Using the New TCD Statistics Cluster

Louis J. M. Aslett

Trinity College, University of Dublin

26^{th} March 2012

Trinity College Dublin

sfi Science Foundation Ireland

This talk will be rubbish!
Expected running time: 49.34 minutes

Context: The above was shown alone and without immediate explanation in the talk as the first slide! Later on, in the demo part of the LaTeX section it was explained that the above was generated using LaTeX+Sweave which allows embedding R code as follows:

```
This talk will be
\Sexpr{sample(c("rubbish","so-so","great"),1)}!

Expected running time:  \Sexpr{round(rnorm(1,45,4),2)}
minutes
```

Introducing the Cluster

Machine	Specification
`bayes`	$2 \times 6 \times 2.4$GHz Xeon E5645 32KB L1, 256KB L2, 12MB L3 128GB RAM 3 x 2TB RAID-5 HD

`bernoulli`
`fisher`
`gauss`
`laplace`
`poisson`
	4×3.4GHz Core i7-2600
	32KB L1, 256KB L2, 8MB L3
	16GB RAM
	nVidia GTX 560Ti
	PXE booting

CUDA Driver Version / Runtime Version	4.2 / 4.1
CUDA Capability Major/Minor version number:	2.0
Total amount of global memory:	1279 MBytes
(11) Multiprocessors x (32) CUDA Cores/MP:	352 CUDA Cores
GPU Clock Speed:	1.46 GHz
Memory Clock rate:	1900.00 Mhz
Memory Bus Width:	320-bit
L2 Cache Size:	655360 bytes
Max Texture Dimension Size (x,y,z)	1D=(65536), 2D=(65536,65535), 3D=(2048,2048,2048)
Max Layered Texture Size (dim) x layers	1D=(16384) x 2048, 2D=(16384,16384) x 2048
Total amount of constant memory:	65536 bytes
Total amount of shared memory per block:	49152 bytes
Total number of registers available per block:	32768
Warp size:	32
Maximum number of threads per block:	1024
Maximum sizes of each dimension of a block:	1024 x 1024 x 64
Maximum sizes of each dimension of a grid:	65535 x 65535 x 65535

What's available

- A full Linux environment for every user
- Completely synchronised environment across machines
- 64-bit R
- Custom compiled high-performance ATLAS BLAS library
- CUDA tools for compiling for GPU (excl. bayes)
- OpenMP
- OpenMPI
- JAGS
- LaTeX (+ Sweave) ... coming soon!

Which machine should be used?

As a general rule if the memory use is under 16GB:

Single threaded: Use one of bernoulli / fisher / gauss / laplace / poisson because when a single core is in use, these can "turbo boost" to 3.8GHz

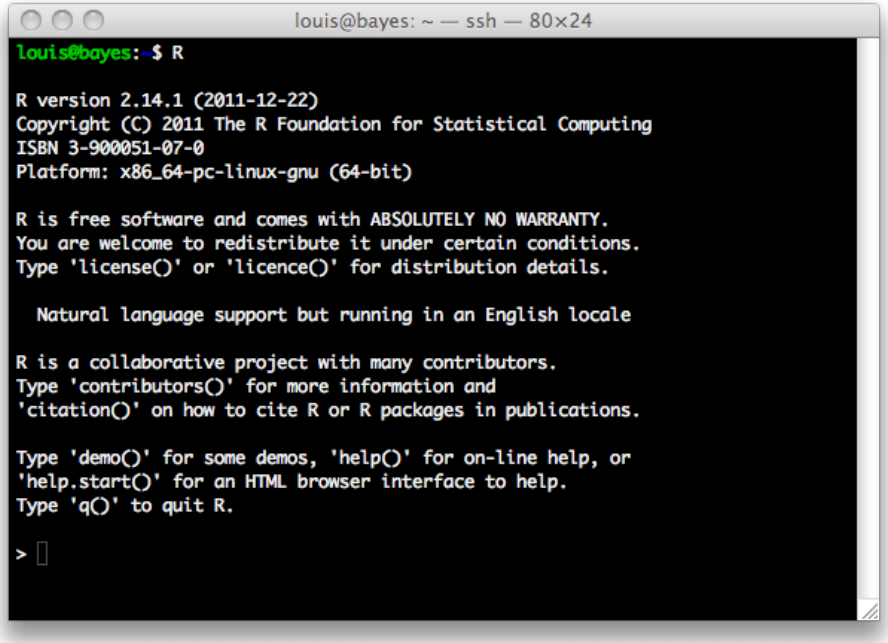
$\leqslant 4$ threads: again, stick with bernoulli / fisher / gauss / laplace / poisson

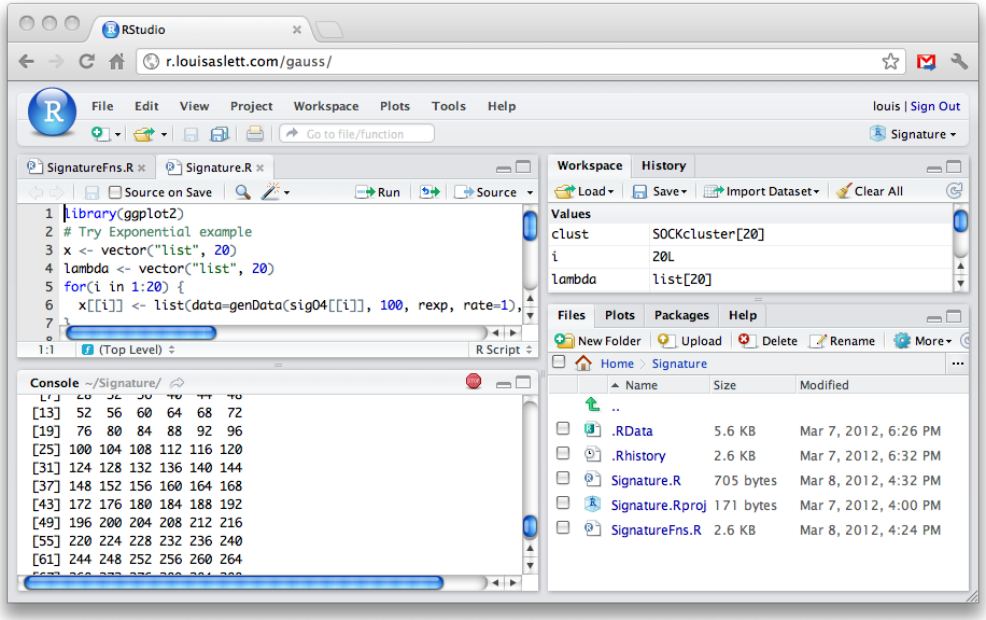
$4 <$ threads $\leqslant 12$: if the threads are not easily split over machines, then choose bayes. Otherwise, spread over bernoulli / fisher / gauss / laplace / poisson.

> 12 threads: will have to split over many machines, otherwise reduce thread use to prevent context swaps.

In all cases, avoid heavy I/O on bernoulli / fisher / gauss / laplace / poisson when possible.

Access to the machines is via SSH or web interface (RStudio).

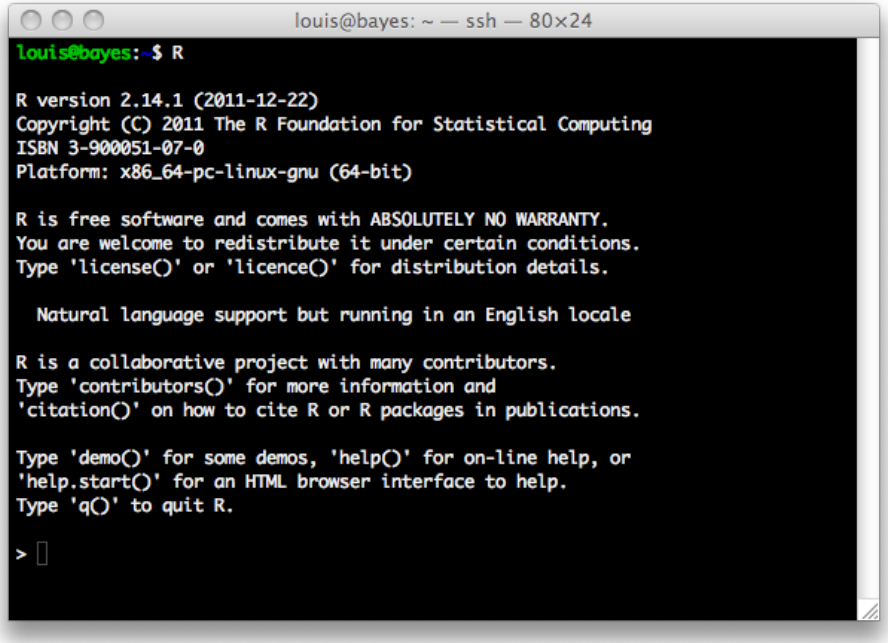


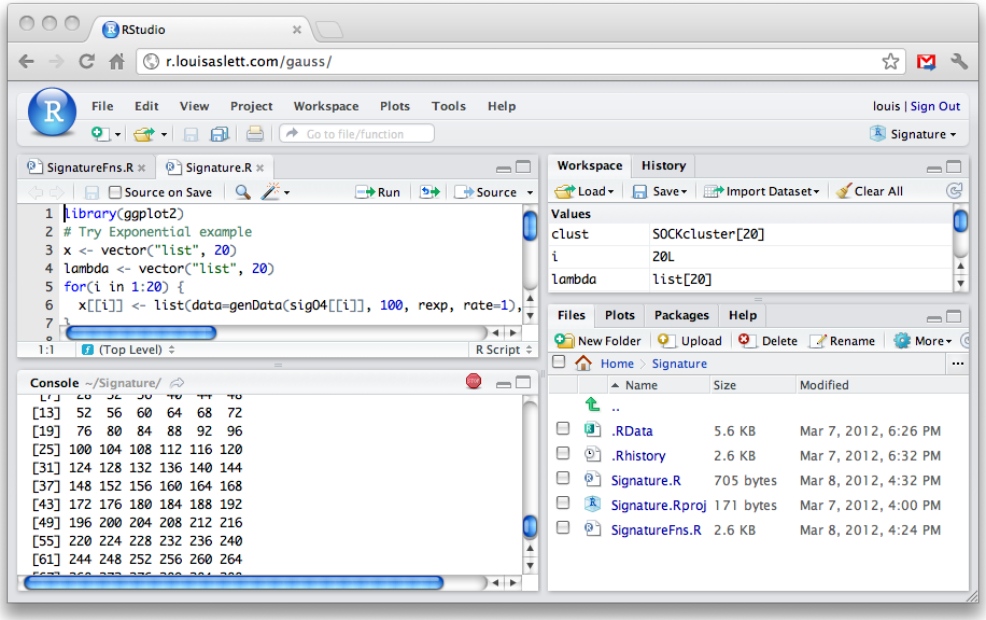


SSH is ideal for complex scenarios involving possibly non-R code. It gives you access to a complete Linux environment.

RStudio allows you to use R from your browser in almost entirely the same way you would on your desktop.

Access to the machines is via SSH or web interface (RStudio).





Inside Trinity

`bayes.scss.tcd.ie`

Outside Trinity

`r.louisaslett.com`

Inside Trinity

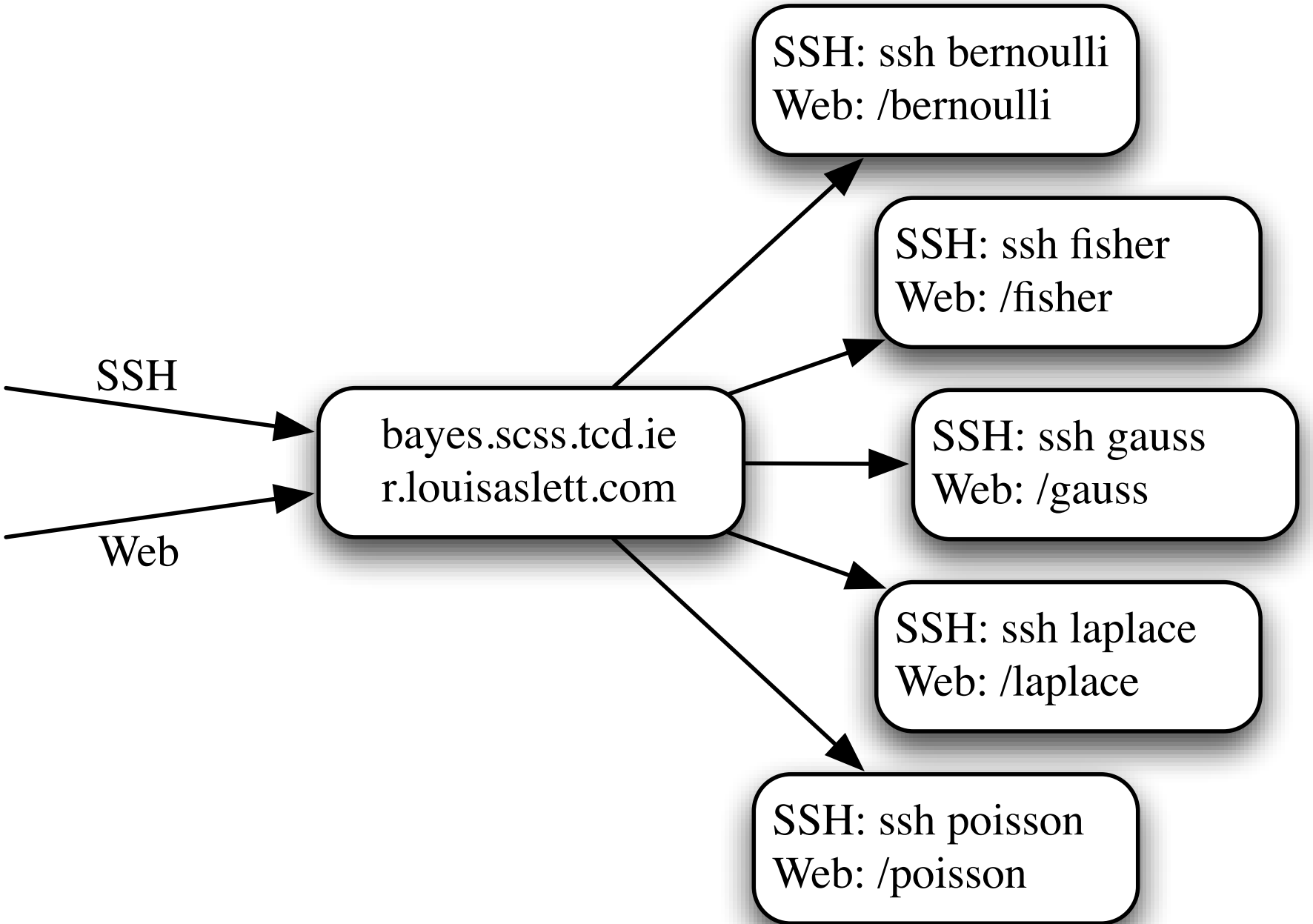
```
bayes.scss.tcd.ie
```

Outside Trinity

```
r.louisaslett.com
```

With both SSH and the web interface, this gets you access to bayes. If using one of the other machines (usually recommended), you must then log on from there to the destination.

Note that for the web, r.louisaslett.com will work both inside and outside Trinity, but for SSH each is mutually exclusive.


SSH: ssh bernoulli
Web: /bernoulli
SSH: ssh fisher
Web: /fisher
SSH
bayes.scss.tcd.ie
r.louisaslett.com
SSH: ssh gauss
Web: /gauss
Web
SSH: ssh laplace
Web: /laplace
SSH: ssh poisson
Web: /poisson

Demo of simple R from SSH and browser.

Demo of installing packages and uploading files.

Bayesian inference Using Gibbs Sampling (BUGS)

WinBUGS is not available as all machines are Linux. However, JAGS (Just Another Gibbs Sampler) which is a very similar alternative is installed at the system level.

Some of the differences make it better! e.g. data format is same as R `dump()`, so matrices etc 'just work'. Key differences:

1. scripting (not important if using R)
2. data format
3. censoring
4. data transformations

JAGS can run from the command line; or interface directly in R using the package rjags.

Consider the well known rat tumour data (Gelman, 2004).

rats.dat — data file in R dump format

```
y <- c(0, 0, 0, 0, 0, 0, ..., 6, 16, 15, 15, 9, 4)
n <- c(20, 20, 20, 20, ..., 20, 52, 46, 47, 24, 14)
N <- 71
```

betabin.jags — BUGS language model file

```
model{
  for(i in 1:N) {
    y[i] ~ dbin(pi[i], n[i])
    pi[i] ~ dbeta(alpha,beta)
  }
  alpha ~ dgamma(0.02, 0.01)
  beta ~ dgamma(0.02, 0.01)
}
```

betabin.init — initial values

```
alpha <- 1
beta <- 1
```

Then, in SSH you can run `jags` at the command line and execute the BUGS model, much like WinBUGS.

jags — command line program

```
model in betabin.jags
data in rats.dat
compile,nchains(1)
initialize
parameters in betabin.init,chain(1)
monitor pi,thin(1)
update 5000
coda pi,stem(pi)
```

Even easier — BUGS directly in R

RStudio

```
source("rats.dat")
jm <- jags.model("betabin.jags")
res <- coda.samples(jm, c("alpha", "beta"), 5000)
```

Demo of BUGS from SSH and browser.

LaTeX from your browser!

Naturally, with LaTeX installed, one can compile documents from command line.

Now: incredibly easy to compile a LaTeX document via RStudio too. Simply create a document with the extension `.Rnw` and RStudio will provide compile options.

Moreover, embed R code directly in your LaTeX document which is evaluated at compile time. Part of drive for 'reproducible research'.

Demo of LaTeX + Sweave.

Demo of the self referential talk!

OpenMP

OpenMP for shared memory parallelism is available for use on all systems. Each system has hand tuned environment variables defined for every user to correctly select number of cores.

Usage is as simple as compiling with `-fopenmp`

```
gcc -fopenmp test.c -o test
```

OpenMP

OpenMP for shared memory parallelism is available for use on all systems. Each system has hand tuned environment variables defined for every user to correctly select number of cores.

Usage is as simple as compiling with `-fopenmp`

```
gcc -fopenmp test.c -o test
```

BUT! ... CAUTION

OpenMP

OpenMP for shared memory parallelism is available for use on all systems. Each system has hand tuned environment variables defined for every user to correctly select number of cores.

Usage is as simple as compiling with `-fopenmp`

```
gcc -fopenmp test.c -o test
```

BUT! ... CAUTION R is *not* thread-safe. Unfortunately, this means you cannot call any R functions from your C code within an OpenMP clause.
Even random number generation using `rand()` requires care: risk of highly correlated RNG stream. Currently, SPRNG seems best bet.

Demo of compiling OpenMP & number of threads.

CUDA

The CUDA SDK for compiling GPU kernels is available on all machines. All machines except bayes have CUDA capable GPUs.
To compile the first lab example from the recent CUDA course:

Option 1 — if need to combine many object files

```
nvcc -g -O2 -c -o first_kernel.o first_kernel.cu
--ptxas-options -v -I/usr/local/cudaSDK/C/common/inc

/usr/bin/gcc-4.4 -o first_kernel first_kernel.o -lcuda
-lcudart -L/usr/local/cuda/lib64
```

Option 2 — straight to the point

```
nvcc -O2 -o first_kernel first_kernel.cu
-I/usr/local/cudaSDK/C/common/inc --ptxas-options -v
```


Demo of compiling CUDA code (?)

CUDA in R

To compile to a shared object which you can call in R requires two changes: an `extern` definition in the source for the functions which should be callable from R and a modified compiler line.

Compiling object for R

```
nvcc -O2 --shared -o first_kernel_R.so
first_kernel_R.cu -I/usr/local/cudaSDK/C/common/inc
--ptxas-options -v -I/usr/share/R/include
-L/usr/lib/R/lib -lR --compiler-options '-fPIC'
```

Demo of compiling CUDA code for R.

Using all 32 CPUs at once

The parallel package in R provides a nice way to run your code on all 32 CPUs in the cluster in just 3 lines of code!

Example

```
library(parallel)

clust <- makePSOCKcluster(rep(c("gauss", "fisher",
"laplace", "poisson", "bernoulli"), each=4))

clusterEvalQ(clust,
source("~/Signature/SignatureFns.R"))

lambda <- clusterApplyLB(clust, x, clustMCMC, ...)

stopCluster(clust)
```

Caveat emptor! All maths libraries are not equal.

Apple Mac's have substantially faster maths libraries than Linux, leading to curious results (e.g this 5-year old laptop being faster than the new servers!)
Computing 200-million exponentials in a loop:

- 5-year old Mac: $\sim$ 4.1 sec
- New server: $\sim$ 6.0 sec, 50% slower!

Caveat emptor! All maths libraries are not equal.

Apple Mac's have substantially faster maths libraries than Linux, leading to curious results (e.g this 5-year old laptop being faster than the new servers!)
Computing 200-million exponentials in a loop:

- 5-year old Mac: $\sim$ 4.1 sec
- New server: $\sim$ 6.0 sec, 50% slower!

Moral: you may benefit from a TCHPC account in order to get use of ICC.

Compiling with ICC on Lonsdale

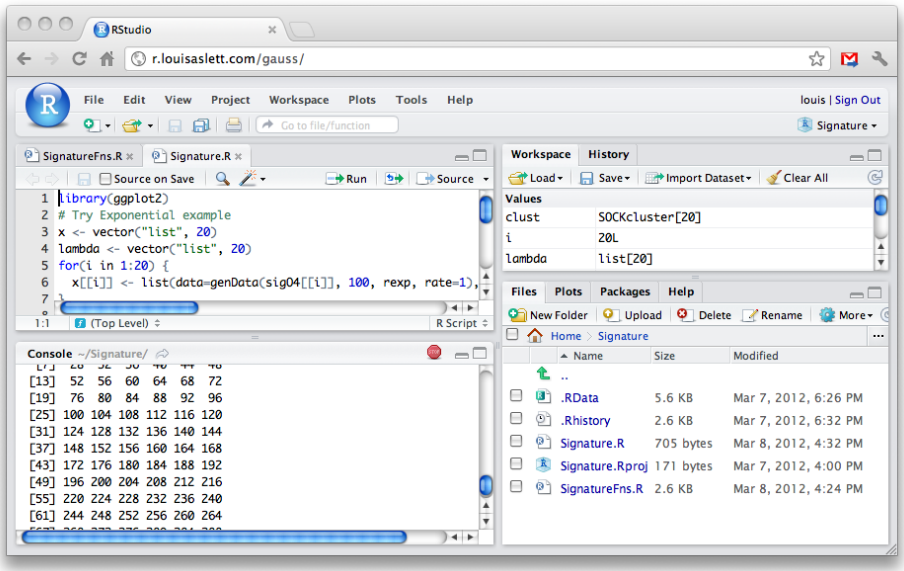
```
module load intel/cc
icc EXP.c -o EXP -xSSE4.2 -static
```

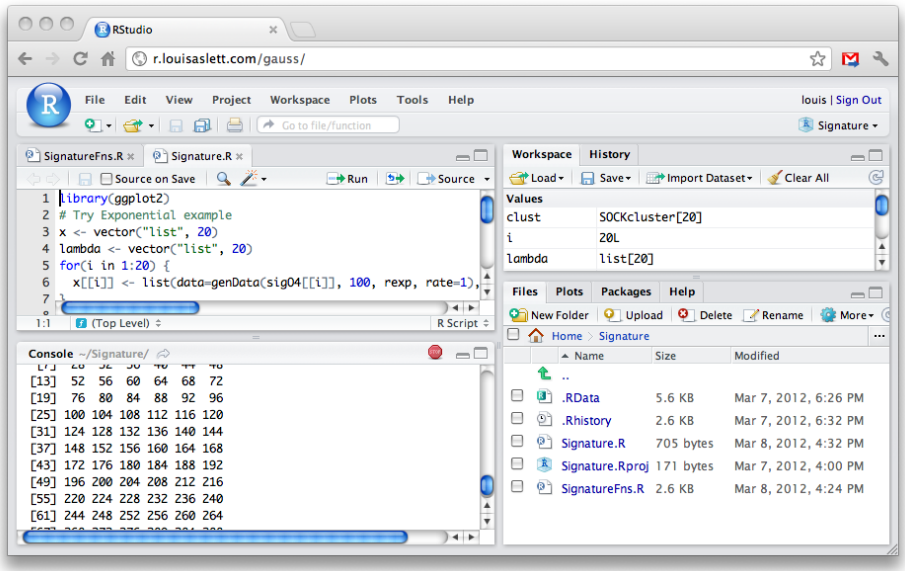
- New server /w ICC: $\sim$ 1.1 sec to 4 sec

Demo (?)

Caveat emptor! RStudio not designed for this setup.

RStudio on two machines simultaneously *might* be a recipe for disaster!





Demo of the problem.