CS423: Data mining

Introduction to Julia and PCA

Jakramate Bootkrajang

Department of Computer Science

Chiang Mai University


julia

ภาษา Julia

- เป็นภาษาโปรแกรมมิงที่ถูกพัฒนาขึ้นใหม่ เพื่อรองรับการคำนวณทางวิทยาศาสตร์
- การคำนวณที่ต้องเกี่ยวข้องกับ vector และ matrix สามารถทำได้โดยสะดวก
- เริ่มพัฒนาโดยกลุ่มของนักพัฒนาจากมหาวิทยาลัย MIT
- super fast.
- รองรับภาษาอื่นนอกจากภาษาอังกฤษ (Support unicode characters)
  - แปลว่าเราสามารถตั้งชื่อตัวแปรเป็นภาษาไทยได้

ถูกนำไปใช้งานในสาย Data science มากมาย

- Finance and Economics `https://lectures.quantecon.org/jl/`
- Machine learning `https://github.com/JuliaML`
- Statistics `https://github.com/JuliaStats`
- Bioinformatics `https://github.com/BioJulia`
- Astronomy `https://github.com/JuliaAstro`

การติดตั้ง Julia

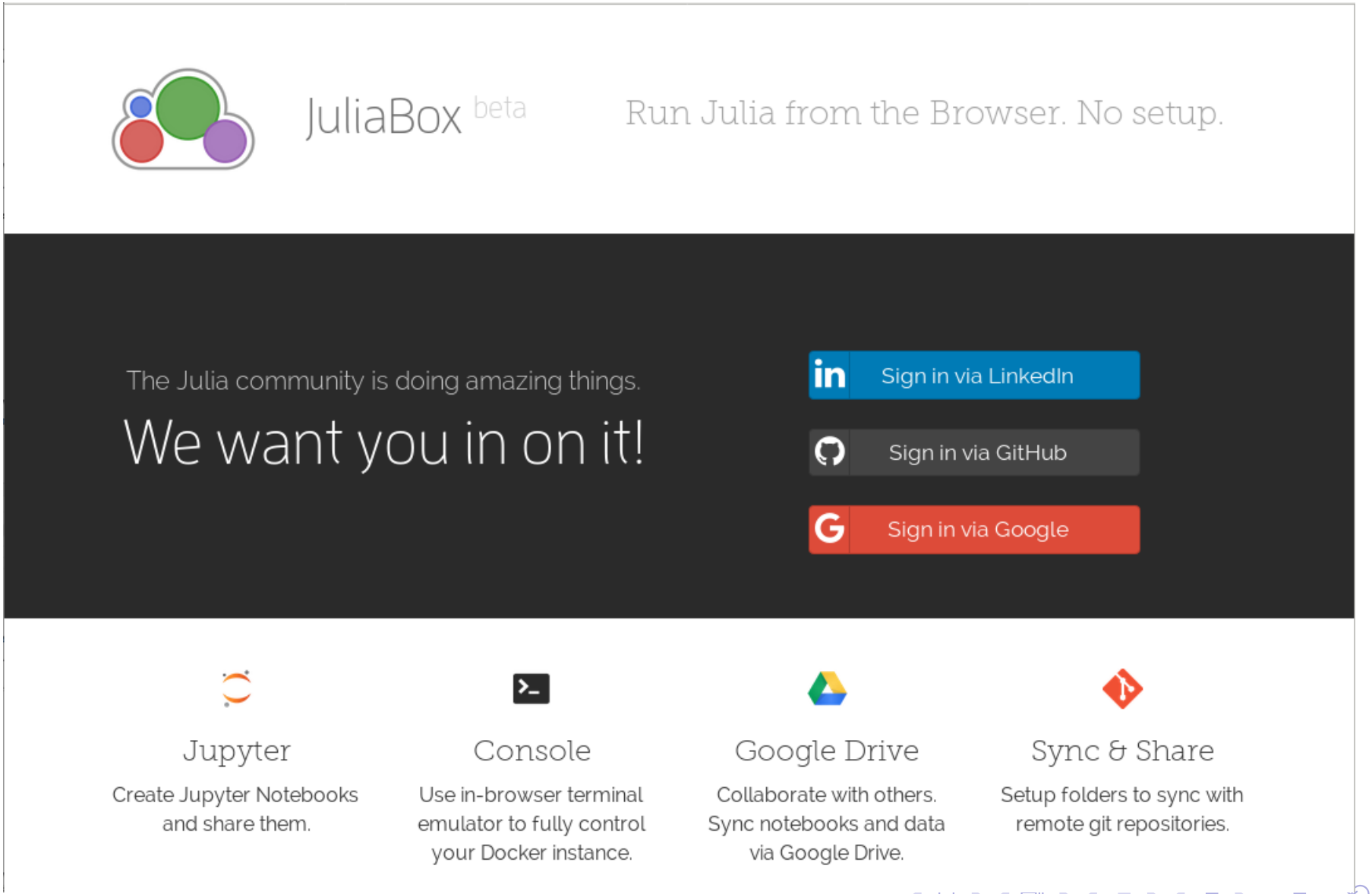
สามารถทำได้สองวิธีคือ

- การติดตั้งบนเครื่องของตนเอง
  - สามารถ download โปรแกรมแปลภาษา Julia ได้จาก

    `http://www.julialang.org`

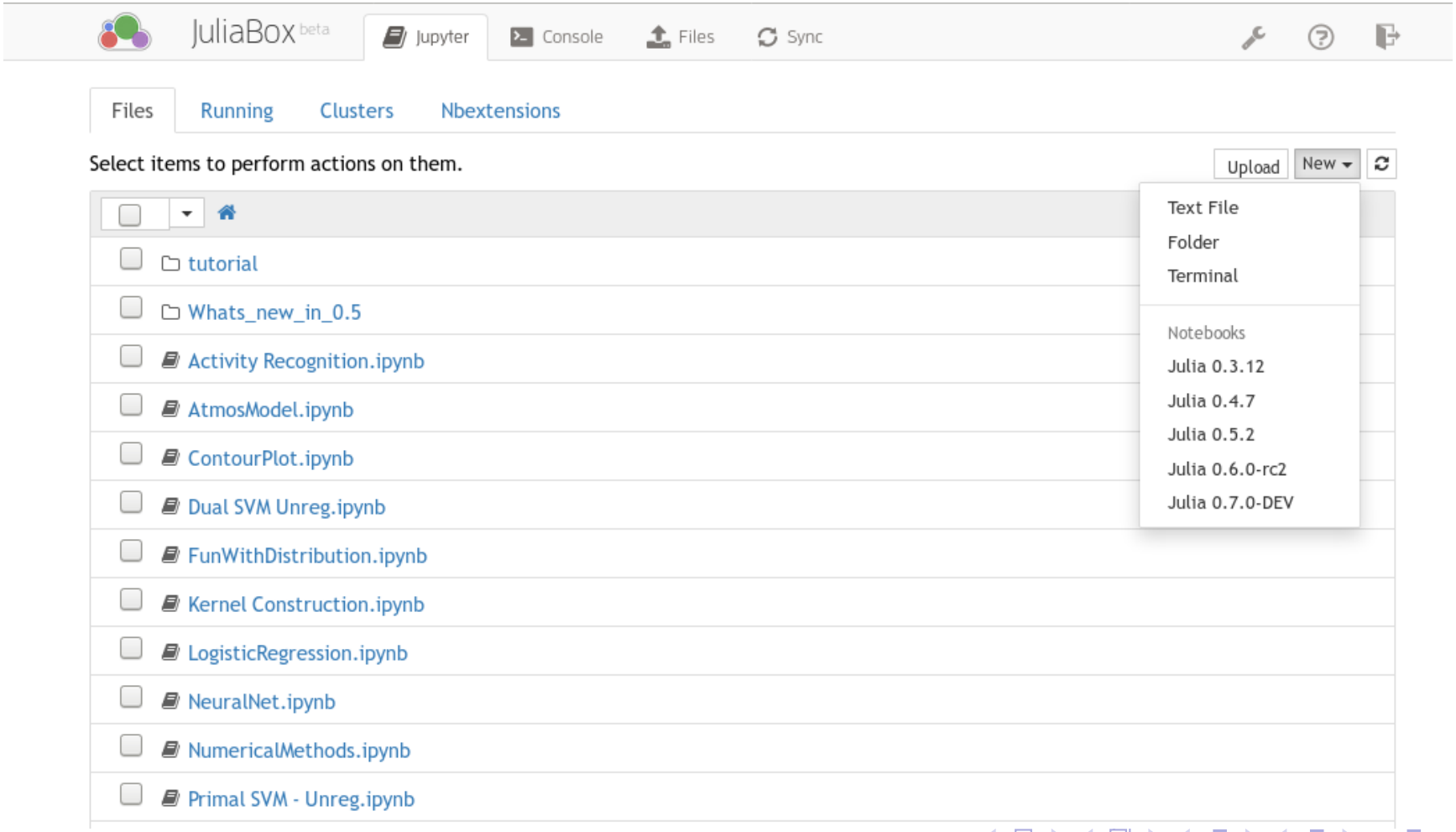
  - การใช้งานแบบนี้มีความยืดหยุ่นสูงและตอบสนองการใช้งานเร็วกว่า
- การใช้งาน Julia ผ่านเวป
  - `www.juliabox.com`
  - จำเป็นต้องมี account ของ google หรือ LinkedIn หรือ Github
  - ใช้งานได้ทันทีโดยไม่ต้องติดตั้งโปรแกรมบนเครื่องตนเอง

การเข้าใช้งาน Juliabox

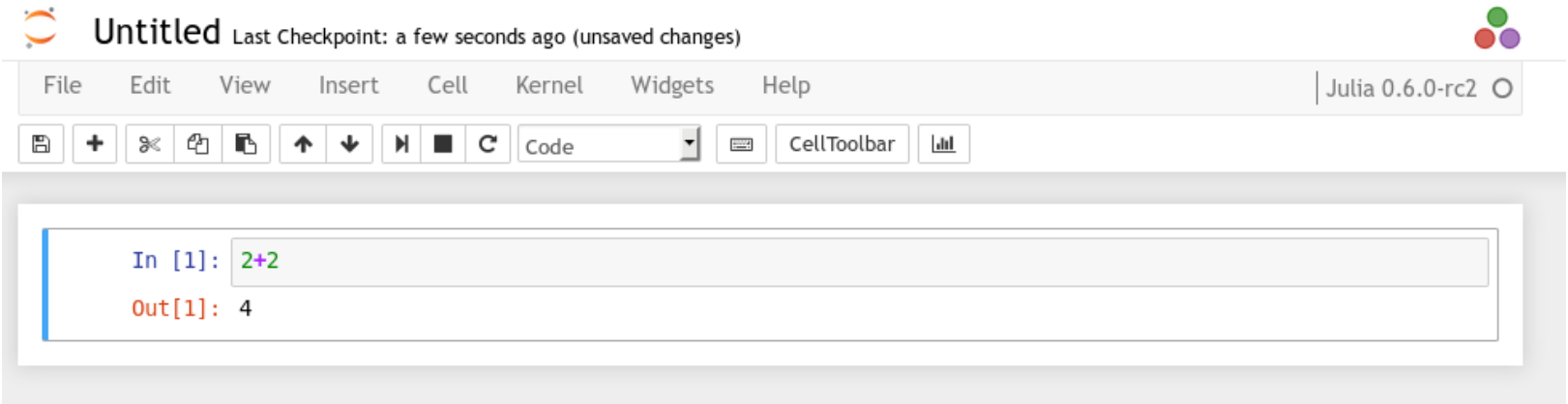


เริ่มสร้างไฟล์ใหม่

- สามารถทำได้โดยกด New -> แล้วเลือก version ของ Julia ให้เลือก 0.6.0-rc2

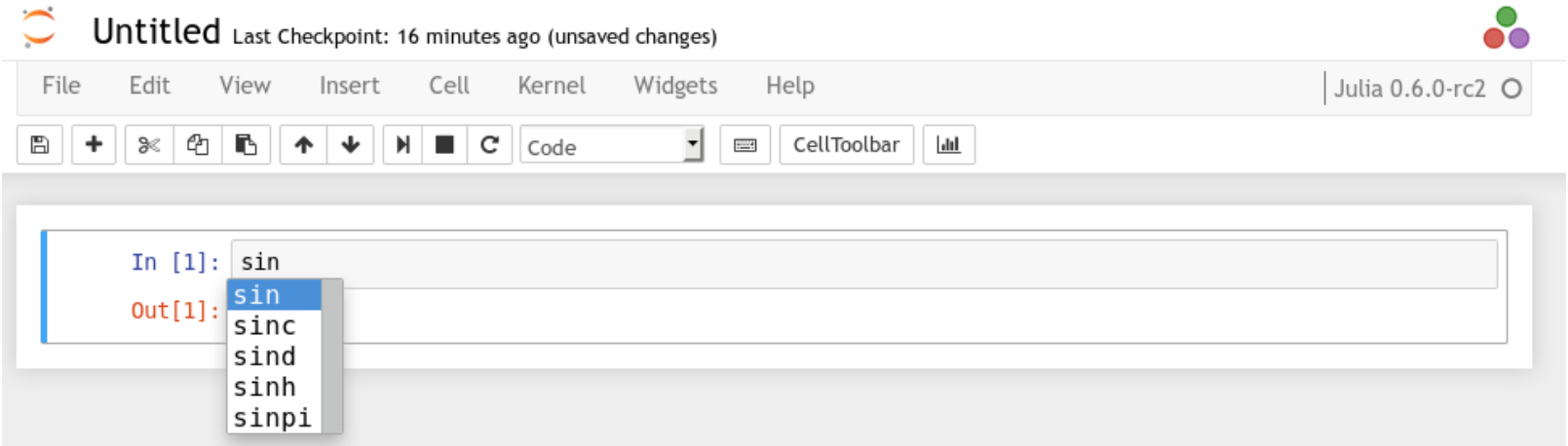


การใช้งานทั่วไป



- พิมพ์คำสั่งที่ต้องการลงในช่อง Input cell
- หากมีหลายคำสั่งสามารถกด Enter เพื่อใส่คำสั่งเพิ่มเติมในบรรทัดใหม่
- หากต้องการประมวลผลคำสั่งทั้งหมดที่ใส่ลงไป ให้กด Ctrl+Enter ผลลัพธ์จะแสดงด้านล่าง

การใช้งานทั่วไป



- ระหว่างพิมพ์คำสั่งสามารถกด Tab เพื่อให้ Julia แสดงคำสั่งที่ขึ้นต้นด้วยสิ่งที่พิมพ์ค้างไว้

Primitive data types

- Number:
  - Integers, Floating points
  - Special numbers: NaN (Not a number), Inf (Infinity)
- Array:
  - Vector: Array หนึ่งมิติ
  - Matrix: Array สองมิติ
- String
- ใช้ฟังก์ชัน typeof() เพื่อตรวจดูชนิดของตัวแปร

การสร้างอะเรย์

- สร้าง array โดยใช้ [ ] ครอบตัวเลข
  - X = [1 2 3 4] สร้าง 4x1 vector
  - Y = [1 2; 3 4] สร้าง 2x2 matrix
- สามารถเรียกดูมิติของ array โดยใช้ฟังก์ชัน `size()`
  - Vector: Array หนึ่งมิติ
  - Matrix: Array สองมิติ

Useful array creating functions

- `ones(n,m)`, `zeros(n,m)` สร้างเมทริกซ์ขนาด n x m ที่มีสมาชิกเป็น 1 หรือ 0 ทั้งหมด
- `eye(m)` สร้าง identity matrix
- `fill(k, n, m)` สร้างเมทริกซ์ขนาด n x m ที่มีสมาชิกทุกตัวมีค่าเท่ากับ k
- `randn(n,m)` สร้างเมทริกซ์ขนาด n x m ที่มีสมาชิกทุกตัวสุ่มจาก normal distribution

Array indexing

- `a = randn(4, 4)`
- `a[1, 1]` คือการดึงสมาชิกแถวที่ 1 คอลัมน์ 1
- `a[1, :]` คือการดึงสมาชิกทุกตัวของแถวแรก
- `a[:, 1]` คือการดึงสมาชิกทุกตัวของหลักแรก

Functions on array

```
x = [-1 0 1]
```

ฟังก์ชัน	ค่าที่ให้	ตัวอย่าง
`sum(x)`	หาผลบวกของสมาชิกในอะเรย์	`y = sum(x)`
`mean(x)`	หาค่าเฉลี่ยของสมาชิกในอะเรย์	`m = mean(x)`
`std(x)`	หา s.d. ของสมาชิกในอะเรย์	`t = std(x)`
`var(x)`	หา variance ของสมาชิกในอะเรย์	`t = var(x)`
`maximum(x)`	หาค่าสูงสุดของอะเรย์	`t = maximum(x)`
`minimum(x)`	หาค่าต่ำสุดของอะเรย์	`z = minimum(x)`
`sort(x)`	จัดเรียงค่าของสมาชิกในอะเรย์ x	`z = sort(x)`

Range

- สามารถสร้างลำดับของตัวเลขได้โดย start:stop
  - ตัวอย่างเช่น 1:10 จะทำการสร้างลำดับเริ่มต้นตั้งแต่ 1 ไปถึง 10
- สามารถกำหนดการเพิ่มของเลขโดยระบุ step เพิ่ม
  - ตัวอย่างเช่น 1:0.5:10 สร้างลำดับ 1,1.5,2,2.5,...,10
- สามารถเรียกดูลำดับได้โดยใช้ฟังก์ชัน collect()
  - collect(1:10)

Basic operations

สัญลักษณ์	หน้าที่	ตัวอย่าง
=	กำหนดค่าตัวแปร	`y = 5`
+	บวก	`y = x + 1`
-	ลบ	`z = m - 1`
*	คูณ	`k = 2 * 2`
/	หาร	`n = t / 10`
^	ยกกำลัง	`y = x^4`

หมายเหตุ จุดตามด้วยสัญลักษณ์ ใช้กระทำบนเมทริกซ์แบบตัวต่อตัว

Element-wise operation

- ใช้ . นำหน้า operator ทางคณิตศาสตร์
- `ones(2, 2) * ones(2, 2)` คือ Matrix multiplication
- `ones(2, 2) .* ones(2, 2)` คือ Element by element multiplication

Basic maths function

ฟังก์ชัน	ค่าที่ให้	ตัวอย่าง
`sqrt(x)`	รากที่สองของ x	`y = sqrt(x+5)`
`abs(x)`	ค่าสัมบูรณ์ของ x	`d = abs(x) * y`
`sin(x)`	ค่าไซน์ของ x (เรเดียน)	`t = x + sin(x)`
`asin(x)`	ค่า arcsin ของ x (เรเดียน)	`t = x + asin(x)`
`sind(x)`	ค่าไซน์ของ x (degree)	`t = x + sind(x)`
`log(x)`	ค่า natural log ของ x	`z = log(1+x)`
`log10(x)`	ค่า log ฐาน 10 ของ x	`z = log10(1-2*x)`
`exp(x)`	ค่า exponential ของ x (e^x)	`p = .7 * exp(x)`

Try-out: result = exp(-2.33) * cos(22 * 180 / pi)

Boolean expressions

- Expression ที่ประมวลผลเป็น จริง หรือ เท็จ (true / false)

ฟังก์ชัน	ค่าที่ให้	ตัวอย่าง
==	เป็นจริงเมื่อค่าสองค่าเท่ากัน x	`y == 5`
!=	เป็นจริงเมื่อค่าสองค่าไม่เท่ากัน	`7 != 7`
<=	เป็นจริงเมื่อค่าด้านซ้ายน้อยกว่าหรือเท่ากับค่าด้านขวา	`p <= 40`
>=	เป็นจริงเมื่อค่าด้านซ้ายมากกว่าหรือเท่ากับค่าด้านขวา	`p >= 2`
<	เป็นจริงเมื่อค่าด้านซ้ายน้อยกว่าค่าด้านขวา	`x < 5`
>	เป็นจริงเมื่อค่าด้านซ้ายมากกว่าค่าด้านขวา	`x+y+z > 16`
&&	คำเชื่อม และ	`x >= 5 && x <= 10`
||	คำเชื่อม หรือ	`x <= 5 || x >= 10`
!	นิเสธ	`!(5 == 5)`

Control structure: if

การเลือกทำโดยใช้ if...else

```
if (age >= 18)
    print("You can drive")
else
    print("Sorry you are too young")
end
```

สังเกตว่าจะต้องปิดด้วย end

Control structure: if...elseif

การเลือกทำโดยใช้ if...elseif

```
if (age >= 18)
    print("You can drive")
elseif (age >= 15)
    print("You can ride a motorcycle")
else
    println("Sorry you are too young")
end
```

สังเกตว่าจะต้องปิดด้วย end

Control structure: for

การทำซ้ำโดยใช้ for loop รู้จำนวนรอบในการทำซ้ำที่แน่นอน

```
for i in 1:10      # loop over some range
    println(e^i)     # print with newline
end


A = [1 3 5 7 9]   # loop over an array
for j in A
    print(j%2)       # print without newline
end
```

สังเกตว่าจะต้องปิดด้วย end

Control structure: while

การทำซ้ำแบบ while ไม่รู้จำนวนรอบที่แน่นอน ทำจนกว่า condition จะเป็นเท็จ

```
while (condition)
    some statements
end
```

```
p = 10  # initialise condition variable
while (p > 0)
    println(p/10)
    p = p - 1     # update condition variable
end
```

Functions

- กลุ่มของคำสั่งที่ถูกผูกกันไว้ พร้อมกับถูกตั้งชื่อ
- Useful functions
  - print(“put this on the screen”)
  - println(“with new line”)
  - readline()
  - typeof() เรียกดูชนิดของตัวแปร

การนิยามฟังก์ชัน

```
# inline function definition
f(x) = x^3
df(x) = 3*x^2


# more common function definition
function cube(x)
    y = x^3             # compute the cube of x
    return y            # return the result
end
```

Packages

- กลุ่มของฟังก์ชันที่ใช้ในการประมวลข้อมูลเฉพาะทาง
- Julia มี package มากมายสามารถดูข้อมูลได้จาก
  - Julia package listing — `https://pkg.julialang.org/`
- เช่น Package สำหรับการพลอตกราฟ
- การติดตั้ง Package ทำได้โดยคำสั่ง `Pkg.add("packageName")`
- ก่อนใช้งาน Package ต้องเรียกคำสั่ง `using packageName`
- Note: Julia is a case-sensitive language.

Useful packages: Plotting

- การสร้างกราฟใน Julia สามารถทำได้หลายวิธี มี package รองรับอยู่มาก
- ที่เราสนใจเป็นพิเศษคือ package ที่ชื่อว่า Plots ซึ่งเป็นคล้ายตัวกลางที่ประมวลและส่งผ่านคำสั่งในการสร้างกราฟไปให้ package อื่นที่ทำหน้าที่วาดกราฟ
- เราเรียก Plots ว่าเป็น front-end ส่วน package ที่ทำหน้าที่วาดกราฟ เราเรียกว่า back-end
- ตัวอย่าง back-end
  - PyPlot, GR, UnicodePlots, PlotlyJS

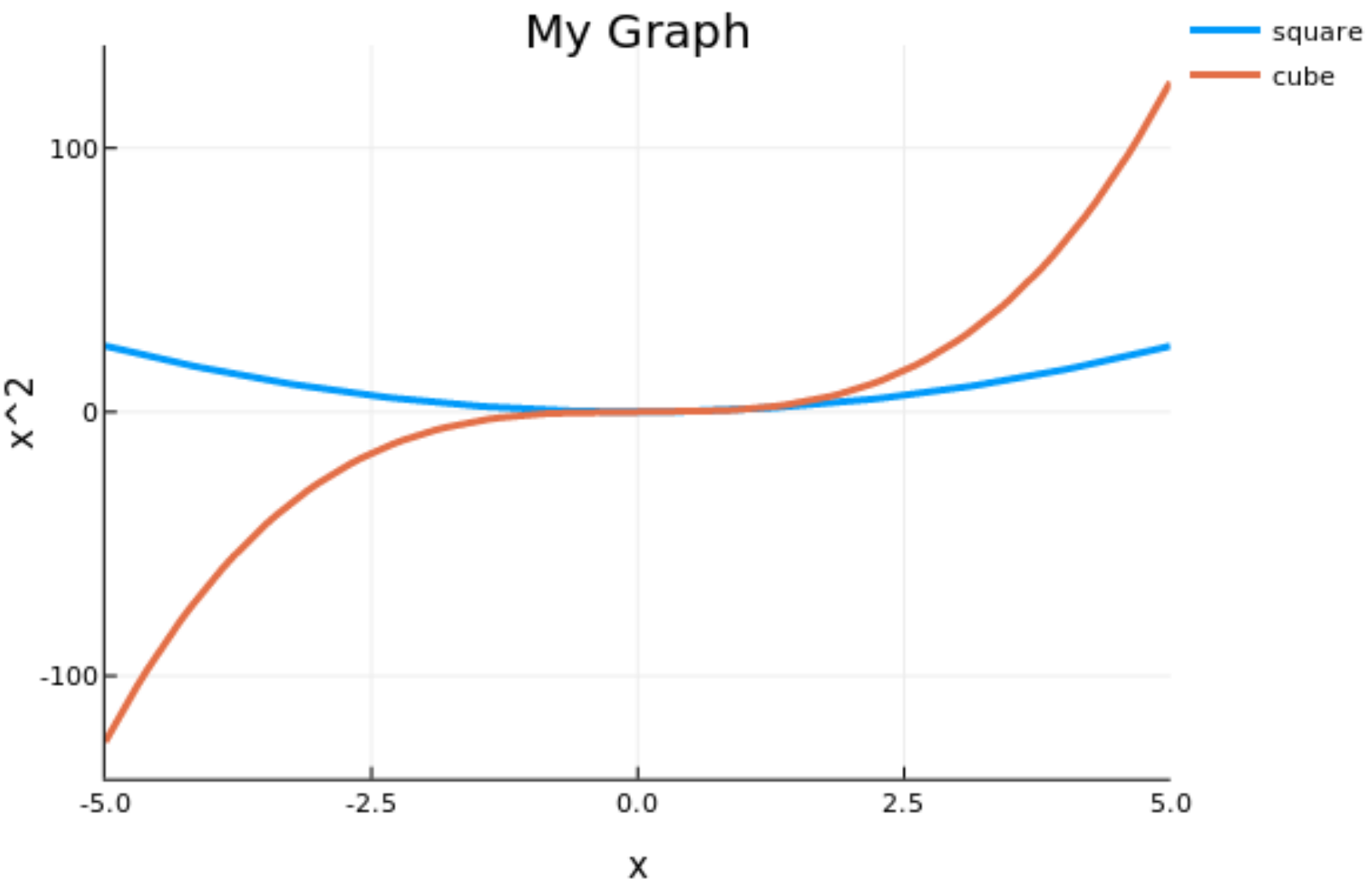
การติดตั้ง package ต่างๆเกี่ยวกับการวาดกราฟ

```
Pkg.add("Plots")          # front-end
Pkg.add("PyPlot")         # pyplot back-end
Pkg.add("GR")             # back-end
Pkg.add("PlotlyJS")       # back-end
```

การเรียกใช้งาน Plots และ default backend

```
using Plots             # front-end
# use default back-end
# or call gr() for GR back-end
# or plotly() for PlotlyJS back-end
f(x) = x.^2             # define a function
a = -5:0.1:5        # construct a range
plot(a,f(a),linewidth=3,
    title="My Graph", label="square",
    xlabel="x", ylabel="x^2")
plot!(a,a.^3,linewidth=3, label="cube")
```

The result



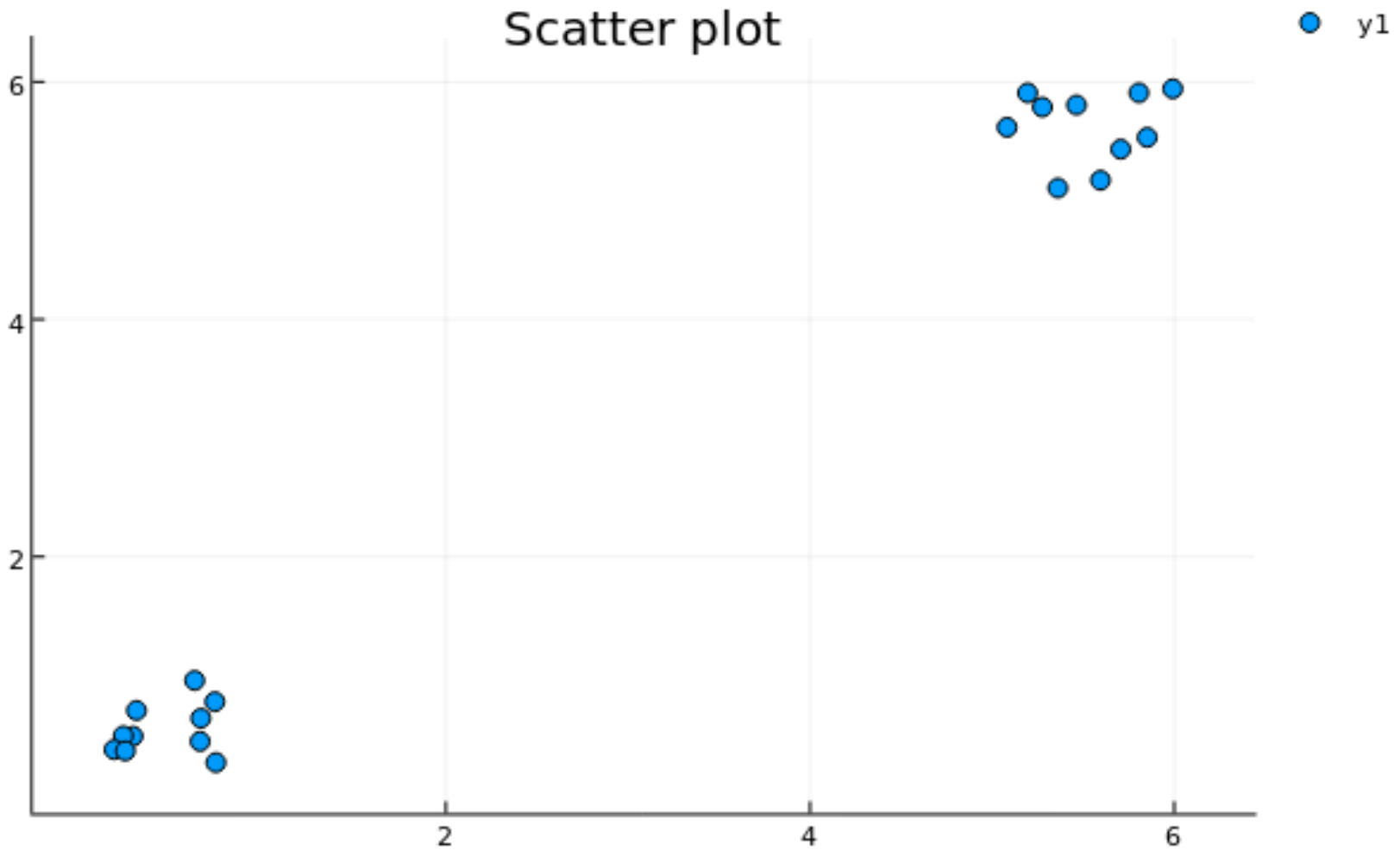
การเรียกใช้งาน Plots และ default backend

```
using Plots
plotly()

x = rand(10,2) + 5
x = [x; rand(10,2)]

scatter(x[:,1],x[:,2],title="Scatter plot")
```

The result



Principle Component Analysis

- Standardising X.
  - Subtract feature value by its mean and divide by its S.D.
- Calculate V, a covariance matrix for X
  - Useful function `cov(X)`
- Find all the eigenvectors of V.
  - Useful function `eigfact(V)`.
- Select k most important principle components put it in a matrix A.
- Project X onto A by calculating AX^T.

The `eigfact(V)` function.

- `F = eigfact([1.0 0.0 0.0; 0.0 3.0 0.0; 0.0 0.0 18.0])`
- Eigenvalues are stored in `F.values`
- Eigenvectors are stored in `F.vectors`

Reading Matlab's data file

- Install package
  - `Pkg.add("MAT")`
- Use package
  - `using MAT`
- Reading matlab's data file
  - `m = matread("filename.mat")`

Visualising m-dimensional data using PCA

```
d = matread("ionosphere.mat")  # reading data
# z-score normalisation
X = d["x"]
X = (X - repmat(mean(X,1),351,1))
          ./ repmat(std(X,1), 351, 1)
# calculating covariance
V = cov(X)
# finding eigenvectors and eigen values
F = eigfact(V)
```

Visualising m-dimensional data using PCA

```
# sort eigenvalues from max to min
# ord is the index of max to min
ord = sortperm(F.values, rev=true)

# project data
A = F.vectors[:, ord[1:2]]
P = A' * d
```

Visualising m-dimensional data using PCA

```
# find index of class -1 and 1
c1idx = squeeze(m["y"]==1,2)
c2idx = squeeze(m["y"]==-1,2)
# project data
A = F.vectors[:, ord[1:2]]
P = A' * d
scatter(P[c1idx,1],P[c1idx,2],title="Ionosphere")
scatter!(P[c2idx,1],P[c2idx,2])
```

Must read

- Learn X in Y minutes for Julia

`https://learnxinyminutes.com/docs/julia/`