Lección:
Lenguaje de Programación JULIA

ICI3140 Métodos Numéricos

Profesor : Dr. Héctor Allende-Cid
e-mail : hector.allende@ucv.cl

JULIA

- *Julia is a highlevel, **high-performance** dynamic programming language for technical computing, **with syntax that is familiar** to users of other technical computing environments. It provides a **sophisticated compiler**, distributed parallel execution, numerical accuracy, and an **extensive mathematical function library**.*
- *The core of the Julia implementation is licensed under the **MIT license**.Various libraries used by the Julia environment include their own licenses such as the GPL, LGPL, and BSD (therefore the environment, which consists of the language, user interfaces, and libraries, is under the GPL).*
- *— http://julialang.org/*

¿Porqué Julia?

- Multiplataforma
- Desempeño
- Comunidad activa
- Capacidad (Distribución y Paralelismo)

	Fortran gcc 4.8.2	Julia 0.3.7	Python 2.7.9	R 3.1.3	Matlab R2014a	Octave 3.8.1	Mathematica 10.0	JavaScript V8 3.14.5.9	Go go1.2.1	LuaJIT gsl-shell 2.3.1	Java 1.7.0_75
fib	0.57	2.14	95.45	528.85	4258.12	9211.59	166.64	3.68	2.20	2.02	0.96
parse_int	4.67	1.57	20.48	54.30	1525.88	7568.38	17.70	2.29	3.78	6.09	5.43
quicksort	1.10	1.21	46.70	248.28	55.87	1532.54	48.47	2.91	1.09	2.00	1.65
mandel	0.87	0.87	18.83	58.97	60.09	393.91	6.12	1.86	1.17	0.71	0.68
pi_sum	0.83	1.00	21.07	14.45	1.28	260.28	1.27	2.15	1.23	1.00	1.00
rand_mat_stat	0.99	1.74	22.29	16.88	9.82	30.44	6.20	2.81	8.23	3.71	4.01
rand_mat_mul	4.05	1.09	1.08	1.63	1.12	1.06	1.13	14.58	8.45	1.23	2.35

Figure: benchmark times relative to C (smaller is better, C performance = 1.0).

Instalacion

- Descargar Julia v0.3.1 from http://julialang.org/downloads/
- Evitar versiones v0.2.1 and v0.1.2
- Seguir instrucciones para instalar

- Para descargar en Ubuntu con apt-get es necesario agregar un repositorio.
  - sudo add-apt-repository ppa:staticfloat/juliareleases
  - sudo add-apt-repository ppa:staticfloat/julia-deps
  - sudo apt-get update
  - sudo apt-get install julia

Tipos de Datos

Integer types

Type	Signed?	Number of bits	Smallest value	Largest value
`Int8`	✓	8	-2^7	2^7 - 1
`Uint8`		8	0	2^8 - 1
`Int16`	✓	16	-2^15	2^15 - 1
`Uint16`		16	0	2^16 - 1
`Int32`	✓	32	-2^31	2^31 - 1
`Uint32`		32	0	2^32 - 1
`Int64`	✓	64	-2^63	2^63 - 1
`Uint64`		64	0	2^64 - 1
`Int128`	✓	128	-2^127	2^127 - 1
`Uint128`		128	0	2^128 - 1
`Bool`	N/A	8	`false` (0)	`true` (1)
`Char`	N/A	32	`'\0'`	`'\Uffffffff'`

Tipos de Datos

Floating-point types

Type	Precision	Number of bits
`Float16`	half	16
`Float32`	single	32
`Float64`	double	64

```
10000           #> 10000
typeof(10000)   #> Int64
0x12            #> 0x12
typeof(0x12)    #> Uint8
0x123           #> 0x0123
typeof(0x123)   #> Uint16
1.2             #> 1.2
typeof(1.2)     #> Float64
1.2e-10         #> 1.2e-10
```

- Números complejos y racionales

```
1 + 2im  # 1 + 2i
6//9     # 2/3
```

Tipos de Datos

- `BigInt` - arbitrary precision integer
- `BigFloat` - arbitrary precision floating point numbers

```
big_prime = BigInt("5052785737795758503064406447721934417290878968063369478337")
typeof(big_prime)   #> BigInt

precise_pi = BigFloat("3.14159265358979323846264338327950288419716939937510582097")
typeof(precise_pi)  #> BigFloat
```

Si son necesarios tipos "a medida", se tienen constructores

```
type Point
    x::Float64
    y::Float64
end

# Point is the constructor.
p1 = Point(1.2, 3.4)
p2 = Point(0.2, -3.1)
```

Operaciones y funciones básicas

- +, -, *, / operadores

```
a = 4.0
b = 7.0
c = (b - a) * (b + a) / a
```

- Elevado a "^"

```
a = 2 ^ 10
```

- Todas las funciones mat. comunes, como exp, sin, . . .

```
result = exp(-2.33) * cos(22 * 180 / pi)
```

Expresiones booleanas

- Evalua true o false
- Usa operadores ==, !=, <, >, <=, >=

```
value = 4
value == 4
value == "4"
value > 9.0
value <= 5.3
```

- Negación de la expresión booleana !

```
!(value == "4")
```

- Combinar expresiones booleanas usando && and ||

```
(value == 4) && (value == "4")
(value == 4) || (value == "4")
```

If/else

- Testea si una expresion booleana es T o F y ejecuta el código correspondiente:

```
if (value < 5)
        value = 10
else
        value = 20
end
```

- Mas de 2 casos

```
if (value < 5)
        value = 10
elseif (value == 5)
        value = 15
else
        value = 20
end
```

Sintáxis simple y familiar

- n:m crea un rango incluyendo ambos limites.
  - range(n, m) de Python incluye el límite izquierdo pero no el derecho lo que induce a confusión.
- [... for x in xs] crea un arreglo de xs, el cual es iterable.
  - **list comprehension** en Python y Haskell.
- 4:8 *#> 4:8*
- [x **for** x **in** 4:8] *#> [4,5,6,7,8]*
- [4:8] *#> [4,5,6,7,8]*
- [x * 2 **for** x **in** 4:8] *#> [8,10,12,14,16]*

Sintáxis simple y familiar

- Los indices de los arreglos comienzan con 1 y no 0.
  - Todos los indices de 1:n son accesibles.
- Se puede usar datos de rango para copiar parte de un arreglo.
  - Los pasos de un rango se coloca entre start y stop. (i.e. start:step:stop)
  - Se pueden especificar rangos negativos.
  - Indice especial "end" que indica el ultimo indice del arreglo.
- xs = [8, 6, 4, 2, 0]
- xs[1:3] *#> [8,6,4]*
- xs[4:**end**] *#> [2,0]*
- xs[1:2:**end**] *#> [8,4,0]*
- xs[**end**:-2:1] *#> [0,4,8]*

Listas

- Se puede crear una lista enumerada de distintos tipos, e.g.,

```
my_list = {"a", 1, -0.76}
```

- Se accede al elemento i usando [i]

```
my_list[2] + my_list[3]
```

- A diferencia de otros lenguajes el primer indices es 1

```
my_list[1] # primer elemento
my_list[0] # error
```

- Indice especial para el último indice

```
my_list[end] # último elemento de la lista
my_list[end - 1] # antepenúltimo
```

- Largo de la lista

```
length(my_list)
```

Ciclos for

- Ejecuta un bloque de código varias veces
- Uso común iterar sobre un rango

```
value = 0
for i in 1:10
        value += i # abreviación para value = value + i
end
```

- O iterar sobre una lista

```
value = 0
my_list = {1, 2, 3, 4, 5}
for i in my_list
        value += i
end
```

Funciones

- Trozo de código que se puede repetir muchas veces, e.j.,

```
println("Hello, World!")
println(“Hola como estas?")
println(49876)
```

- Las funciones toman argumentos, e.g., println imprime su argumento
- Las funciones pueden retornar un valor el cual se puede almacenar en una variable

```
length_of_list = length(my_list)
```

Algunas funciones importantes

- Salir de Julia Julia: quit()
- Imprimir una ayuda de una función: help(sin)
- Número aleatorio entre 0 y 1: rand()

Funciones Propias

- Estructura parecida a Python y Matlab. Permite parametros por defecto. (Matlab = nargin)

```
function mifuncion(a,b=1,c=2)
        #Codigo

        return variable/s

end
```

Suprimir la salida

- El correr un comando en el terminal de Julia automaticamente imprimira su salida

- Para suprimir la impresion se coloca al final de la linea de código ";"

```
value = 3
value = 3;
```

Just-In-Time Compiler

- Para correr un programa en JULIA, no es necesario compilarlo antes.

```
% cat myprogram.jl
n = 10
xs = [1:n]
println("the total between 1 and $n is $(sum(xs))")
% julia myprogram.jl
the total between 1 and 10 is 55
```

- Desde la version 0.3, las librerias estandar estan precompiladas, cuando se hacen un build de Julia.

```
% time julia myprogram.jl
the total between 1 and 10 is 55
0.80 real 0.43 user 0.10 sys
```

Corriendo scripts

- Se pueden correr archivos directamente del terminal
- Con pwd() indica el directorio actual
- Con cd se puede cambiar de directorio

```
cd("Documentos/ICI3140")
```

- Correr un script o archivo usando el comando “include”

```
include("testfile.jl")
```

Librerias

- Varias librerias adicionales en la librería estandar de Julia

- Numerical computing
  - OpenBLAS — basic linear algebra subprograms
  - LAPACK — linear algebra routines for solving systems
  - Intel® Math Kernel Library (optional) — fast math library for Intel processors
  - SuiteSparse — linear algebra routines for sparse matrices
  - ARPACK — subroutines desined to solve large scale eigenvalue problems
  - FFTW — library for computing the discrete Fourier transformations
- Other tools
  - PCRE — Perl-compatible regular expressions library
  - libuv — asynchronous IO library

Librerias

- Algunas funciones de la libreria de algebra lineal

```
a = randn((50, 1000))    #   50x1000 matrix
b = randn((50, 1000))    #   50x1000 matrix
x = randn((1000, 1000))  # 1000x1000 matrix

# dot product
dot(vec(a), vec(b))
# matrix multiplication
a * x
# LU factorization
lu(x)
# eigen values and eigen vectors
eig(x)
```

Librerias

- Hay librerias que la comunidad contribuye que no forman parte de la instalación básica, e.j., graficar
- Instala una libreria oficial de Julia con Pkg.add(), e.j., Gadfly,

```
Pkg.add("Gadfly")
```

- Actualizar librerias con Pkg.update

```
Pkg.update()
```

- Para usar el código de una libreria simplemente agregarla al comienzo del script

```
using Gadfly
```

- Prueba el ejemplo!

```
x_values = 0:0.1:10
plot(x=x_values, y=sin(x_values), Geom.point)
```

Type System

- El type system de Julia entra en la categoria de dynamic type-checking, en donde se verifica en tiempo real.

```
x = 12
typeof(x)  #> Int64
y = 12.0
typeof(y)  #> Float64

# this function only accepts an Int64 argument
function foo(x::Int64)
    println("the value is $x")
end

foo(x)  #: the value is 12
foo(y)  #! ERROR: no method foo(Float64)
```

Type System

- Los tipos pueden ser parametrizados por otros tipos o valores. (type parameters)
- Ej., un arreglo tiene 2 type parameters – tipo elemento y las dimensiones
  - The `Array{T,D}` type contains elements typed as `T`, and is a `D` dimensional array.

```
typeof([1, 2, 3])                 #> Array{Int64,1}
typeof([1.0, 2.0, 3.0])           #> Array{Float64,1}
typeof(["one", "two", "three"])   #> Array{ASCIIString,1}
typeof([1 2; 3 4])                #> Array{Int64,2}
```

Julia permite definir tipos parametrizados

```
type Point{T}
    x::T
    y::T
end

Point{Int}(1, 2)          #> Point{Int64}(1,2)
Point{Float64}(4.2, 2.1)  #> Point{Float64}(4.2,2.0)
```

Links interesantes

- http://learnxinyminutes.com/docs/julia/

  http://bogumilkaminski.pl/files/julia_express.pdf

- https://en.wikibooks.org/wiki/Introducing_Julia

- http://samuelcolvin.github.io/JuliaByExample/

Ejercicio

- Dada una matriz cuadrada NxN, cree una funcion que la entregue transpuesta.

```
A=[[1 2 3 4], [5 6 7 8], [9 10 11 12], [13 14 15 16]]
```

Matrix Int64, 4×4

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

```
B=transponer(A)
```

Matrix Int64, 4×4

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

Ejercicio

- Un polígono esta determinado por la lista de sus vértices. Escriba una función "perimetro(vertices)" que entregue el perímetro del polígono definido por la lista vertices:
- p = [(4, 1), (7, 2), (7, 4), (5, 9)]
- perimetro(p)
- 18.610

Ejercicio

- A partir de 4 archivos de texto, obtener el corpus de estos. Es decir todas las palabras distintas que contienen estos 4 archivos y escribirlo en un archivo nuevo que se llame “Corpus.txt”.

Fin Lección:

Métodos Numéricos

Profesor : Dr. Héctor Allende-Cid
e-mail : hector.allende@ucv.cl