

Get the list of files.

```
setwd("H:/Data/DSP/MD/Data/1980")
getwd()
```

```
## [1] "H:/Data/DSP/MD/Data/1980"
```

```
files = Sys.glob( file.path( "S2.*.scalars.csv"))
length(files)
```

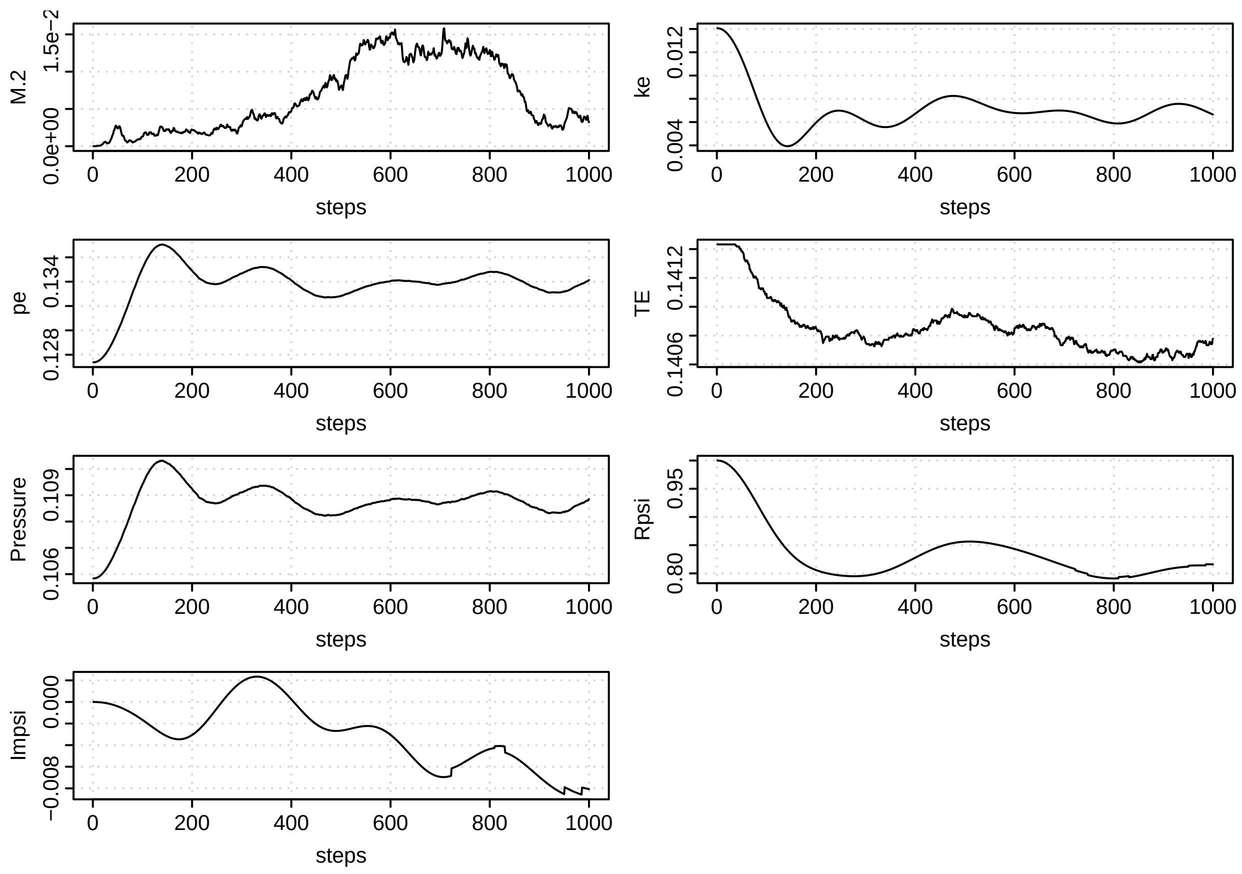
```
## [1] 300
```

The following will read and plot a file.

```
setwd("H:/Data/DSP/MD/Data/1980")
i <- 40
files[i]
```

```
## [1] "S2.0014.0.scalars.csv"
```

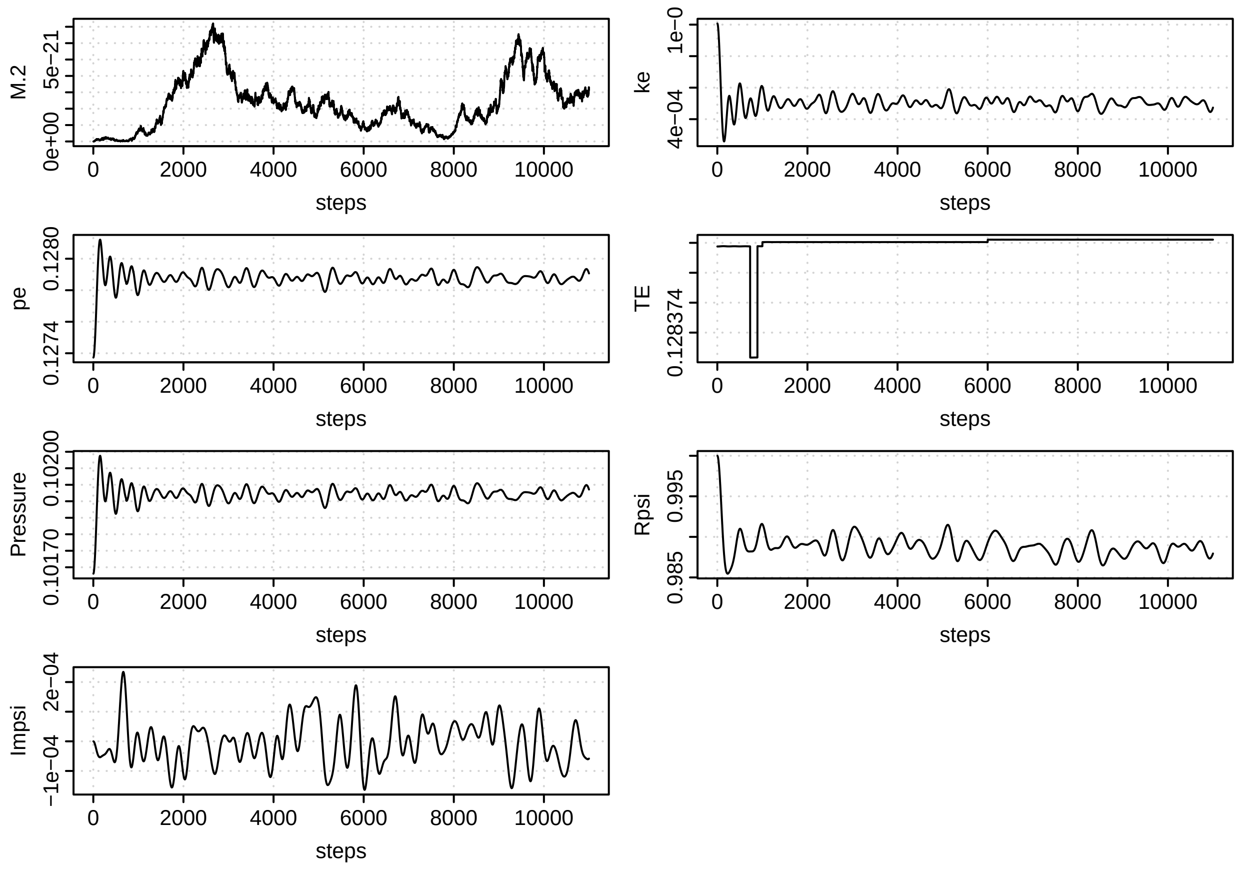
```
readplot( files[i])
```



```
setwd("H:/Data/DSP/MD/Data/1980")
x = combineruns(  ".", "S2\\.0001\\.", verbose=TRUE)
```

```
## Searching Directory:  .  for pattern: S2\.0001\..(.*).scalars.csv
## Files found:  S2.0001.0.scalars.csv S2.0001.1.scalars.csv S2.0001.2.scalars.csv
```

```
plotMD2( x)
```



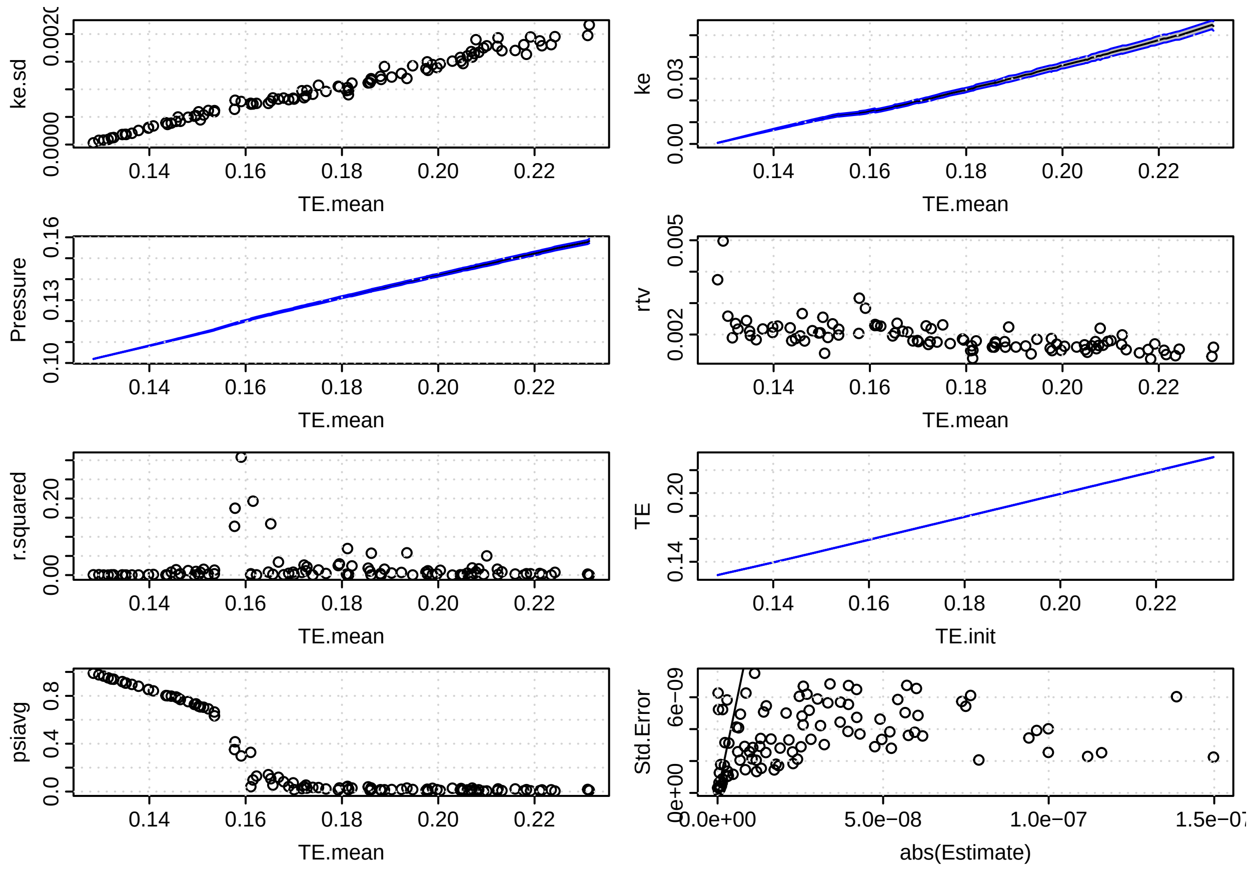
The momentum is moving around a bit, but that it runs up and back down again is a good sign. Perhaps more concerning is the change in total energy.

The database can be created with the following:

```
setwd("H:/Data/DSP/MD/Data/1980")
MD2DF <- createdf(  ".", files, clusters=num_cores)
save( MD2DF, file="MD2DF.S2.Rdata")
```

Now that we have the data, we can plot it.

```
op=  par( mfrow = c(4, 2), mar = c(3, 3, 0.5, 0.5), tcl = -0.3,
        mgp = c(1.7, 0.4, 0) )
plotScalars( MD2DF)
```



```r
par(op)
```

Extending Temperature Runs

So something is going on around 0.16, but we didn't run hot enough. Updated `sript2.jl` to `script3.jl` below:

```julia
function scriptrun_floop(trange)
    steps = 5000
    initsteps = 1000
    @floop for t in trange
        run3( td, t, initsteps, steps)
    end
end

scriptrun_floop(31:100)
```

This took about three and a half minutes to execute 70 runs. Rather than fool around with adding to the database, we'll just re-create it.

```r
setwd("H:/Data/DSP/MD/Data/1980")
files = Sys.glob( file.path( "S2.*.scalars.csv"))
length( files)
```

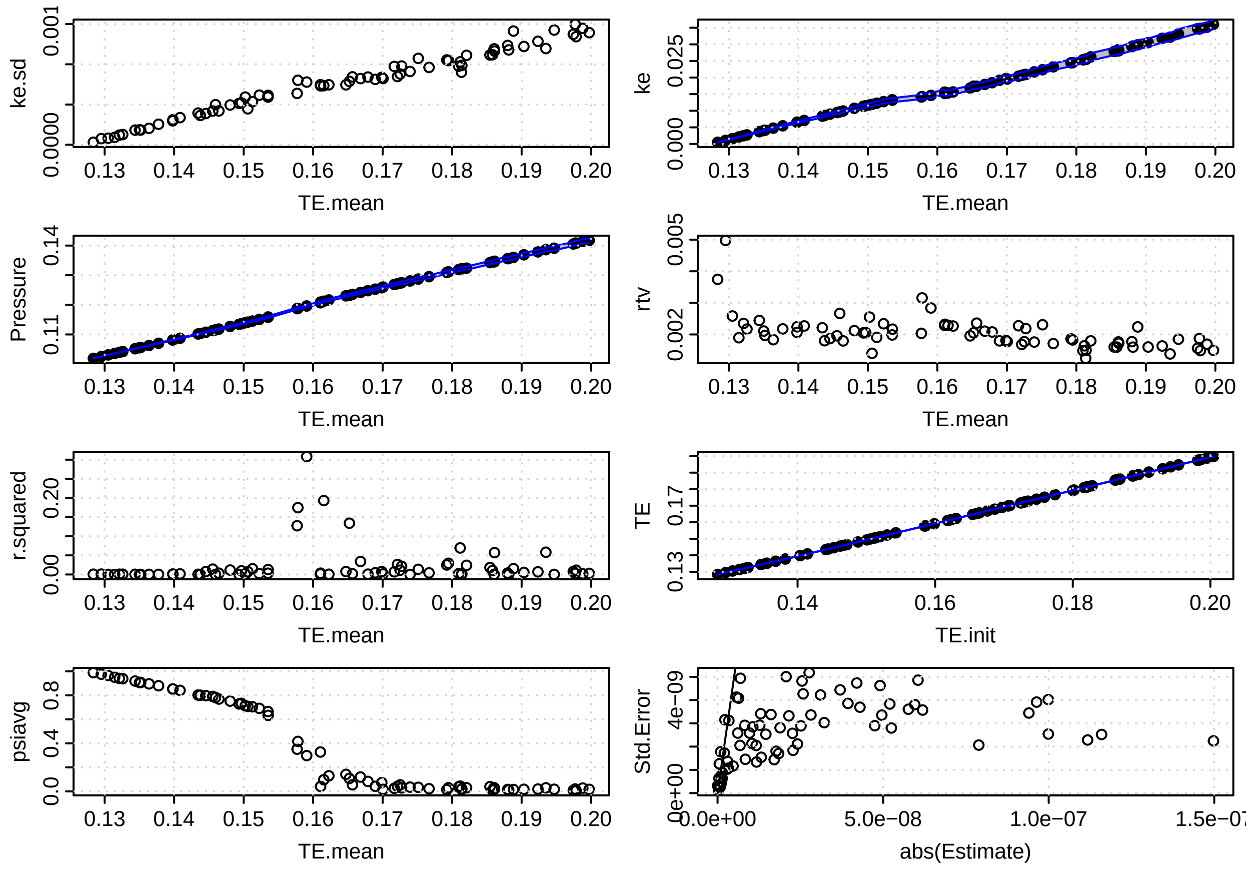
```
## [1] 300
```

```
MD2DF.2 <- createdf(  ".", files, clusters=num_cores)
save( MD2DF.2, file="MD2DF.S2.2.Rdata")
```

Data Analysis

Below is a plot of some of the main parameters of the averaged data.

```
op=  par( mfrow = c(4, 2), mar = c(3, 3, 0.5, 0.5), tcl = -0.3,
       mgp = c(1.7, 0.4, 0) )
plotScalars( subset( MD2DF.2, TE.mean< 0.2))
```



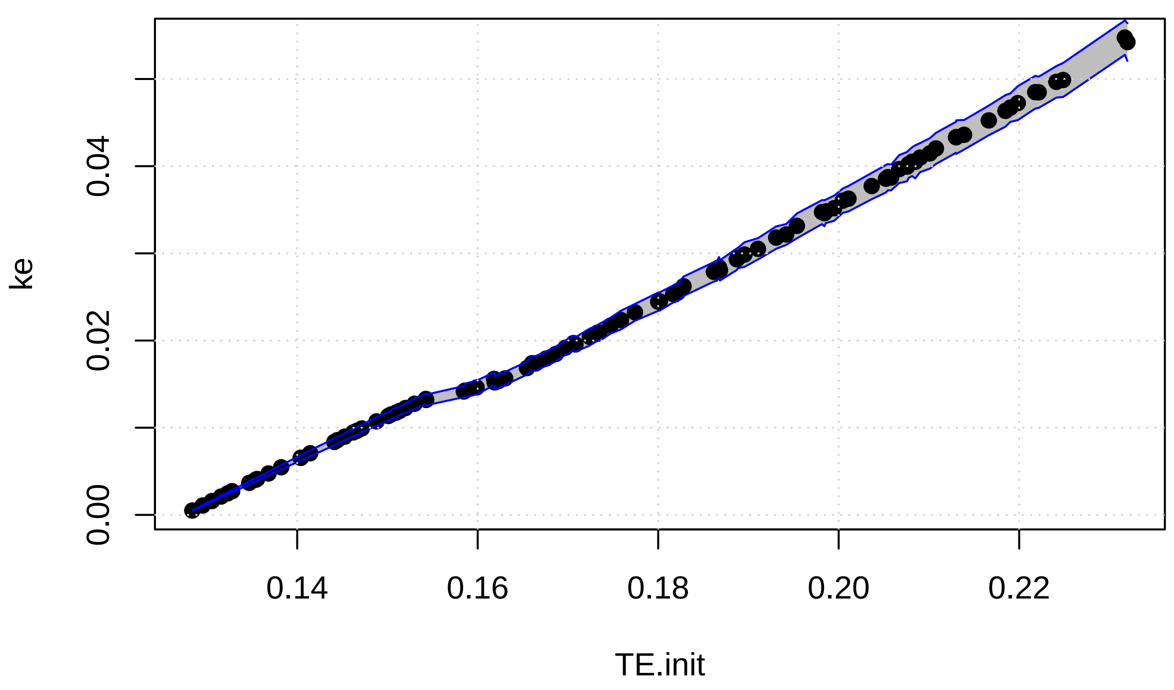
```
par(op)
```

A few things are apparent here. One, while the initial kinetic energy was increased incrementally, the total energy is not increasing incrementally (note the gaps in the first three plots above). Two, the relative temperature variance (`rtv`) is pretty high in the first two runs, and also around TE = 0.16, the second near a gap. Three, the temperature (`ke` in the above) and pressure may have a break around TE = 0.14.

The temperature (really just the kinetic energy, `ke`) and pressure have a clear break, and the total energy cannot be reached between 0.16 and 0.165. We can see that adding energy between 0.155 and 0.165 is difficult.

The plot below though clearly shows a break where adding energy does not increase the temperature.

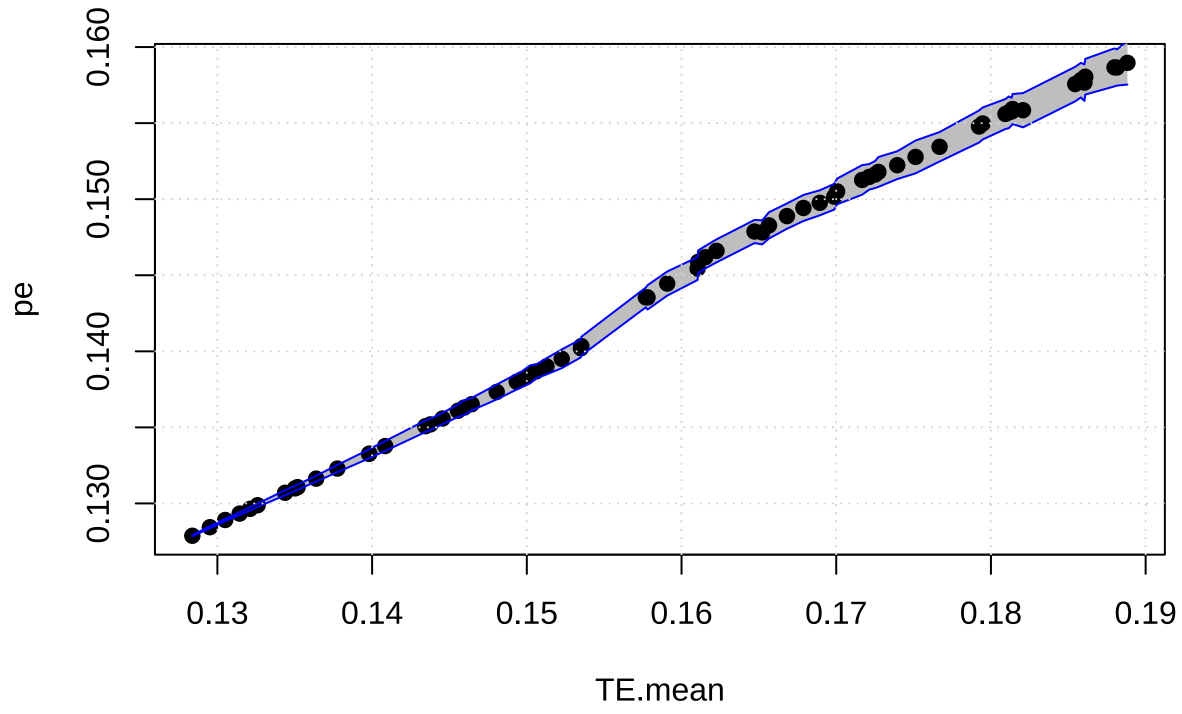
```
plotconfMD2( x="TE.init", y="ke", df = MD2DF.2, linethresh=110)
```



```
  # plot( Pressure.mean ~ TE.mean, data = MD2DF, ...)
```

The potential energy increases in the same region.

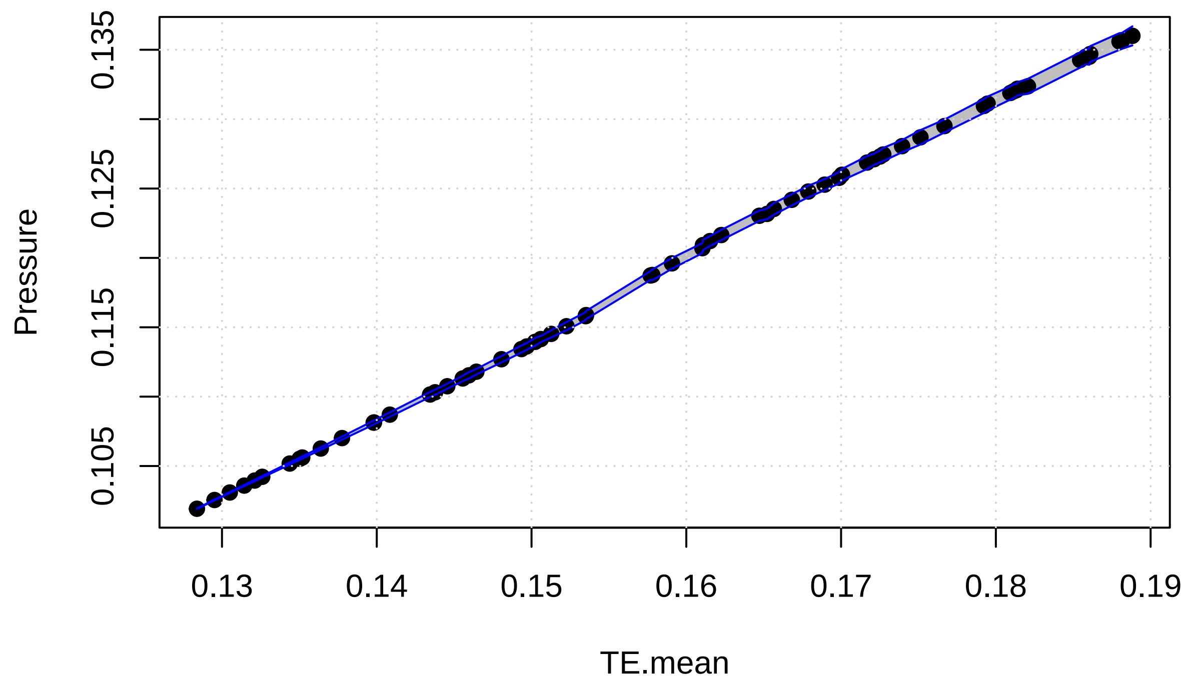
```
plotconfMD2( y="pe", df = MD2DF.2[MD2DF.2$TE.mean < 0.19,])
```



```
  # plot( Pressure.mean ~ TE.mean, data = MD2DF, ...)
```

Perhaps oddly, the pressure shows a slight break.

```
plotconfMD2( y="Pressure", df= MD2DF.2[MD2DF.2$TE.mean < 0.19,])
```

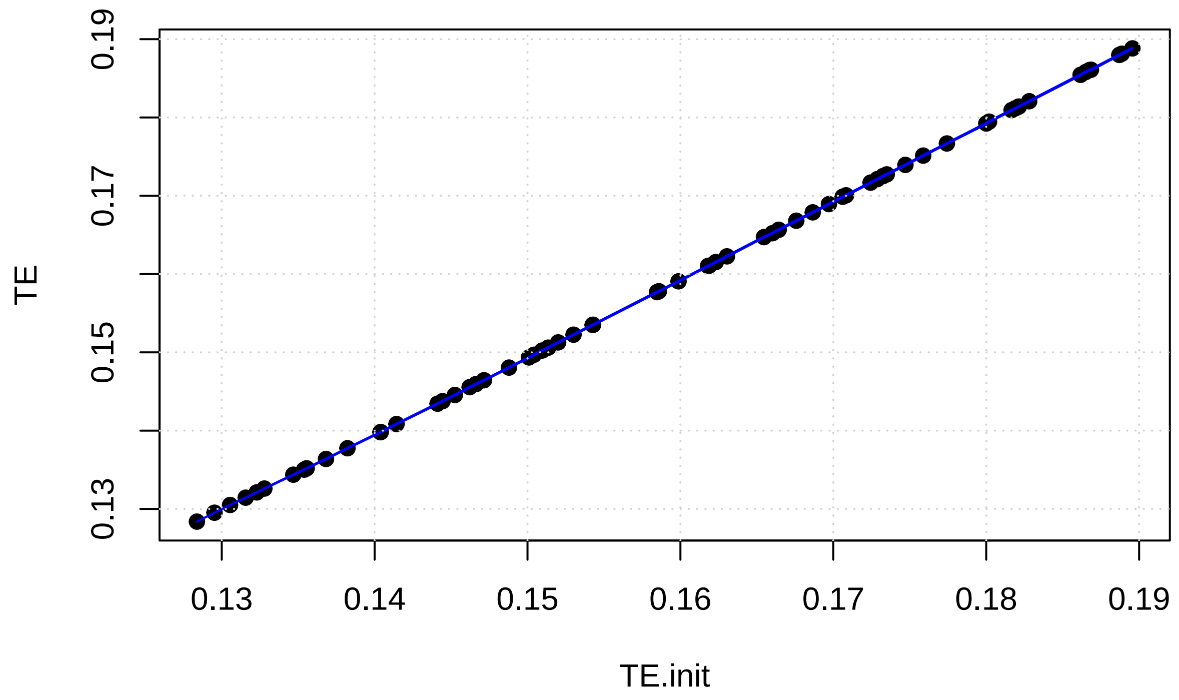


```
# plot( Pressure.mean ~ TE.mean, data = MD2DF, ...)
```

Total Energy Variances

The following graph shows that the even though the there seemed to be an initial decrease in the total energy in the first several hundred steps, this decrease is small relative to the energy. The graph below, is like the other graphs, in that it shows the 1σ confidence region. The gaps may be attributed to the random nature of the initialization.

```
plotconfMD2( df=MD2DF.2[MD2DF.2$TE.mean < 0.19,], x="TE.init", y="TE")
```



The relative energy loss is likely due to the second nearest neighbor distance ($\sqrt{3}a \approx 3.3$), where $a \approx 1.905$ is the grid spacing) being near the skin depth of the potential (3.5). While the average density is π, so the average number of particles within the skin depth is $3.5^2 = 12.25$, very cold systems have exactly 13 particles within each particles skin (including the subject particle). Thhe initialization starts the system in the grid position, but the hot systems quickly get outside that. A future investigation may look at moving the skin out a bit further.

The plot below shows the energy loss as a percent of the total energy, with the energy standard deviation asa percentage of the total energy.

```
plot( x = MD2DF.2[, 'TE.init'],
      y = 100 * (MD2DF.2[, 'TE.init'] - MD2DF.2[, 'TE.mean']) /  MD2DF.2[, 'TE.init'],
      xlab='TE.init', ylab='% loss', main="Energy Loss", panel.first=grid())
points(  x = MD2DF.2[, 'TE.init'],
         y = 100 * MD2DF.2[, 'TE.sd'] /  MD2DF.2[, 'TE.init'], col='red')
```

Energy Transfer to from Kinetic to Potential

The system is initialized with all the points in the lattice positions (i.e., minimum potential energy) and given an initial velocity. The x and y velocity components are drawn from a normal distribution with zero mean and variance corresponding to an input "temperature" (in quotes because it is never the final temperature). Energy is immediately transferred from kinetic to potential, and the below will plot the percentage of kinetic energy that ends up as potential (at least from the averages).

The nearest neighbors are a distance $a = \sqrt{\pi/\sin(\pi/3)} \approx 1.90463$. The second nearest neighbors, of which there are six, are at a distance of $a\sqrt{3}$. The initial potential energy per particle is is one half the potential

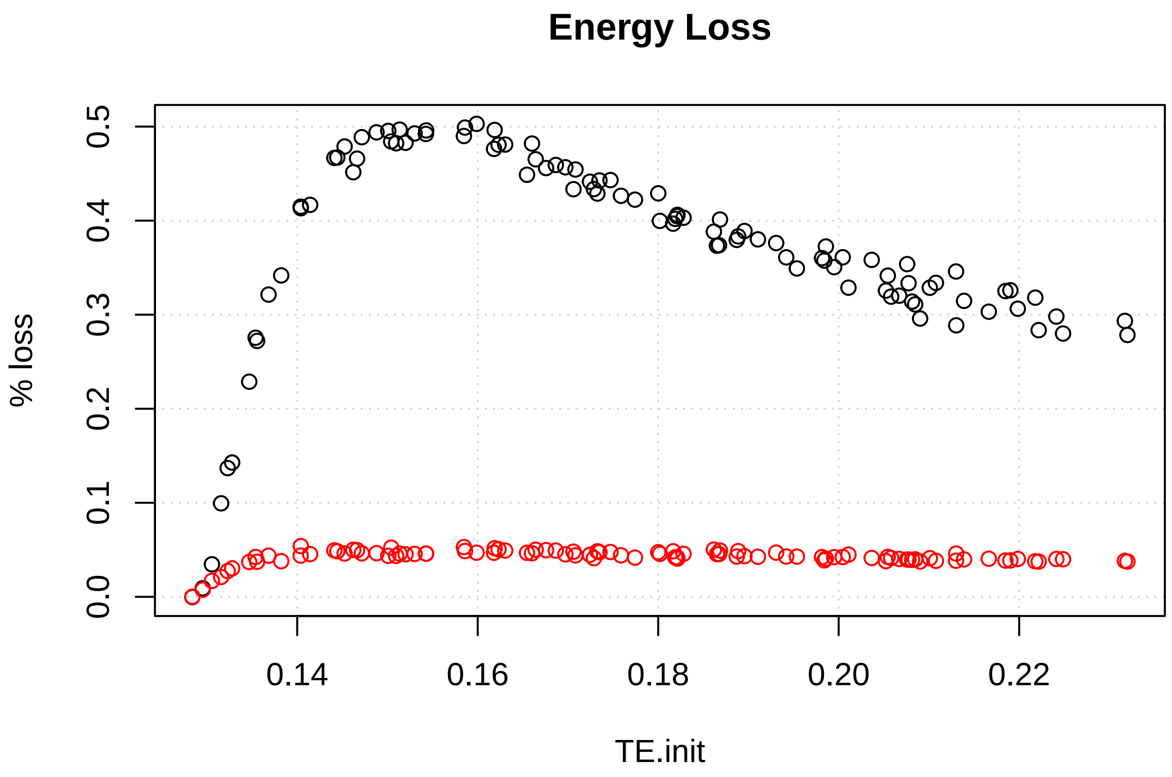


Figure 1: Energy loss (black) and standard deviation (red) as a percent of the total energy.

energy between each pair of particles. There are twelve such pairs at initialization: six nearest neighbors, and six next-nearest neighbors.

```r
(a <- sqrt( pi / sin(pi/3)))
```

```
## [1] 1.904626
```

```r
(b <- a * sqrt(3))
```

```
## [1] 3.298908
```

```r
(6/2) * ((a^-5) + (b^-5))
```

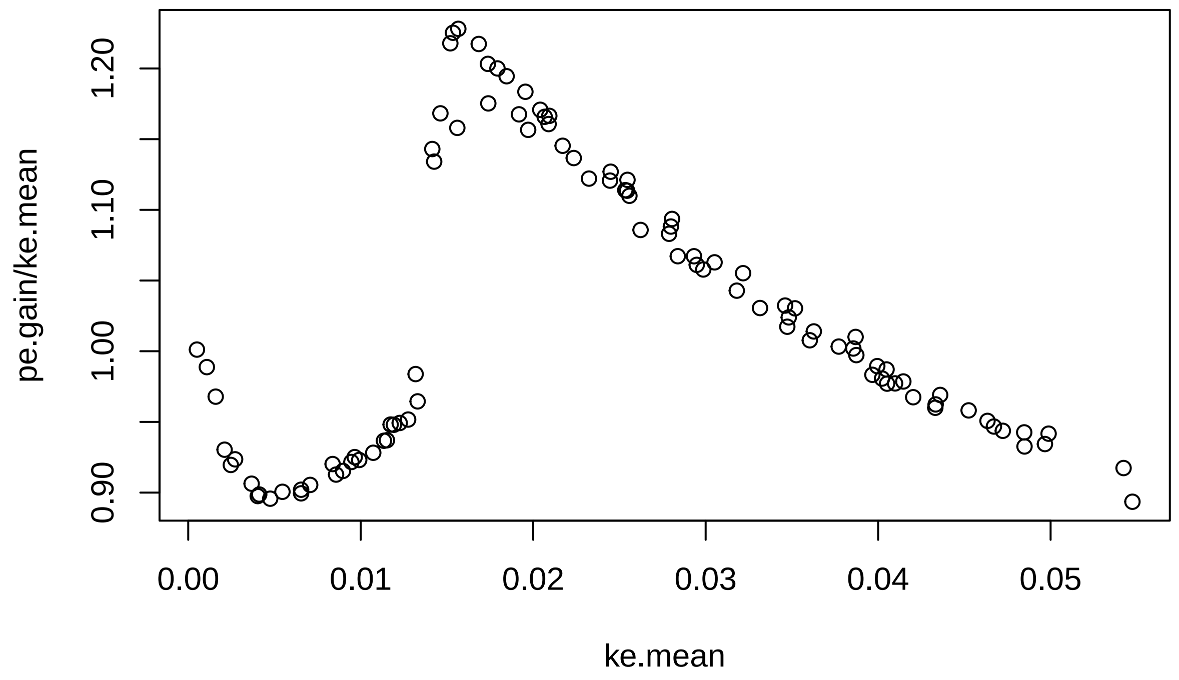
```
## [1] 0.1273726
```

```r
x[1,'pe']
```

```
## [1] 0.1273726
```

We can use the `TE.init` value and the above to compute how much kinetic energy was initially put into the system (i.e., `TE.init - 0.127326`), and compare that to the final averages.

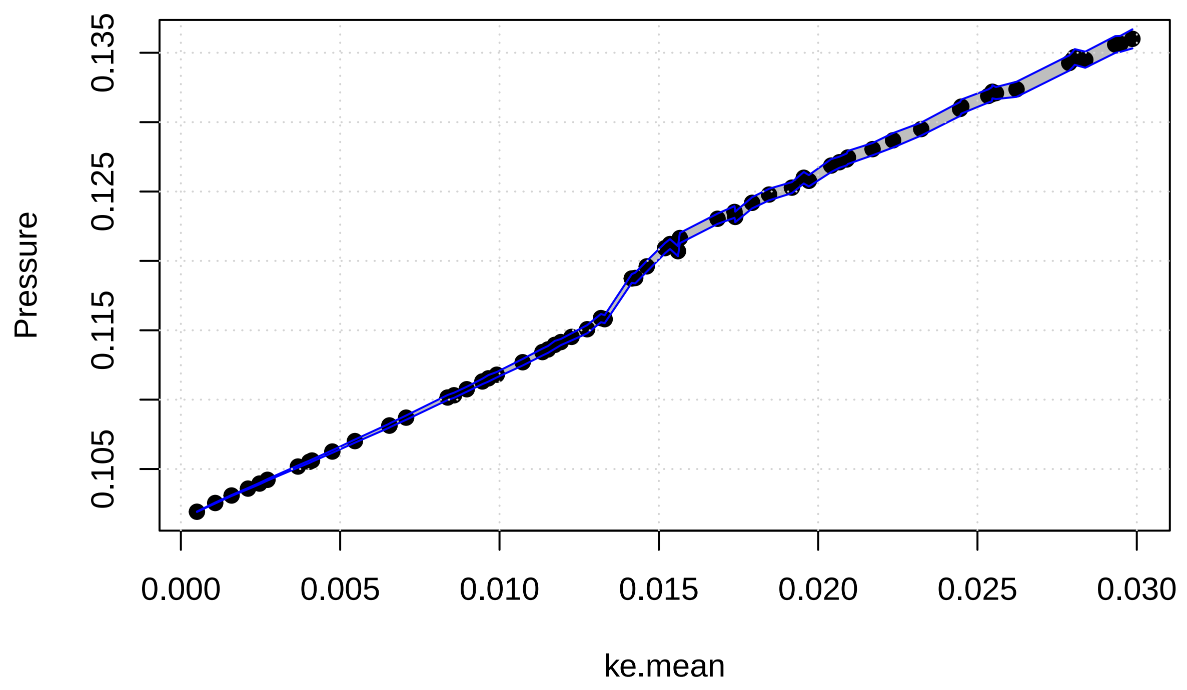
```r
# recompute here to keep it together.
a <- sqrt( pi / sin(pi/3))
b <- a * sqrt(3)
pe.init <- (6/2) * ((a^-5) + (b^-5))
MD2DF.2$pe.gain <- MD2DF.2$pe.mean - pe.init
plot( pe.gain/ke.mean ~ ke.mean, data=MD2DF.2)
```



Phase Transition

The pressure should show a break when plotted against temperature.

```
plotconfMD2( x="ke.mean", y="Pressure", df= MD2DF.2[MD2DF.2$TE.mean < 0.19,])
```



```
  # plot( Pressure.mean ~ TE.mean, data = MD2DF, ...)
```

The pressure is given by the following calculation. It is evident that the mean of the step-by-step pressure is identical to the pressure calculated from the mean kinetic and potential energies.

$$
\begin{aligned}
p &= \rho\langle k_e\rangle\left(1+\frac{n\langle U_T\rangle}{2\langle k_T\rangle}\right) \\
&= \rho\left(\langle k_e\rangle+\frac{n\langle U_T\rangle}{2N}\right) \\
&= \rho\left(\langle k_e\rangle+\frac{n\langle U_e\rangle}{2}\right)
\end{aligned}
$$

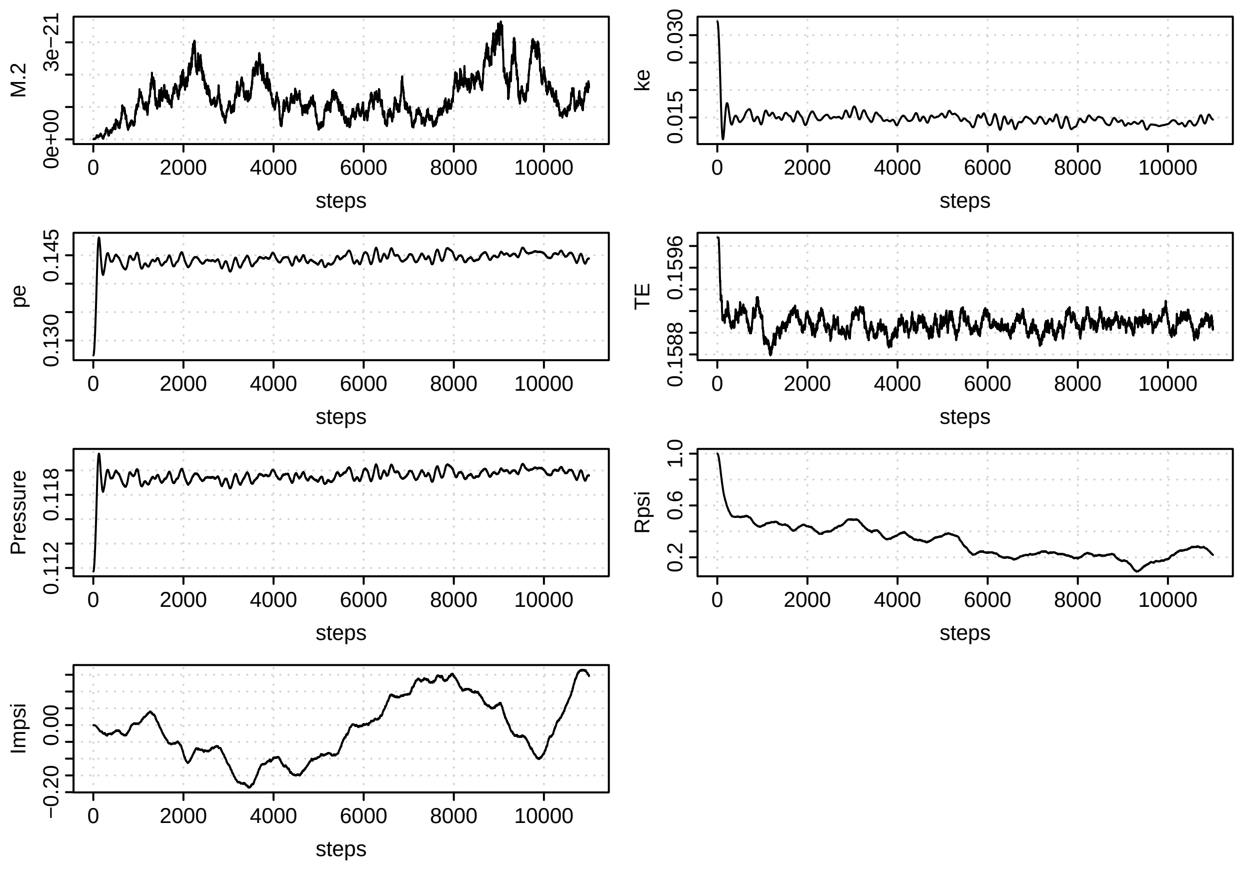
Worst Correlation

The run with the worst (i.e., the highest) regression coefficient is represented below. Note that while the `which.max` finds it the worst correlation, the file name is hand coded.

```
#setwd("H:/Data/DSP/MD/Data/1980")
which.max(MD2DF.2[,"r.squared"])
```

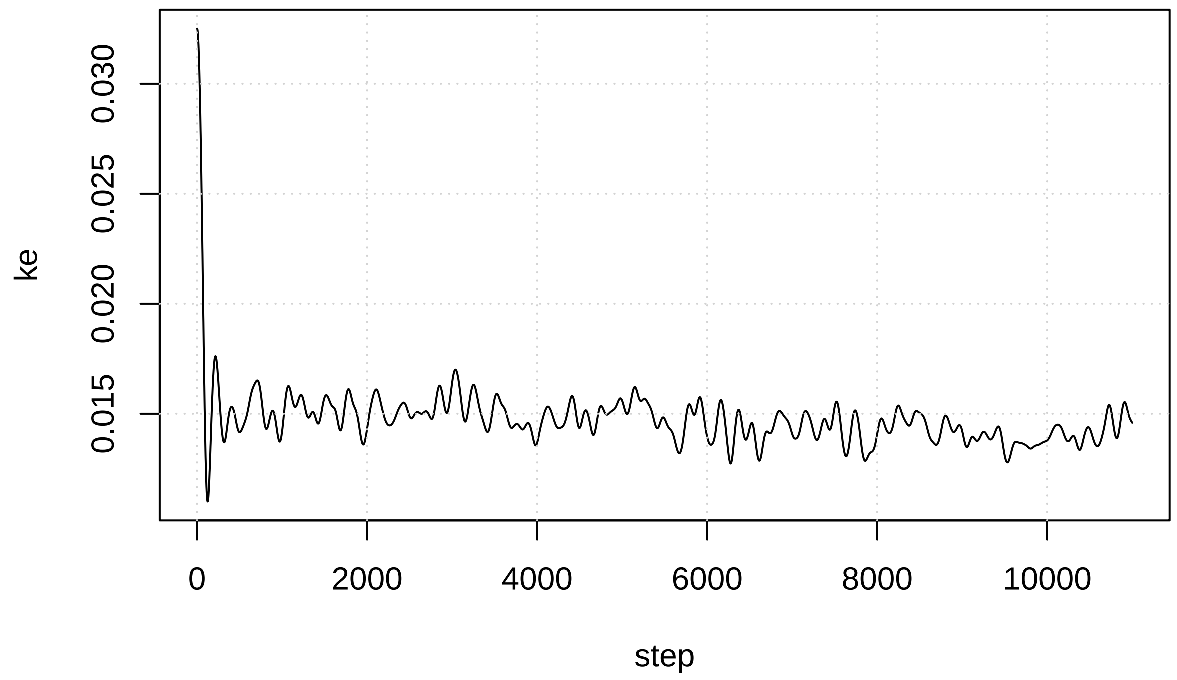
```
## [1] 36
```

```
x36 = combineruns(  "H:/Data/DSP/MD/Data/1980", "S2\\.0036\\.")
plotMD2( x36)
```



Below is the full kinetic energy plot.

```
plot( ke ~ step, data=x36, type='l')
grid()
```



It is possible that the large fluctuation preceding step 2,000 caused the correlation. The `createdf` function will skip 1,000 steps by default.

```
x36lm <- lm( ke ~ step, tail( x36, -2000))
summary( x36lm)
```

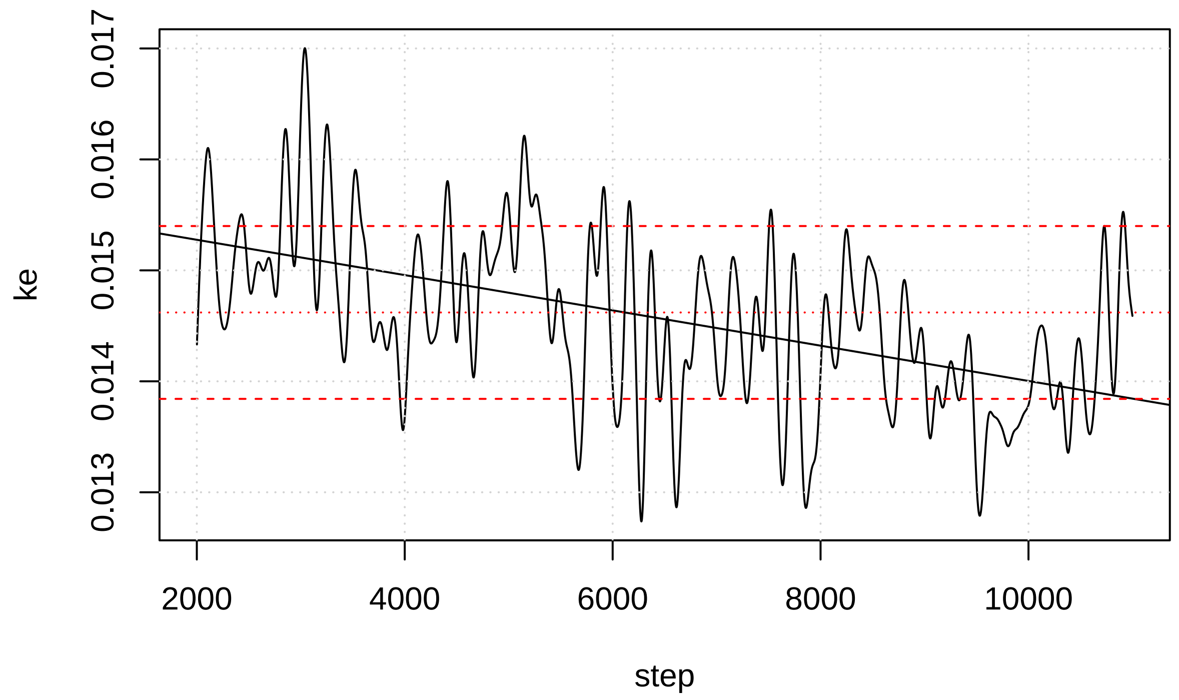
```
##
## Call:
## lm(formula = ke ~ step, data = tail(x36, -2000))
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -1.858e-03 -4.330e-04 -2.695e-05  4.407e-04  1.892e-03
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.559e-02  1.842e-05  846.45   <2e-16 ***
## step        -1.590e-07  2.632e-09  -60.44   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0006486 on 8998 degrees of freedom
## Multiple R-squared:  0.2887, Adjusted R-squared:  0.2886
## F-statistic:  3652 on 1 and 8998 DF,  p-value: < 2.2e-16
```

```
(k36 <- MD2DF.2[ 36, c('ke.mean', 'ke.sd')])
```

```
##       ke.mean        ke.sd
## 36 0.01462063 0.0007788672
```

The estimate is 60 times the standard error of the estimate. The plot below shows the regression line, along with the average computed by `createdf` (thus, only 1,000 points skipped, not the 2,000 below), along with $\pm 1\sigma$ in red.

```
plot( ke ~ step, data=tail( x36, -2000), type='l')
abline( coef=coef(x36lm))
abline( h = k36$ke.mean - k36$ke.sd, col='red', lty=2)
abline( h = k36$ke.mean, col='red', lty=3)
abline( h = k36$ke.mean + k36$ke.sd, col='red', lty=2)
grid()
```



There may be oscillations, which would be bad. The following is a spectral estimate. The frequency axis is in units of $1/\tau$.

```
source("~/DSP/R/Common/Common/common.R")
```

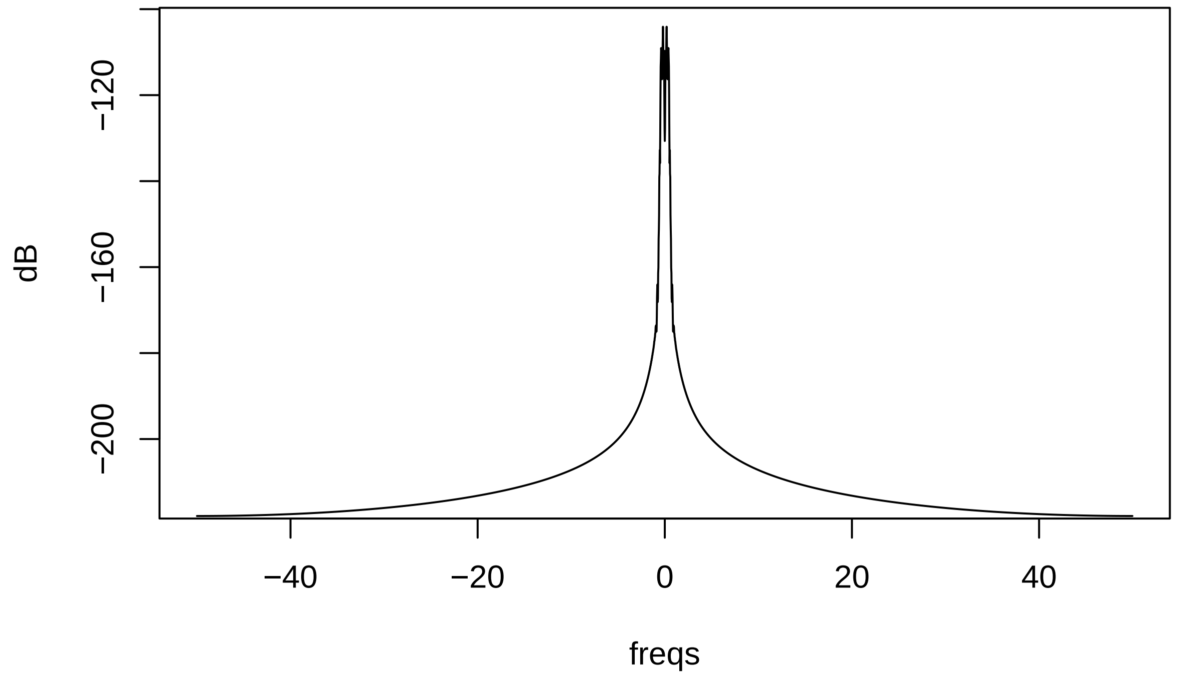
```
## Loading required package: pracma
```

```
## Loading required package: signal
```

```
##
## Attaching package: 'signal'

## The following objects are masked from 'package:pracma':
##
##     conv, ifft, interp1, pchip, polyval, roots

## The following objects are masked from 'package:stats':
##
##     filter, poly

## Loading required package: data.table
```

```
sx36 <- detrend( tail( x[, "ke"], -2000)) # remove mean and linear trend
x36sdf <- qsdf(sx36, blocksize=4096, yrange=110, dt=0.01)
```

```
## sdf: Total Size: 9000, Step: 2048, Number of FFTs: 3, window Power: 1535.62
```



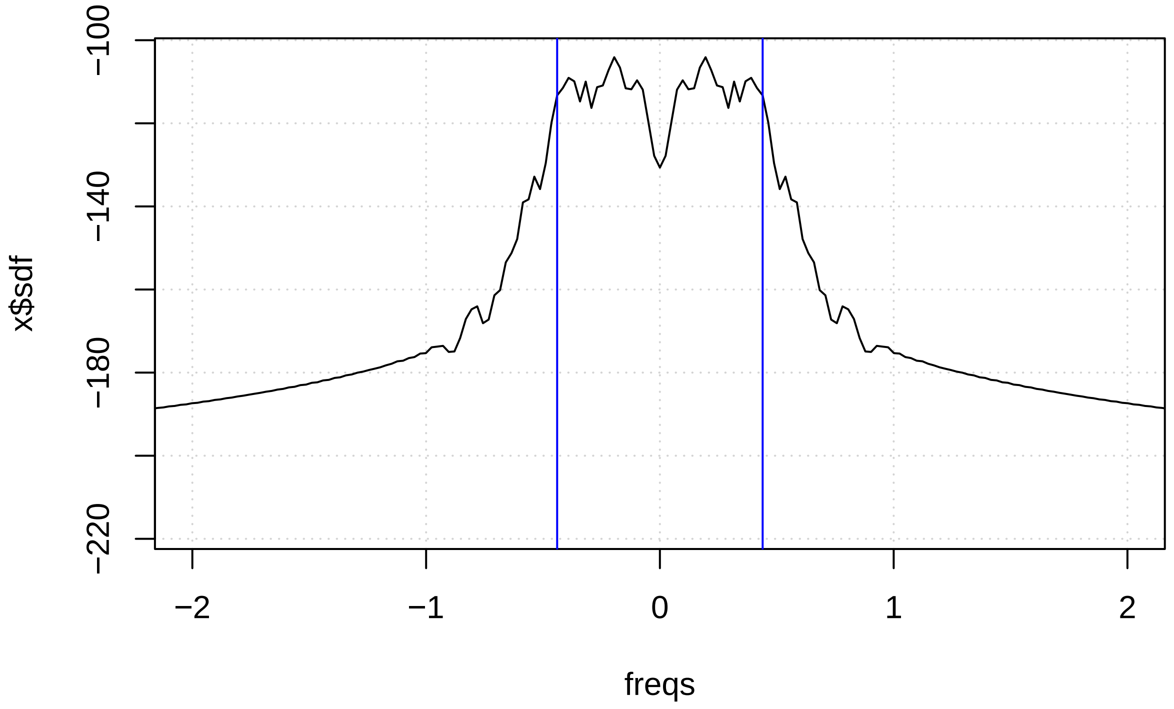
```
x36sdflin <- undB( x36sdf$sdf) # linear coefficients. qsdf returns Decibels
x36tp <- sum( x36sdflin) # total power.
x36sdfsum <- cumsum(x36sdflin) / x36tp
low36  <- which.max( x36sdfsum > 0.005)
high36 <- which.max( x36sdfsum > 0.995)
(freq36 <- x36sdf$freqs[c(low36, high36)])
```

```
## [1] -0.4394531  0.4394531
```

99% of the energy in the temperature spectrum is between the frequencies of -0.7324 to 0.7324 for a total main lobe width of $1.4648/\tau$. A block average will have a null-to-null bandwidth of twice the reciprocal of the block size, so since the time step is 0.01τ, those frequencies are the equivalent of about 137 samples. This is important because the we will want to sum independent samples of temperature so that the t test can be used with the proper number of samples.

Zooming in shows a broad band of low frequency noise. The vertical lines show the 99% power limit.

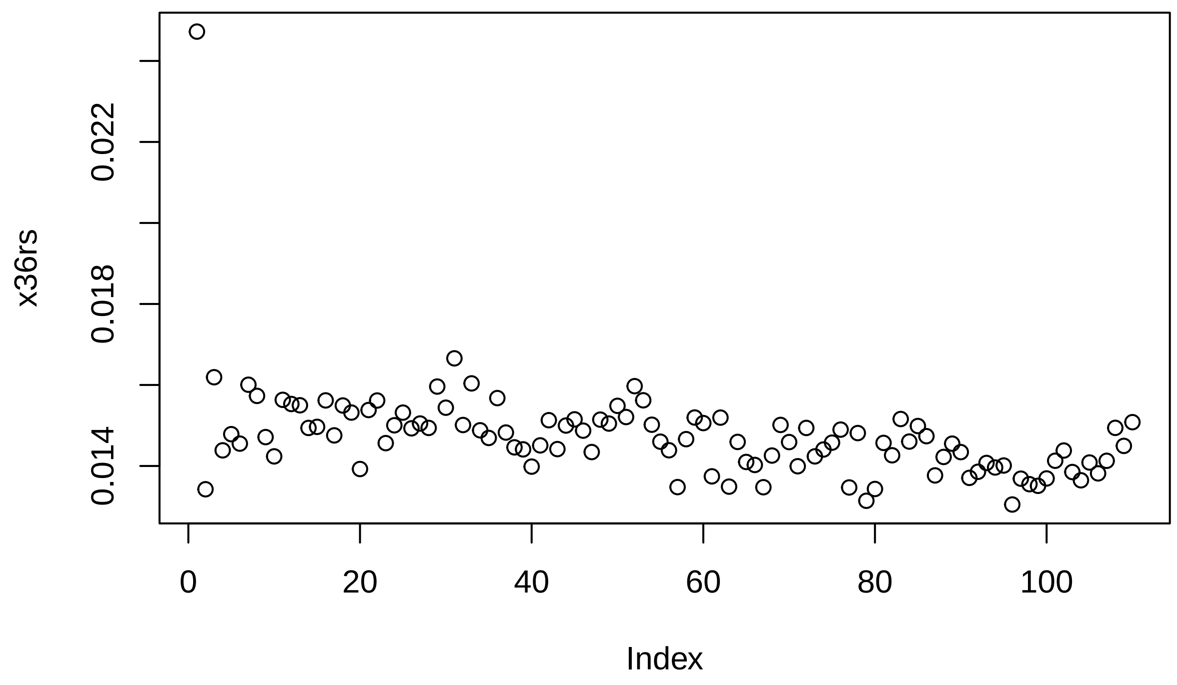
```
plot( x36sdf, xlim=c( -2, 2), panel.first=grid())
abline( v=freq36, col='blue')
```



Resampling

Here is how to re-sample. Since the thermodynamics are fundamentally averages, we forgo signal processing style re-sampling in favor of simple averages. We'll reshape the vector into an array of length n columns, truncating any remaining points.

```
rsmpavg <- function(x, n){
  # re-sample x by n point averages, returning one sample per average.
  # truncates if length of x is not multiple of n
  outlen <- floor( length(x) / n)
  x <- x[1:(n * outlen)]
  dim(x) <- c( n, outlen)
  apply( x, 2, mean) # essentially, column means
}
x36rs <- rsmpavg( x36$ke, 100)
plot( x36rs)
```

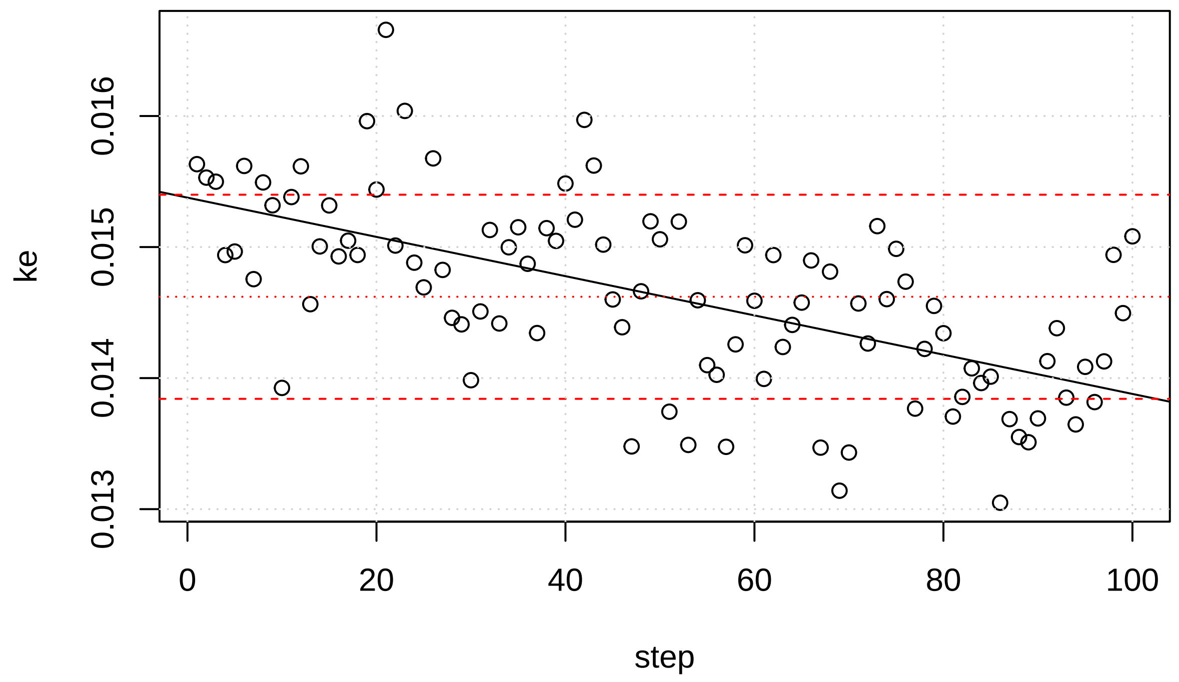


Below is the linear model fit skipping the first 1,000 steps. Note the t-value is only about 7 compared to the previous 60.

```
x36rsdf <- data.frame( step=1:(length(x36rs) - 10), ke=tail( x36rs, -10))
x36rslm <- lm( ke ~ step, x36rsdf) # skip 1,000 steps.
summary( x36rslm)
```

```
##
## Call:
## lm(formula = ke ~ step, data = x36rsdf)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -1.302e-03 -3.804e-04  9.250e-06  3.416e-04  1.595e-03
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.538e-02  1.144e-04 134.422  < 2e-16 ***
## step        -1.498e-05  1.967e-06  -7.616 1.66e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0005677 on 98 degrees of freedom
## Multiple R-squared:  0.3718, Adjusted R-squared:  0.3654
## F-statistic:    58 on 1 and 98 DF,  p-value: 1.658e-11
```

```
plot( ke ~ step, data=x36rsdf, type='p')
abline( coef=coef(x36rslm))
abline( h = k36$ke.mean - k36$ke.sd, col='red', lty=2)
abline( h = k36$ke.mean, col='red', lty=3)
abline( h = k36$ke.mean + k36$ke.sd, col='red', lty=2)
grid()
```



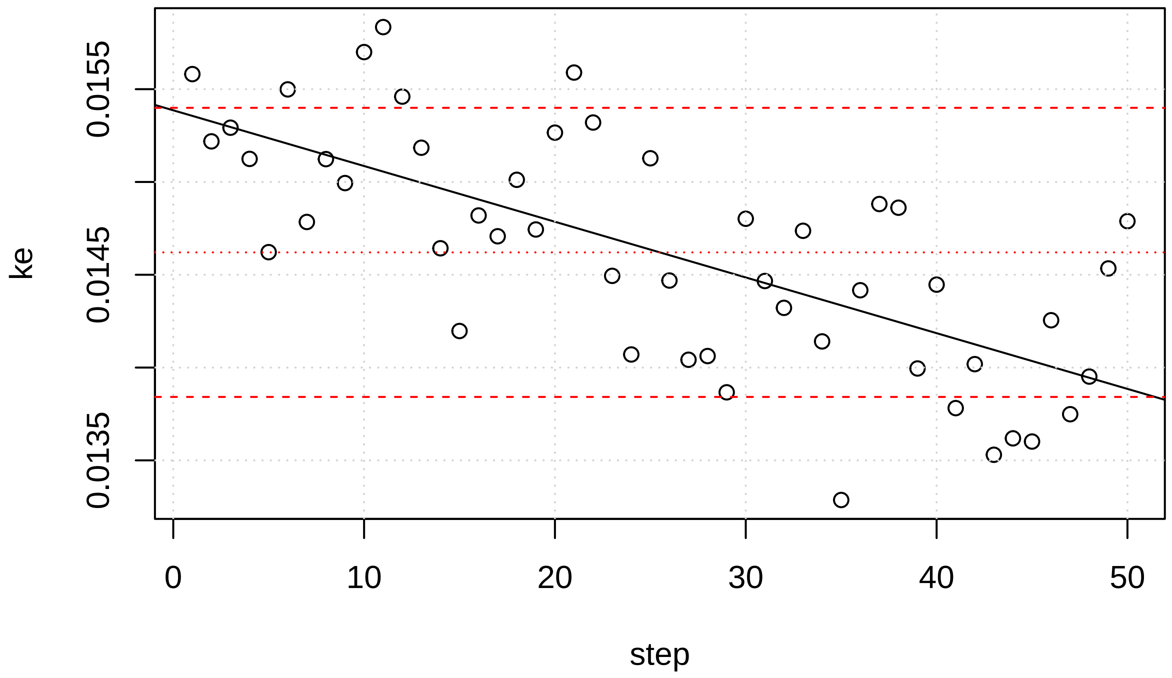
The samples still look pretty correlated, so lets try a 200 point average.

```
x36rs2 <- rsmpavg( x36$ke, 200)
x36rs2df <- data.frame( step=1:(length(x36rs2) - 5), ke=tail( x36rs2, -5))
x36rs2lm <- lm( ke ~ step, x36rs2df) # skip 1,000 steps.
(sx36rs2lm <- summary( x36rs2lm))
```

```
##
## Call:
## lm(formula = ke ~ step, data = x36rs2df)
##
## Residuals:
```

```
##        Min         1Q     Median         3Q        Max
## -1.049e-03 -2.990e-04 -7.909e-05  3.104e-04  9.037e-04
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.539e-02  1.307e-04 117.749  < 2e-16 ***
## step        -3.002e-05  4.460e-06  -6.732 1.89e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0004551 on 48 degrees of freedom
## Multiple R-squared:  0.4856, Adjusted R-squared:  0.4749
## F-statistic: 45.32 on 1 and 48 DF,  p-value: 1.894e-08
```

```r
plot( ke ~ step, data=x36rs2df, type='p')
abline( coef=coef(x36rs2lm))
abline( h = k36$ke.mean - k36$ke.sd, col='red', lty=2)
abline( h = k36$ke.mean, col='red', lty=3)
abline( h = k36$ke.mean + k36$ke.sd, col='red', lty=2)
grid()
```



The points look more random now, and the t statistic has come down a bit (6.7). The t statistic for the regression coefficient is (Edwards, eq. 6.1, pg. 68):

$$t = \sqrt{n-2}\frac{r}{\sqrt{1-r^2}}$$

```
(t36 <-sqrt(48) * sx36rs2lm$r.squared / sqrt( 1 - sx36rs2lm$r.squared))
```

```
## [1] 4.691439
```

```
pt(t36, df=48)
```

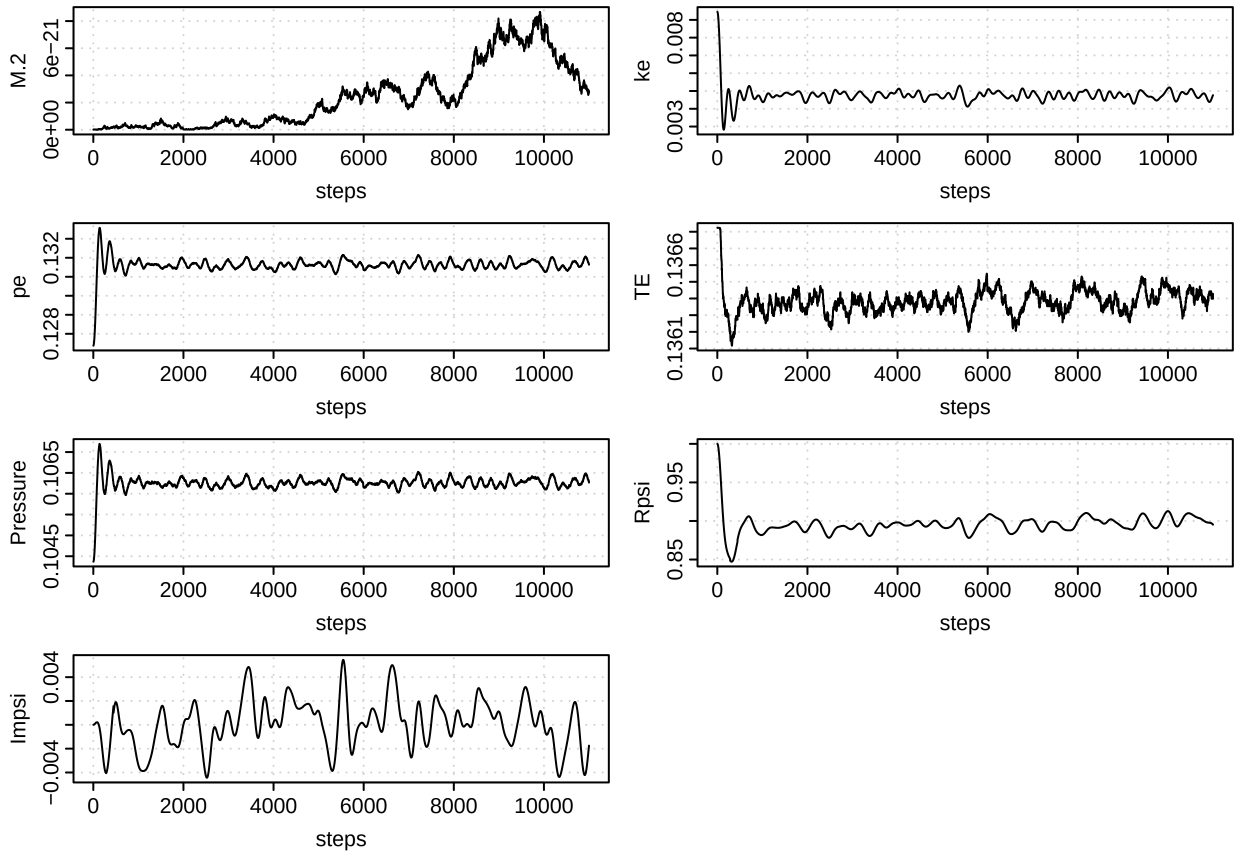
```
## [1] 0.9999886
```

So r^2 is significant at “four-nines”.

Comparison to Cold Temperature

Below we’ll look a a fairly cold run.

```
#setwd("H:/Data/DSP/MD/Data/1980")
x10 = combineruns(  "H:/Data/DSP/MD/Data/1980", "S2\\.0010\\.")
plotMD2( x10)
```



Here is the regression summary.

```
x10lm <- lm( ke ~ step, tail( x10, -1000))
summary(x10lm)
```

```
##
## Call:
## lm(formula = ke ~ step, data = tail(x10, -1000))
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -6.326e-04 -1.345e-04  1.684e-05  1.479e-04  5.470e-04
##
## Coefficients:
##              Estimate Std. Error  t value Pr(>|t|)
## (Intercept) 4.746e-03  4.695e-06 1010.949   <2e-16 ***
## step        1.209e-09  7.050e-10    1.714   0.0865 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0002035 on 9998 degrees of freedom
## Multiple R-squared:  0.0002939,  Adjusted R-squared:  0.0001939
## F-statistic: 2.939 on 1 and 9998 DF,  p-value: 0.08647
```

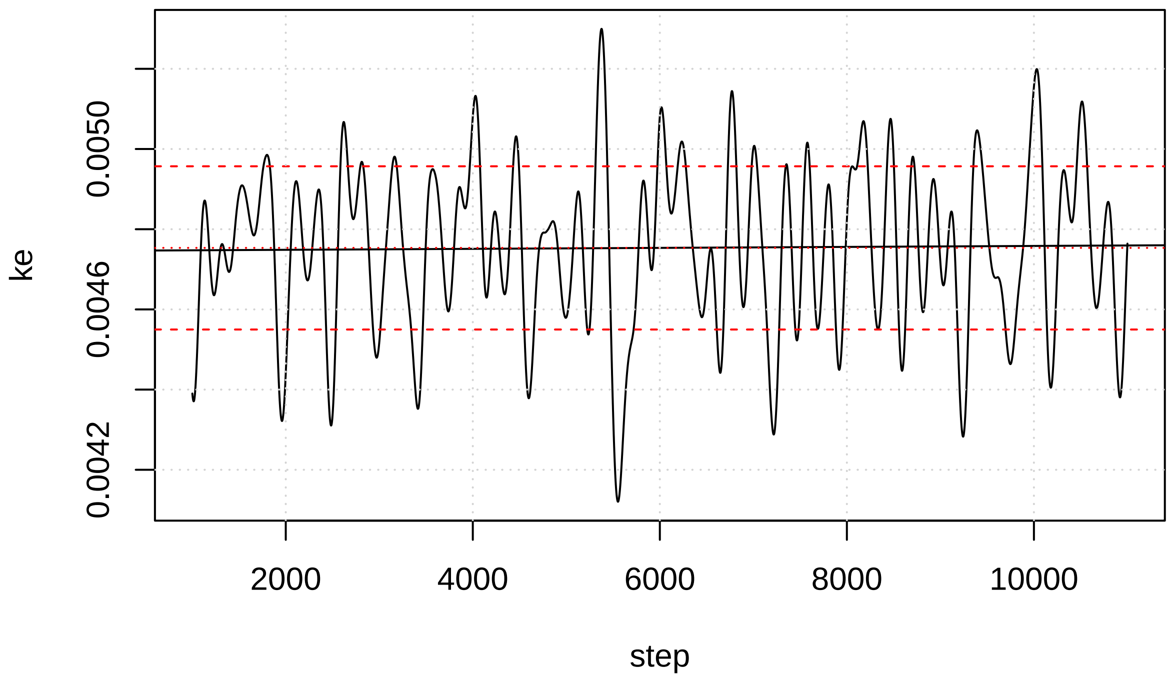
An here are the `createdf` results.

```
(k10 <- MD2DF.2[10,c('ke.mean', 'ke.sd', 'r.squared', 'Estimate', 'Std.Error')])
```

```
##        ke.mean        ke.sd    r.squared     Estimate    Std.Error
## 10 0.004753414 0.0002035495 0.0002939204 1.208798e-09 7.050479e-10
```

The system truly appears to be in equilibrium. Note the regression line lies on top of the red average line.

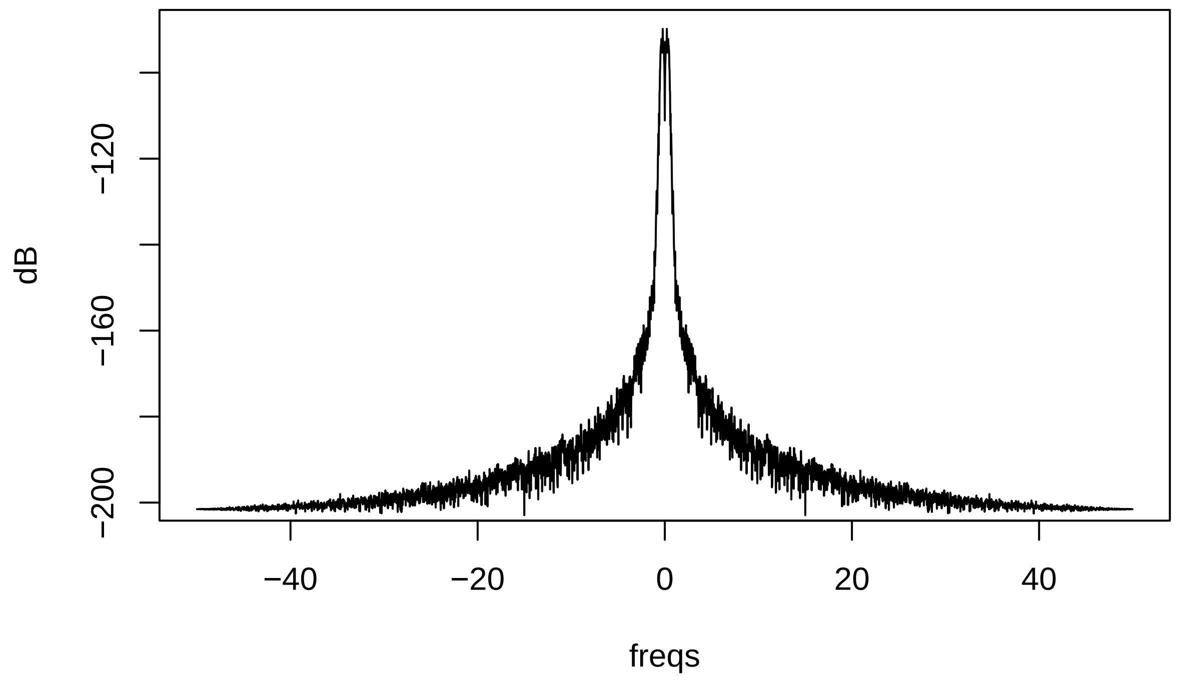
```
plot( ke ~ step, data=tail( x10, -1000), type='l')
abline( coef=coef(x10lm))
abline( h = k10$ke.mean - k10$ke.sd, col='red', lty=2)
abline( h = k10$ke.mean, col='red', lty=3)
abline( h = k10$ke.mean + k10$ke.sd, col='red', lty=2)
grid()
```



The spectrum looks very similar to the worst example.

```
sx10 <- detrend( tail( x10[, "ke"], -1000))
x10sdf <- qsdf(sx10, blocksize=4096, yrange=110, dt = 0.01)
```

```
## sdf: Total Size: 10000, Step: 2048, Number of FFTs: 3, window Power: 1535.62
```



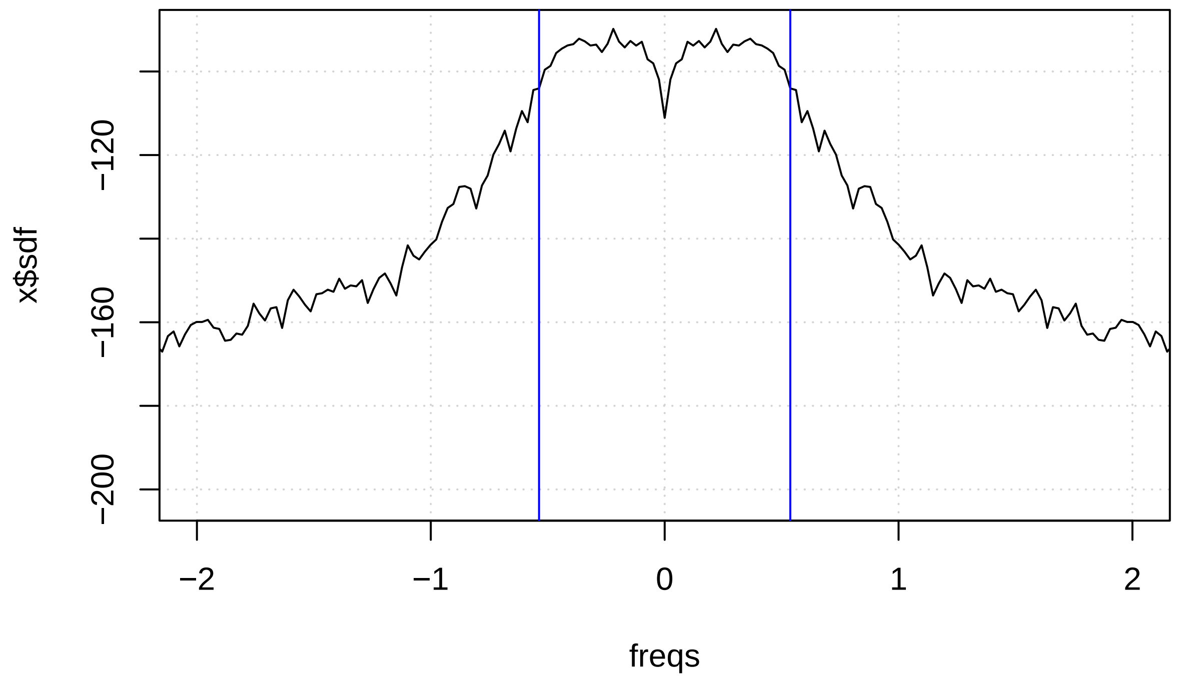
The 99% power points are a bit smaller

```
x10sdflin <- undB( x10sdf$sdf) # linear coefficients. qsdf returns Decibels
x10tp <- sum( x10sdflin) # total power.
x10sdfsum <- cumsum(x10sdflin) / x10tp
low10  <- which.max( x10sdfsum > 0.005)
high10 <- which.max( x10sdfsum > 0.995)
(freq10 <- x10sdf$freqs[c(low10, high10)])
```

```
## [1] -0.5371094  0.5371094
```

This corresponds to about 186 samples.

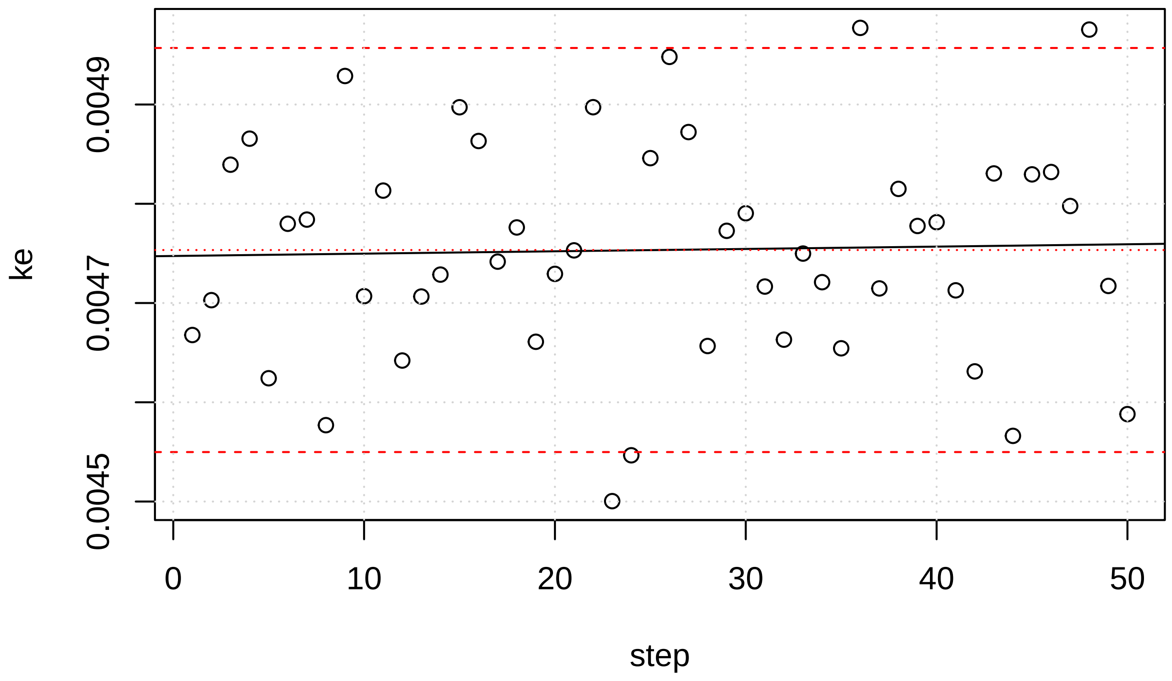
```
plot( x10sdf, xlim=c( -2, 2), panel.first=grid())
abline( v=freq10, col='blue')
```



```r
x10rs <- rsmpavg( x10$ke, 200)
x10rsdf <- data.frame( step=1:(length(x10rs) - 5), ke=tail( x10rs, -5))
x10rslm <- lm( ke ~ step, x10rsdf) # skip 1,000 steps.
summary( x10rslm)
```

```
##
## Call:
## lm(formula = ke ~ step, data = x10rsdf)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -2.524e-04 -7.115e-05 -2.326e-06  7.266e-05  2.213e-04
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 4.747e-03  3.207e-05 148.036   <2e-16 ***
## step        2.361e-07  1.095e-06   0.216     0.83
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0001117 on 48 degrees of freedom
## Multiple R-squared:  0.0009684,  Adjusted R-squared:  -0.01984
## F-statistic: 0.04653 on 1 and 48 DF,  p-value: 0.8301
```

```r
plot( ke ~ step, data=x10rsdf, type='p')
abline( coef=coef(x10rslm))
abline( h = k10$ke.mean - k10$ke.sd, col='red', lty=2)
abline( h = k10$ke.mean, col='red', lty=3)
abline( h = k10$ke.mean + k10$ke.sd, col='red', lty=2)
grid()
```

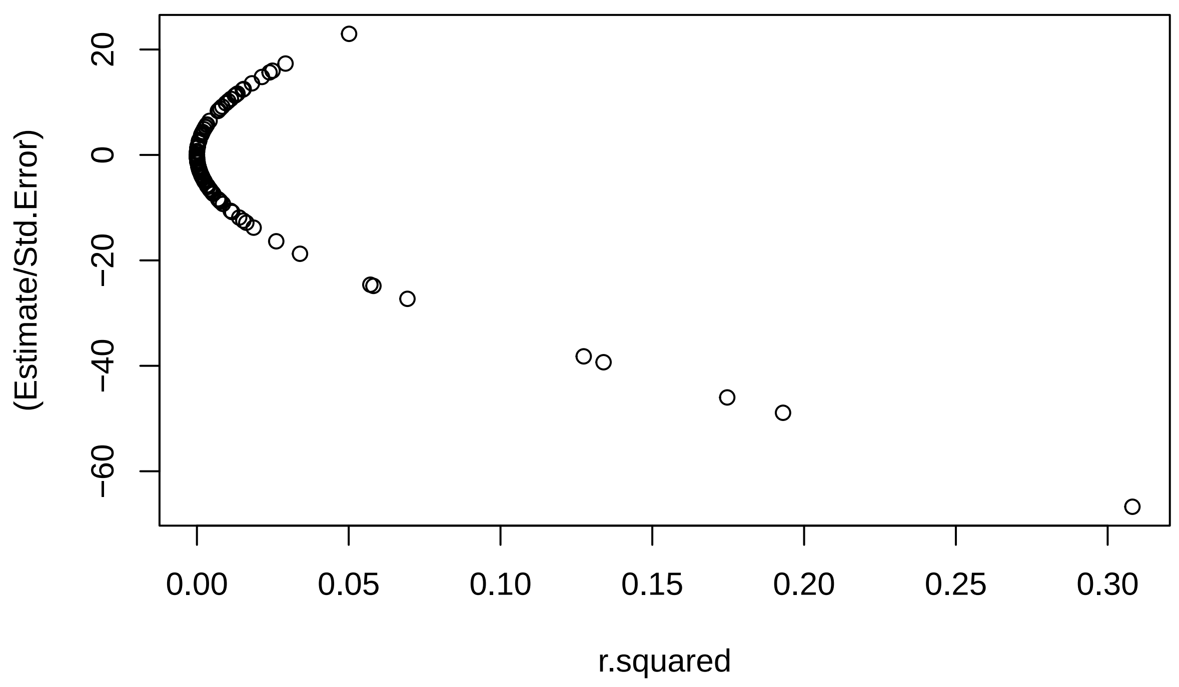


Regression Coefficient and Standard Error of the Estimate

In 1980, we looked for runs which had a regression coefficient less than 0.2, *and* runs whose slope estimate were less than the standard error of the slope estimate. It turns out that this is sort of redundant, the latter condition being the more restrictive.

Below are the standard error plotted against r^2.

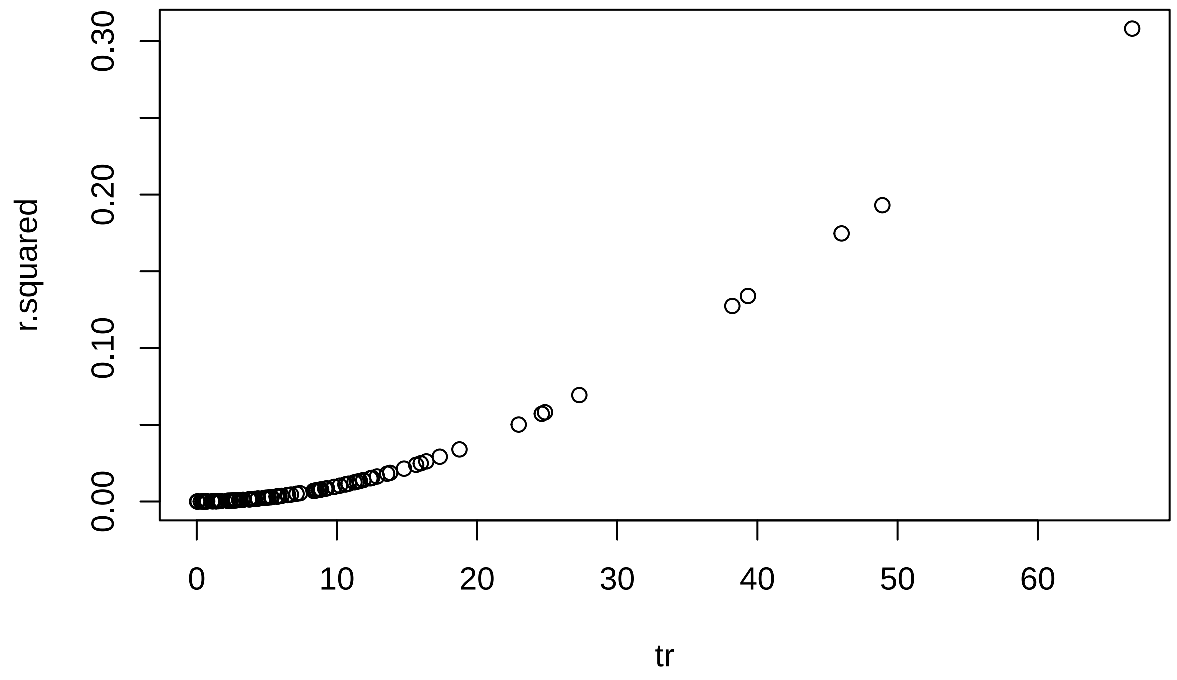
```r
plot( (Estimate/Std.Error) ~ r.squared, data = MD2DF.2)
```



```
  #abline( a = 0, b = 1/2)
```

That is a bit surprising. Let us also add the t statistic for the regression coefficient. The large number of samples (hardley independent) yields large t values.

```
MD2DF.2$tr <- sqrt(MD2DF.2$steps.avg - 1) *
  sqrt( MD2DF.2$r.squared) / (sqrt( 1 - MD2DF.2$r.squared))
plot( r.squared ~ tr, data = MD2DF.2)
```



r near 0.2

Below is a look at the run with a regression coefficient near 0.2. As before, we find the run is number 33 with `which.max`, but hand code the file name.

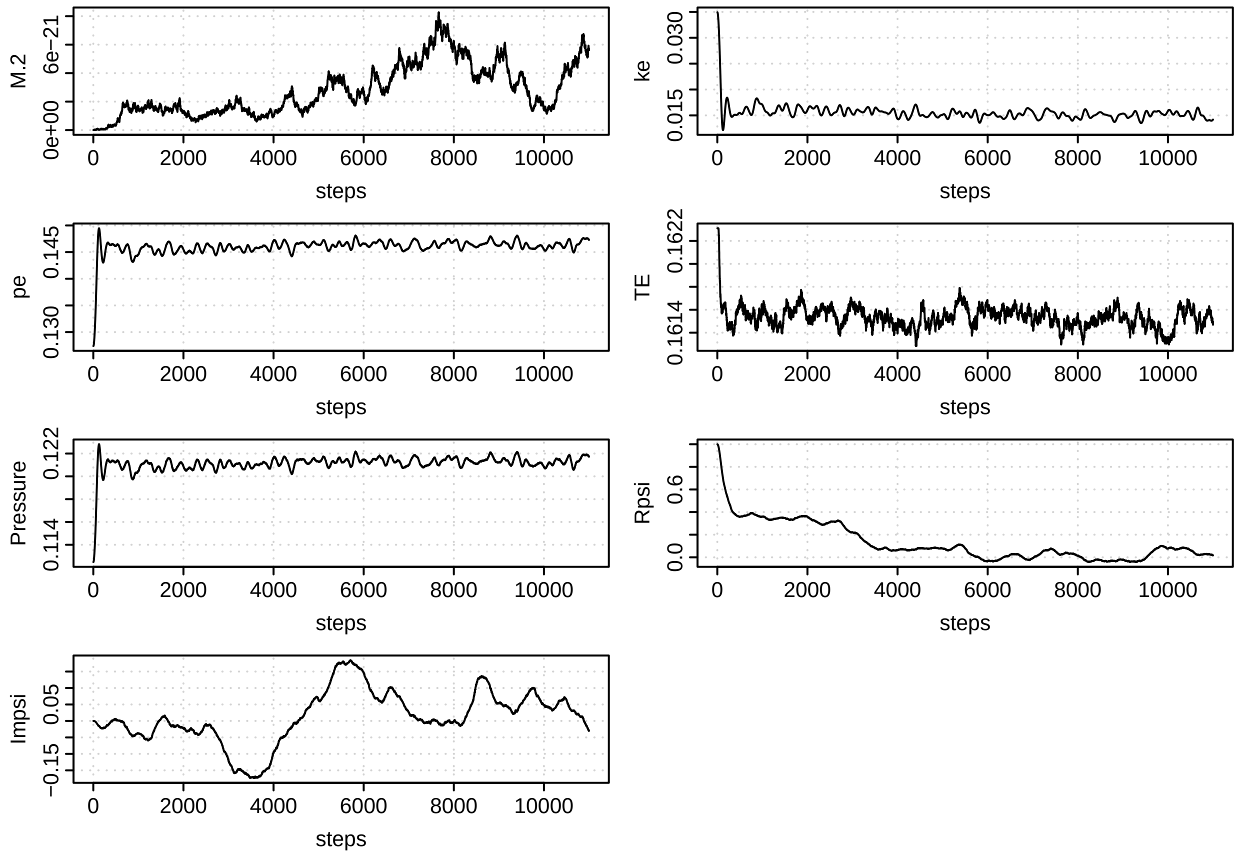
```
# departing from labeling via the run number.
(n02 <- which.max( MD2DF.2[MD2DF.2[,'r.squared']<0.21,'r.squared']))
```

```
## [1] 33
```

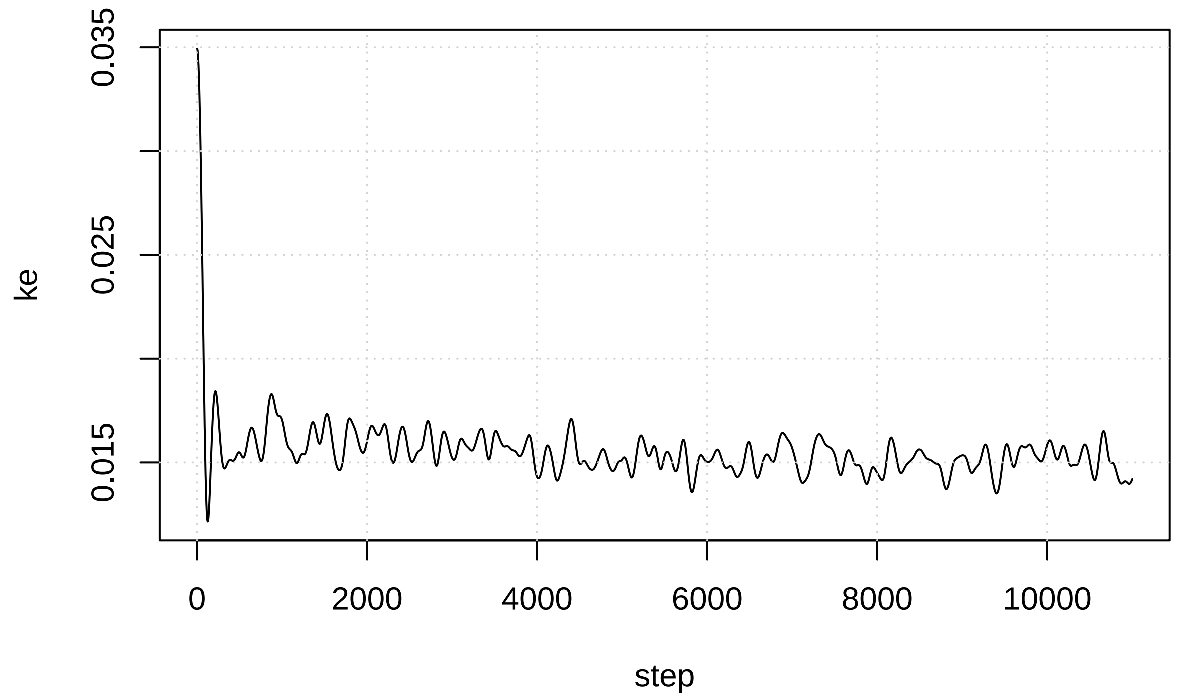
```
MD2DF.2[n02,]
```

```
##    series energy total.steps steps.avg   TE.init xMomentum.mean yMomentum.mean
## 33     S2   0033       11000     10001 0.1623105  -9.013771e-12  -4.616551e-11
##        M.2.mean    ke.mean   pe.mean   TE.mean Pressure.mean  Rpsi.mean
## 33 2.978688e-21 0.01535008 0.1461802 0.1615303     0.1212126 0.09565144
##    Impsi.mean psiabs.mean xMomentum.sd yMomentum.sd       M.2.sd        ke.sd
## 33 0.01586439   0.1434121 2.227176e-11 1.643872e-11 1.715672e-21 0.0007342867
##           pe.sd        TE.sd  Pressure.sd   Rpsi.sd   Impsi.sd psiabs.sd
## 33 0.0007382804 8.205219e-05 0.0003561706 0.1241442 0.08269382 0.1052748
##    xMomentum.min yMomentum.min      M.2.min     ke.min    pe.min    TE.min
## 33 -4.824918e-11 -7.810397e-11 5.436527e-22 0.01350728 0.1441962 0.1612831
##    Pressure.min    Rpsi.min Impsi.min  psiabs.min xMomentum.max yMomentum.max
## 33    0.1201866 -0.04243357 -0.173606 0.005229198  4.525869e-11  2.352341e-12
##         M.2.max    ke.max     pe.max    TE.max Pressure.max  Rpsi.max Impsi.max
## 33 8.27665e-21 0.0173311 0.1480852 0.1617896    0.1221598 0.3664317 0.1843642
##    psiabs.max      Estimate    Std.Error r.squared         rtv     psiavg
## 33  0.3669469 -1.117598e-07 2.285059e-09 0.1930643 0.002288285 0.09695812
##        pe.gain       tr
## 33 0.01880762 48.91381
```

```
xn02 = combineruns(  "H:/Data/DSP/MD/Data/1980", "S2\\.0033\\.")
plotMD2( xn02)
```



```
plot( ke ~ step, data=xn02, type='l')
grid()
```



```r
xn02lm <- lm( ke ~ step, tail( xn02, -1000))
(sxn02lm <- summary( xn02lm))
```

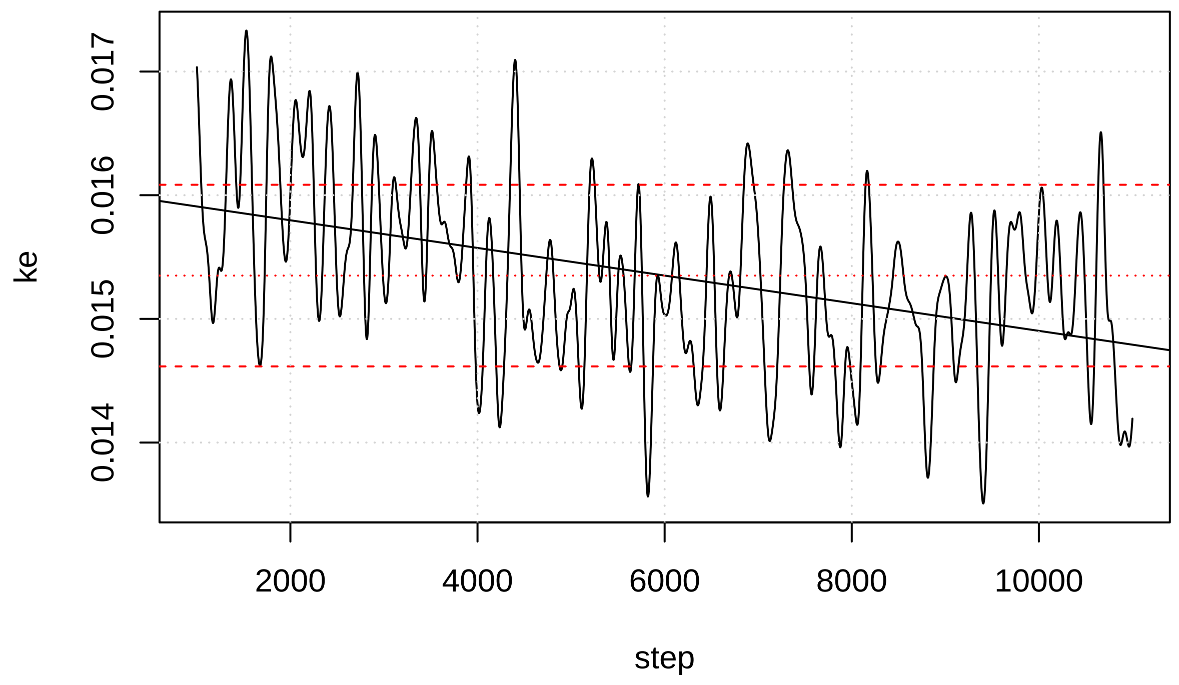
```
##
## Call:
## lm(formula = ke ~ step, data = tail(xn02, -1000))
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -1.807e-03 -4.962e-04 -1.160e-06  5.026e-04  1.680e-03
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.602e-02  1.522e-05 1052.90   <2e-16 ***
## step        -1.118e-07  2.285e-09  -48.91   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0006596 on 9998 degrees of freedom
## Multiple R-squared:  0.1931, Adjusted R-squared:  0.193
## F-statistic:  2392 on 1 and 9998 DF,  p-value: < 2.2e-16
```

```r
(kn02 <- MD2DF.2[ n02, c('ke.mean', 'ke.sd')])
```

```
##       ke.mean        ke.sd
## 33 0.01535008 0.0007342867
```

The estimate is about 49 times the standard error. The plot below shows that a 0.2 regression coefficient is significant. It looks as though the system may have relaxed for about 6,000 steps, but is hard to be sure without running more steps.

```
plot( ke ~ step, data=tail( xn02, -1000), type='l')
abline( coef=coef(xn02lm))
abline( h = kn02$ke.mean - kn02$ke.sd, col='red', lty=2)
abline( h = kn02$ke.mean, col='red', lty=3)
abline( h = kn02$ke.mean + kn02$ke.sd, col='red', lty=2)
grid()
```

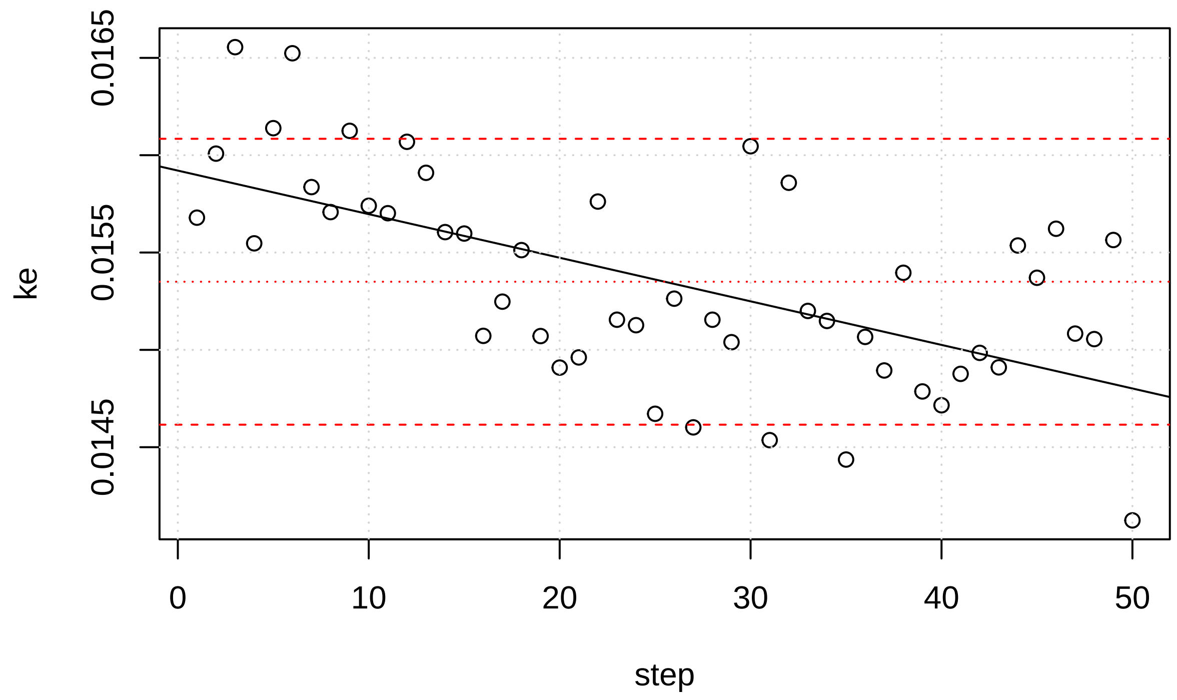


```
xn02rs <- rsmpavg( xn02$ke, 200)
xn02rsdf <- data.frame( step=1:(length(xn02rs) - 5), ke=tail( xn02rs, -5))
xn02rslm <- lm( ke ~ step, xn02rsdf) # skip 1,000 steps.
(xn02rsslm <- summary( xn02rslm))
```

```
##
## Call:
## lm(formula = ke ~ step, data = xn02rsdf)
##
## Residuals:
##        Min         1Q     Median          3Q         Max
## -7.146e-04 -2.603e-04 -8.350e-06  3.143e-04  7.965e-04
```

```
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.592e-02  1.219e-04 130.631  < 2e-16 ***
## step        -2.239e-05  4.160e-06  -5.382 2.17e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0004245 on 48 degrees of freedom
## Multiple R-squared:  0.3764, Adjusted R-squared:  0.3634
## F-statistic: 28.97 on 1 and 48 DF,  p-value: 2.169e-06
```

```
plot( ke ~ step, data=xn02rsdf, type='p')
abline( coef=coef(xn02rslm))
abline( h = kn02$ke.mean - kn02$ke.sd, col='red', lty=2)
abline( h = kn02$ke.mean, col='red', lty=3)
abline( h = kn02$ke.mean + kn02$ke.sd, col='red', lty=2)
grid()
```



The t statistic on that r^2 is below. Note that the regression coefficient of the sampled data is higher than the un-sampled version too.

```
(rsxn02rs <- xn02rsslm$r.squared)
```

```
## [1] 0.3763583
```

```
(txn02rs <- sqrt( 48) * sqrt( rsxn02rs) / sqrt( 1 - rsxn02rs))
```

```
## [1] 5.382125
```

```
pt( txn02rs, df=48)
```

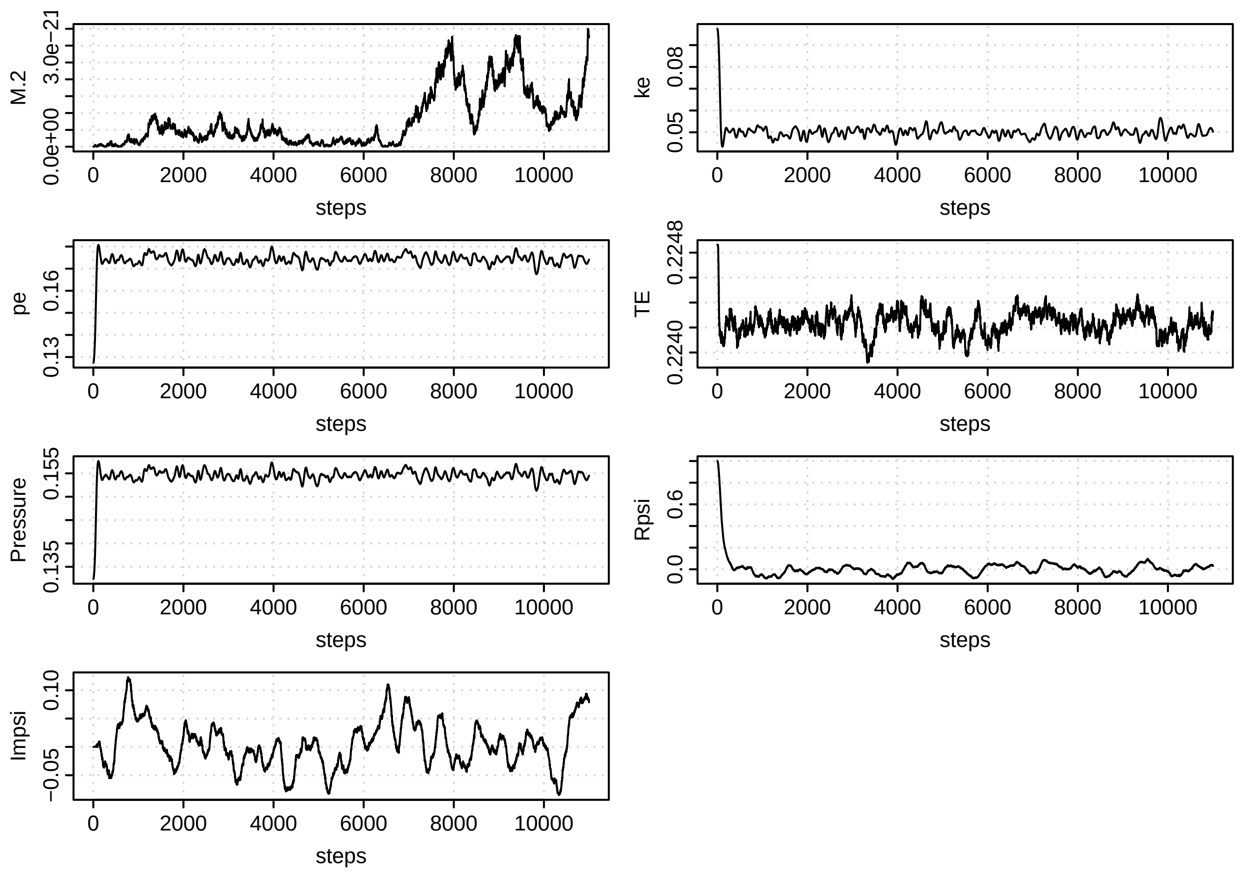
```
## [1] 0.9999989
```

Again, it looks as though the relaxation took about 6,000 iterations, so if we were to skip an additional 20 of the re-sampled data we might convince ourselves that it is near equilibrium. However, runs should extend over many times the relaxation time to ensure that the averages truly reflect equilibrium.

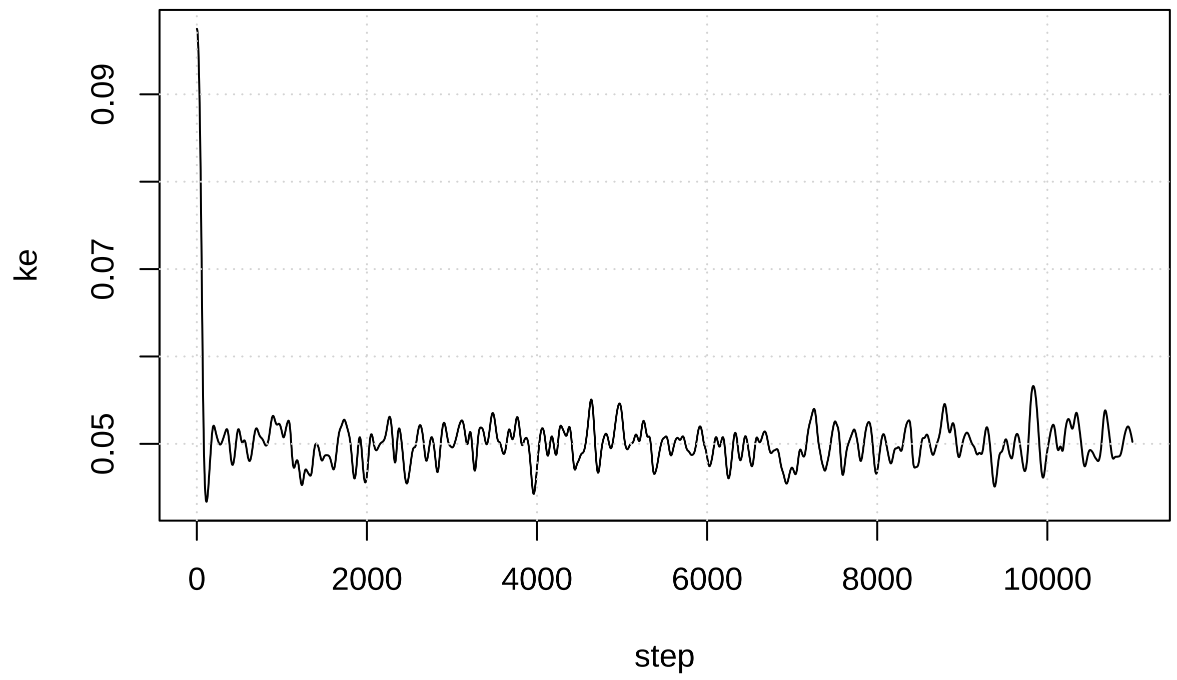
Hot Run

Performing the same analysis on the hottest run.

```
x100 = combineruns(  "H:/Data/DSP/MD/Data/1980", "S2\\.0100\\.")
plotMD2( x100)
```



```
plot( ke ~ step, data=x100, type='l')
grid()
```



```
x100lm <- lm( ke ~ step, tail( x100, -1000))
(sx100lm <- summary( x100lm))
```

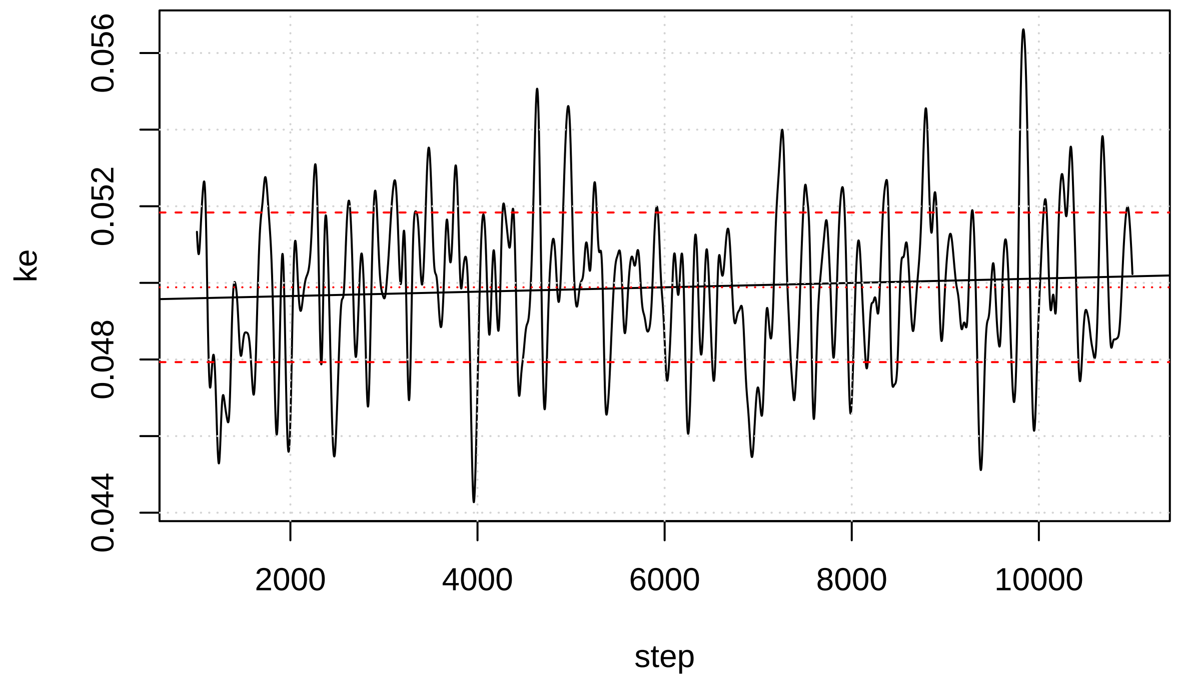
```
##
## Call:
## lm(formula = ke ~ step, data = tail(x100, -1000))
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -0.0054940 -0.0012760  0.0000913  0.0012623  0.0065157
##
## Coefficients:
##              Estimate Std. Error  t value Pr(>|t|)
## (Intercept) 4.954e-02  4.490e-05 1103.416   <2e-16 ***
## step        5.715e-08  6.743e-09    8.475   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.001946 on 9998 degrees of freedom
## Multiple R-squared:  0.007133,   Adjusted R-squared:  0.007034
## F-statistic: 71.83 on 1 and 9998 DF,  p-value: < 2.2e-16
```

```
(k100 <- MD2DF.2[ 100, c('ke.mean', 'ke.sd')])
```

```
##        ke.mean       ke.sd
## 100 0.04988412 0.001953327
```

The estimate is about 8.5 times the standard error. The plot below hints that the slope is significant.

```
plot( ke ~ step, data=tail( x100, -1000), type='l')
abline( coef=coef(x100lm))
abline( h = k100$ke.mean - k100$ke.sd, col='red', lty=2)
abline( h = k100$ke.mean, col='red', lty=3)
abline( h = k100$ke.mean + k100$ke.sd, col='red', lty=2)
grid()
```

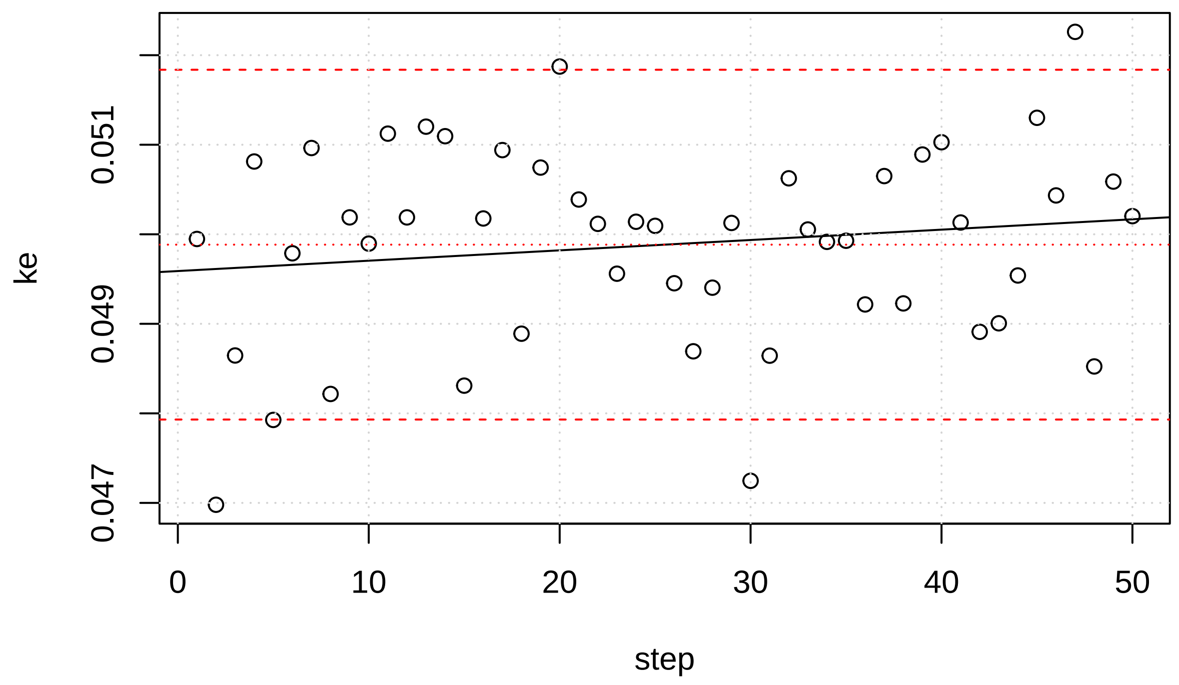


```
x100rs <- rsmpavg( x100$ke, 200)
x100rsdf <- data.frame( step=1:(length(x100rs) - 5), ke=tail( x100rs, -5))
x100rslm <- lm( ke ~ step, x100rsdf) # skip 1,000 steps.
(x100rsslm <- summary( x100rslm))
```

```
##
## Call:
## lm(formula = ke ~ step, data = x100rsdf)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -0.0026896 -0.0007978  0.0001962  0.0006577  0.0021280
##
## Coefficients:
```

```
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 4.959e-02  3.186e-04 155.633   <2e-16 ***
## step        1.156e-05  1.087e-05   1.063    0.293
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.00111 on 48 degrees of freedom
## Multiple R-squared:  0.02299,    Adjusted R-squared:  0.002633
## F-statistic: 1.129 on 1 and 48 DF,  p-value: 0.2932
```

```
plot( ke ~ step, data=x100rsdf, type='p')
abline( coef=coef(x100rslm))
abline( h = k100$ke.mean - k100$ke.sd, col='red', lty=2)
abline( h = k100$ke.mean, col='red', lty=3)
abline( h = k100$ke.mean + k100$ke.sd, col='red', lty=2)
grid()
```



The t statistic on that r^2 is calculated below. Note that the regression coefficient of the sampled data is higher than the un-sampled version too, and is borderline significant.

```
(rsx100rs <- x100rsslm$r.squared)
```

```
## [1] 0.02298773
```

```
(tx100rs <- sqrt( 48) * sqrt( rsx100rs) / sqrt( 1 - rsx100rs))
```

```
## [1] 1.062719
```

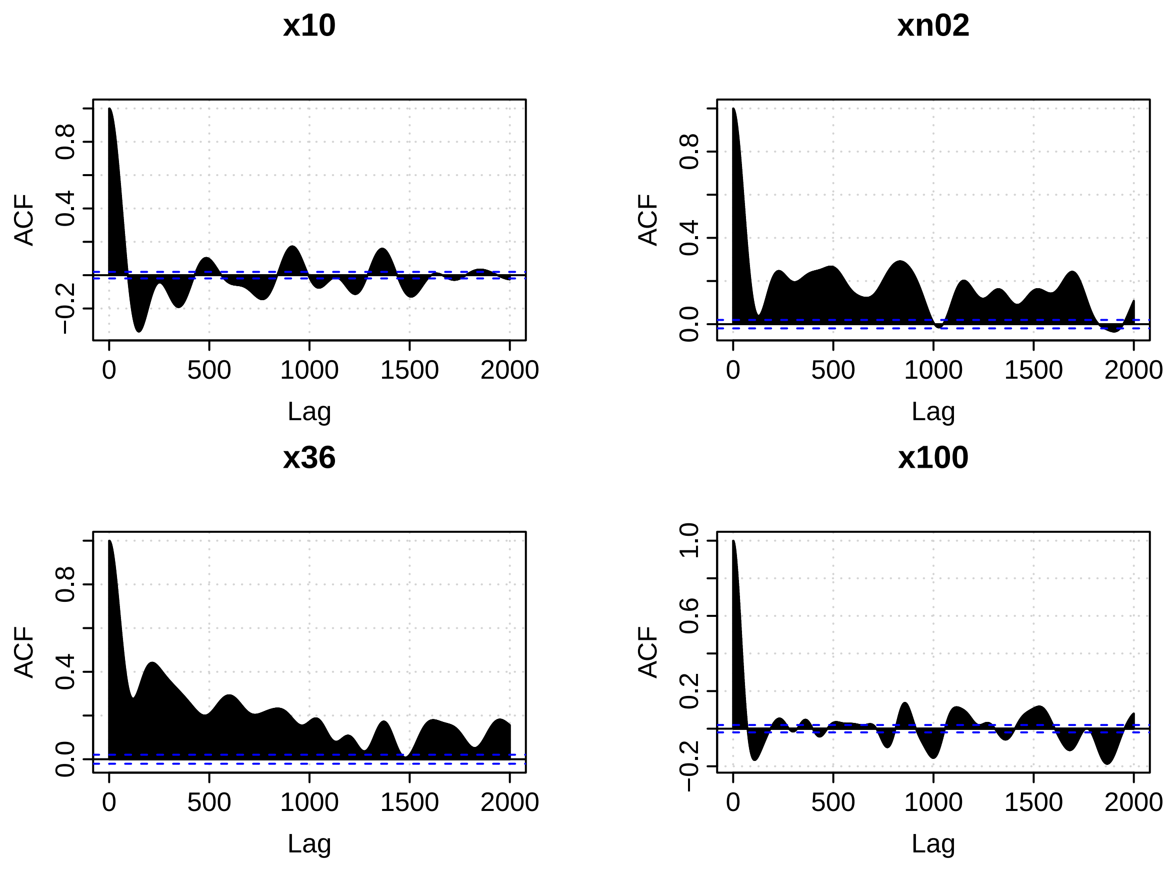
```
pt( tx100rs, df=48)
```

```
## [1] 0.8533866
```

Autocorrelation of the Four Cases

The averages should be long enough to cover the oscillations. The spectrums indicate that the shortest time for the averages (i.e., twice the reciprocal of the main lobe width.) We can get an estimate of hte autocorrelation using the `acf` function.

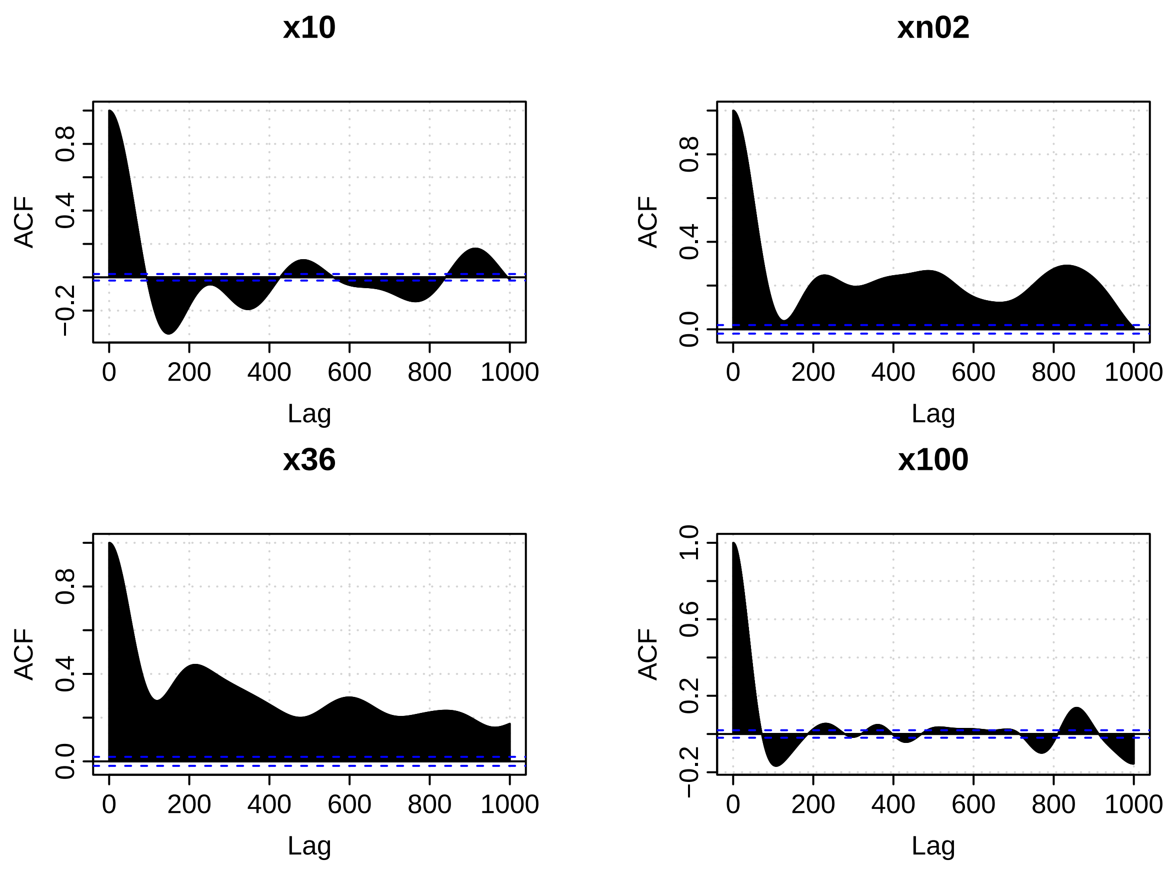
```
lag.max <- 2000
op <- par( mfrow = c(2, 2), mar = c(3, 3, 3, 3), tcl = -0.3,
             mgp = c(1.7, 0.4, 0) )
acf( tail( x10$ke, -1000), lag.max = lag.max, panel.first=grid(), main="x10")

acf( tail( xn02$ke, -1000), lag.max = lag.max, panel.first=grid(), main="xn02")
acf( tail( x36$ke, -2000), lag.max = lag.max, panel.first=grid(), main="x36")
acf( tail( x100$ke, -1000), lag.max = lag.max, panel.first=grid(), main="x100")
```



```
par(op)
```

This looks loike 200 might have been a good choice. We can zoom in a bit.

```
lag.max <- 1000
op <- par( mfrow = c(2, 2), mar = c(3, 3, 3, 3), tcl = -0.3,
              mgp = c(1.7, 0.4, 0) )
acf( tail( x10$ke, -1000), lag.max = lag.max, panel.first=grid(), main="x10")

acf( tail( xn02$ke, -1000), lag.max = lag.max, panel.first=grid(), main="xn02")
acf( tail( x36$ke, -2000), lag.max = lag.max, panel.first=grid(), main="x36")
acf( tail( x100$ke, -1000), lag.max = lag.max, panel.first=grid(), main="x100")
```



```
par(op)
```