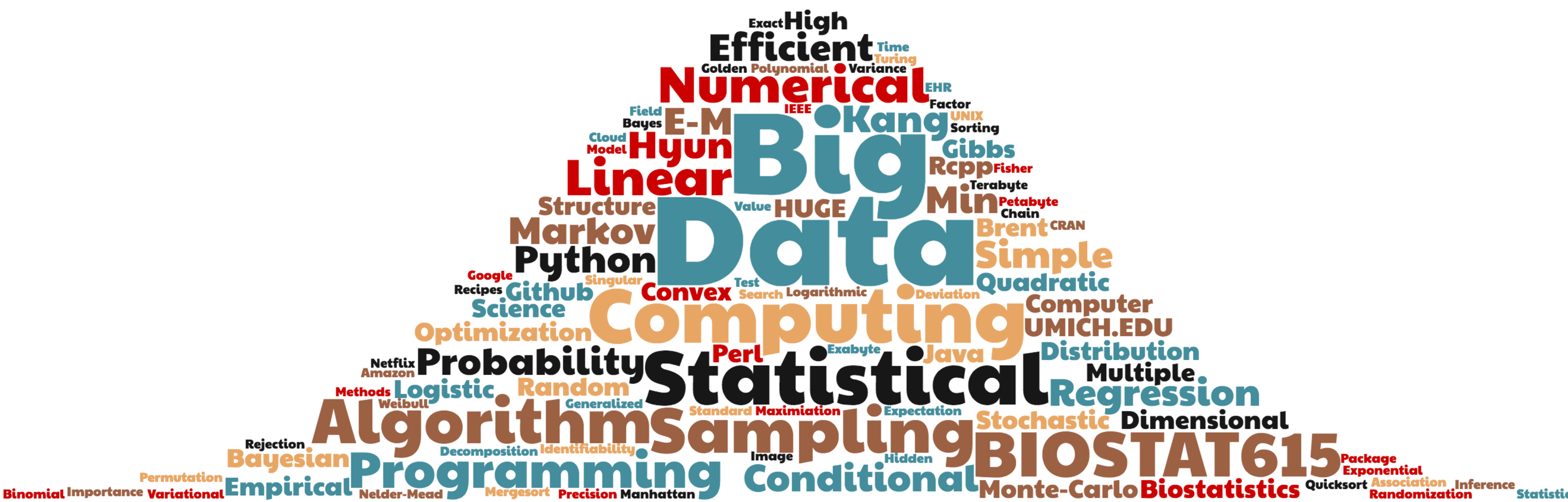
MODULE 1 / UNIT 1

INTERFACING R AND C++



Recap : Computing π numerically

- **Fact:** $$\sum_{i=1}^{\infty} \frac{1}{i^2} = \frac{\pi^2}{6}$$

$$\pi = \sqrt{6 \sum_{i=1}^{\infty} \frac{1}{i^2}}$$

- **How many digits can we calculate accurately?**

A double-precision implementation

```
#include <iostream>
#include <cmath>
using namespace std;
int main(int argc, char** argv) {
  int n = strtof(argv[1],NULL);
  double sum = 0;
  for(int i=n; i >0; --i)
    sum += (1.0/i/i);
  cout.precision(10);
  cout << sqrt(sum * 6) << endl;
  return 0;
}
```

An equivalent implementation in R

```r
args = commandArgs(trailingOnly = TRUE)
n = as.integer(args[1])
sum = 0
for(i in n:1) {
  sum = sum + (1.0/i/i);
}
options(digits=10)
cat(sqrt(6*sum))
cat("\n")
```

An equivalent implementation in python

```python
import sys
import math
sum = 0
for i in range(int(sys.argv[1]),0,-1):
    sum += (1.0/i/i);
print(math.sqrt(sum*6.0))
```

```
kang2015:1-0-basic-prog hmkang$ time python pi.py 100000000
3.14159264404967

real	0m22.782s
user	0m21.976s
sys	0m0.213s
kang2015:1-0-basic-prog hmkang$ time Rscript pi.r 100000000
3.141592644

real	0m15.724s
user	0m14.891s
sys	0m0.217s
kang2015:1-0-basic-prog hmkang$ time ./pi 100000000
3.141592644

real	0m0.329s
user	0m0.316s
sys	0m0.004s
kang2015:1-0-basic-prog hmkang$ time ./pi 1000000000
3.141592653

real	0m3.300s
user	0m3.155s
sys	0m0.035s
```

Comparing running times ($n = 10^8$, $n = 10^9$)

- Which one is fastest?
- By how much?

Making R implementation faster

```r
args = commandArgs(trailingOnly = TRUE)
n = as.integer(args[1])
unit = as.integer(args[2])
s = 0
for(i in seq(n,1,-unit)) {
  s = s + sum(1/(i:(i-unit+1))^2)
}
options(digits=10)
cat(sqrt(6*s))
cat("\n")
```

Some running examples

```
kang2015:1-0-basic-prog hmkang$ time Rscript pi.r 100000000 1000000
3.141592644

real	0m2.432s
user	0m2.194s
sys	0m0.207s
kang2015:1-0-basic-prog hmkang$ time Rscript pi.r 100000000 100000000
3.141592644

real	0m2.868s
user	0m2.253s
sys	0m0.547s
kang2015:1-0-basic-prog hmkang$ time Rscript pi.r 1000000000 1000000
3.141592653

real	0m22.648s
user	0m20.497s
sys	0m1.969s
```

Why is this way faster?

- Making an array, taking squares of each element, and summing them up shouldn't take shorter than a simple loop.

- The secret lies in the implementation of the functions. For example, the implementation of `sum()` function is

```
> sum
function (..., na.rm = FALSE)  .Primitive("sum")
```

`.Primitive(), .Internal(), .Call()` functions mean that these are compiled code by other languages (e.g. C, Fortran).

Using C++ directly inside R

```
args = commandArgs(trailingOnly = TRUE)
n = as.integer(args[1])
library(Rcpp)
cppFunction('double zeta2(int n) {
  double sum = 0;
  for(int i=n; i > 0; --i)
    sum += 1.0/i/i;
  return sum;
}')
options(digits=10)
cat(sqrt(6*zeta2(n)))
cat("\n")
```

`cppFunction()` in **`Rcpp`** package allows us to define a function written in C++

Running examples

```
kang2015:1-0-basic-prog hmkang$ time Rscript pic.r 100000000
3.141592644

real	0m2.976s
user	0m2.743s
sys	0m0.194s
kang2015:1-0-basic-prog hmkang$ time Rscript pic.r 1000000000
3.141592653

real	0m5.730s
user	0m5.503s
sys	0m0.190s
kang2015:1-0-basic-prog hmkang$
```

~2.6 extra seconds compared to pure C++ implementation is due to compile time

Using Jupyter notebook for interactive learning

- Open Terminal (of MobaXTerm)

- Change your current directory when the lecture material is downloaded, e.g.

```
$ cd ~/Downloads/615_1_1/
$ cd
/mnt/c/Users/[YourWindowsUserName]/Downloads/615_1/
```

- Run

```
$ jupyter notebook
```

- And open `rcpp_pi.ipynb`

localhost:8888/notebooks/rcpp_pi.ipynb

jupyter rcpp_pi (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted

R

Markdown

First example of using Jupyter R notebook

We will now use Jupyter notebook to facilitate interactive learning. The example of numerically computing $\pi = 3.14159265359\ldots$ value using $\zeta(\cdot)$ function is described below.

First, to use C++ function, `Rcpp` package must be loaded.

In [1]:

```
library(Rcpp)
```

, and use `cppFunction` to define a C++ function

In [2]:

```
cppFunction('double zeta2(int n) {
  double sum = 0;
  for(int i=n; i > 0; --i)
    sum += 1.0/i/i;
  return sum;
}')
```

Jupyter R notebook: rcpp_pi.ipynb

Jupyter R notebook:
rcpp_pi2.ipynb

Calculating a quadratic sum.

- **Consider calculating**

$$f(\mathbf{x}, A, \mathbf{y}) = \mathbf{x}^T A \mathbf{y}$$

$$= \sum_{i=1}^{n} \sum_{j=1}^{m} x_i A_{ij} y_j$$

  where $A \in \mathbb{R}^{m \times n}$

- **Which one would be fastest and slowest?**
  - 2-dimensional loop, R implementation
  - 2-dimensional loop, C++ implementation.
  - matrix multiplication, R implementation.

Jupyter R notebook: quadsum.ipynb

Summary

- Using C++ within R using Rcpp package.

- Implementation can become much faster than pure R implementation, especially when loop is involved.

- Many built-in functions are already efficiently implemented with C & Fortran, so Rcpp implementation is needed only for certain insufficient parts.

- Matrix operation in R is implemented much faster than C++ implementation of simpler loop.