High Performance Computing with R

For many years, R had a reputation for being slow, unable to process large datasets. This was true until c2012 when its compiler caw switched from `parse tree' to `byte code'. Its speed is now similar to Python, Java, Matlab & IDL. All of these are slower than Fortran, C and C++.

```
# Benchmarking R: Ten million element vector on MacBook laptop
w <- rnorm(10000000)            # 0.9 sec
wsort <- sort(w)                # 1.4 sec
wfft <- fft(w)                  # 1.8 sec
  foo <- 0
  loop <- function(n) { for(i in 1:n) {
        if(tan(i) > 0.5) foo = foo + atan(i)^{2/3 * w[i]} } }
loop(10000000)                  # 6.6 sec Loop with nonlinear computation
quartz() ; plot(w)              # 190 sec
write(format(w, digits=2))      # 34 sec
save(w, file='w.out')           # 0.1 sec
```

R programming tools

library(help='utils') & 'base' & 'tools'

- ✓ Program flow (if, for, else, while, repeat, break, next, stop)
- ✓ Host computer (system, list.files, source, readline, Rscript, pipe)
- ✓ Editors, IDEs and GUIs (Rstudio, Jupyter, edit, emacs, vi)
- ✓ Debugging (debug, browser, try, traceback, Rprof, testthat)
- ✓ LaTeX (xtable, Sweave, knitr)
- ✓ Language interfaces (C, ***C++***, Fortran, ***Python***, Java, Julia, ***Matlab,*** SQL, HTML, Oracle, Tcl/Tk, BUGS, JAGS, Stan)

Use ***rpy2*** to access R from Python:

```
conda install rpy2  ##  on terminal, installs R and rpy2
pip install rpy2    ##  in (i)Python, requires R previously installed
import rpy2
import rpy2.robjects as robjects
R = robjects.r
ranGauss = R.rnorm(100)
print(ranGauss)
```

Strategies for speeding up R code

- Precompile user-created functions
- User vector/matrix operations where possible
- Avoid *for(i in 1:N)* loops where possible
- Use external C, Fortran or C++ routines for computationally intensive steps
- Profile your code: *Rprof*, CRAN *rbenchmark, microbenchmark*

CRAN packages for parallel processing

R is not intrinsically designed for parallel processing but, due to utilities for interaction with the host computer, dozens of CRAN packages are now available to facilitate parallel & distributed computing

- *parallel* and *foreach* functions distributes *for* loop to resident cores
- *multicore, batch & condor* serve multicore computers
- *mclapply* applies any function to each element of a vector in parallel
- *h2o* facilitates machine learning (e.g. RFs, ANNs) in a parallel environment
- *CRAN HadoopStream & hive* serve MapReduce in Hadoop environment
- *CRAN cloudRmpi* serves MPI and
- *Gputools, magma* & *OpenCl* serve GPU clusters
- *RevoScaleR* integrates R with Microsoft SQL Server
- *datatable, ff* & *bigmemory* treat large out-of-memory datasets

While originally designed for an individual exploring small datasets,
R can be pipelined and can treat megadatsets

also

R scripts are very compact
Two Penn State Ph.D. theses completed in 10^2 lines of code