🫶🏻 U1FAF6-project

None

None

None

Table of contents

1. Welcome to 🫶 U1FAF6-project!

- JW
- 2025-09-01
- 2025-09-02

🫶 U1FAF6-project (heart hands).

我想長期維護一個存放整理好的、可公開文件的平台。

維護一個這樣的平台好處是，讓我定期整理我的想法。 把一些「抽屜裡的」、筆記軟體裡面的、雜亂的筆記整理一下。 整理成完整的一份文件。

2. 一種半自動的最小可行寫作方案

- JW
- 2025-09-01
- 2024-02-16

原文：單純使用 `.md` 與 `.ipynb` 格式寫作的最小可行方案

2.1 Problem.

你想用一個簡單隨手可得的編輯器（例如 colab，因為可用 md, ipynb, 可支援 python, r, julia），並且寫好後可以一鍵發佈。

你想單純使用 .md 紀錄純文字、簡單的數學式 (支援 latex)、表格與程式。如果需要有執行結果的，則用 .ipynb 的檔案來紀錄和執行。最後，只需要最非常簡單的事情，甚至不用轉檔，就可以彙整。

我這麼說，就是因為 `Quarto` 也很好用，但他是要轉檔。quarto 在 rstudio 上寫好後，也是可以一鍵發布到 Quarto Pub 上面。 而且可調整性更高。 但是他就是還是需要安裝 r, rstudio, quarto 等軟體，而且主要是在本機上寫，比較難跨機器。如果只是想做簡單編輯、改幾個字，還得大費周章開啓 rstudio.

一種最小可行方案： `mk` / `ipynb` --> `github` --> `mkdocs` --> `readthedocs` 。

這種方法的好處是，編輯器取得非常容易。只要在 google drive 中開啓即可。或是在 github 上直接寫 md. 後續的事只要第一次設定好，基本就是半自動了。

2.2 Solution.

使用 colab 當編輯器，用 github 當後台，用 mkdocs 當引擎，用 readthedos 當部署。

Step 1. Install MkDocs

安裝方式，在 mac 系統上，用 homebrew 就可以了。

```
# general brew install mkdocs # for material theme brew install mkdocs-material
```

在 windows 系統上，就參考官網： mkdcos

```
pip install mkdocs
```

Step 2. Usage

使用上，就按照官網指示： 建立一個新資料夾。

```
# create a new project mkdocs new my-project # go into the project's file cd my-project # show in your local computer mkdocs serve
```

Step 3. Add Files

- 使用 markdown 檔案應該是基本可行的。
- 使用 jupyter 檔案則加上外掛就可行。（如下節所示）

Step 3.1. Add a `requirement.txt` in the docs file, 這一步是為了安裝所需要的 python 套件。例子：

```
mkdocs mkdocs-jupyter mkdocs-material
```

Step 3.2. Add a `.readthedocs.yaml` in the file

這是上傳到 readthedocs 上時會需要的。通常部署時參考他建議的寫法就行。例子：

```
# Read the Docs configuration file for MkDocs projects # See https://docs.readthedocs.io/en/stable/config-file/v2.html for details # Required version: 2 #
```

Step 3.3. change your `mkdocs.yml`

最後，就是修改 `mkdocs.yml` 中的設定。 因為本文目標是「最小可行」，因此就是字型什麼幾乎不改，目錄也不改，都用預設值。唯二需要增加的是：

- latex 語法支援。詳細設定方式請參考 Material for MkDocs - Math
- jupyter 格式支援。在 plugins 加入 `- mkdocs-jupyter`。

例子：

```
site_name: YOUR_SITE_NAME ## if you need to chage #nav: # - ipynb: # - ch3: ch3.ipynb theme: readthedocs ## mathjax for .md markdown_extensions: - footnot
```

關於 markdown_extensions 可以寫什麼，可以參考 Material for MkDocs - Python Markdown Extensions

Step 4. 上傳到 GITHUB

這邊可以用任何你知道的方法，把本機的資料夾上傳到 Github 上面。只需這一次就好。後面的步驟，都是在線上操作，基本上不需要碰到本機的資料夾。

- Github Desktop.

Step 5. 部署到 readthedocs

完成之後，只需要再做幾件事:

1. 把這個資料夾上傳到 Github 上面。
2. 登入 readthedocs，連結你的 github。選擇這個 repo，然後發佈即可。
3. 進入後的 NAME 要注意，他會變成這份文件網址的主要名稱。
4. 在部署選項上，記得要改成 `MkDocs` !!
5. 部署好，網址就會是 `xxxxx.readthedocs.io/` 非常方便好記。

現在，一個簡單而基本可行的介面已經完成了！ 接下來只需要把你的 markdown 和 jupyter 檔案無腦放入 `docs` 資料夾中上傳即可！

Step 6.1. COLAB 使用方式

1. 在 google drive 開啓一個 colab 檔案。
2. 完成你的 md 或是 r, pyhton, julia 作業。
3. 選擇 `File/Save a copy in Github`, 接著選你要存放的 repo.
4. 這時候檔名前面記得要加 `docs/` 因為要存到該 repo 的 docs 資料夾中。
5. 檔案命名方式，前面記得加上日期，例如 `20250901_xxxxx.ipynb` 這樣有利於 mkdocs 自動按日期順序排版。
6. 萬一有相同日期，就在後面加上數字，例如 2024021401, 2024021402, ...

Step 6.2. 當然也可以使用 GITHUB

github 的話，就是直接編輯 markdown 檔案。

2.3 Discussion

這套半自動寫作方案的核心概念是「降低進入門檻」。降低進入門檻的理由，並不完全是技術上的原因。如果要講步驟簡單，依賴工具少，那可能 Quarto 是更好的選擇。對我而言更重要的原因是有利於「原子習慣」的養成。 相較於 Quarto 需要完整的本機開發環境，這個方案只需要瀏覽器就能開始寫作。特別適合需要跨裝置工作、或是偶爾想修改幾個字但不想開啓重型軟體的情況。這樣簡單的方式有利於習慣建立。

當然，這個方案也有取捨。在客製化和進階功能上不如 Quarto 豐富，但對於大多數的學術寫作、筆記整理、或是簡單的技術文件來說已經足夠。重點是一旦設定完成，後續的發布流程幾乎是透明的——寫完就直接同步到網站上。

另一個意外的好處是版本控制。由於所有內容都在 GitHub 上，自然就有了完整的修改歷史記錄，這在傳統的文件編輯軟體中往往需要額外設定。

對於習慣用 Colab 做資料分析的研究者來說，這個方案讓寫作和分析可以在同一個環境中完成，減少了工具切換的成本。

2.4 See Also

- MkDcos
- Quarto
- readthedocs

2.5 Chapter 2

原來他會自動排目錄!

討論一些 We get a significant effect ($p < .05$)

$\alpha + \beta \to \gamma$

```
using Distributions a = rand(Normal(0,1), 100) a
```

```
print(a)
```

2.6 Chapter 3

We get a significant effect ($p < .05$)

$$\alpha + \beta \to \gamma$$

Give a more complex example $\begin{pmatrix} \theta \\ \zeta \end{pmatrix} = N \left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix} \right)$

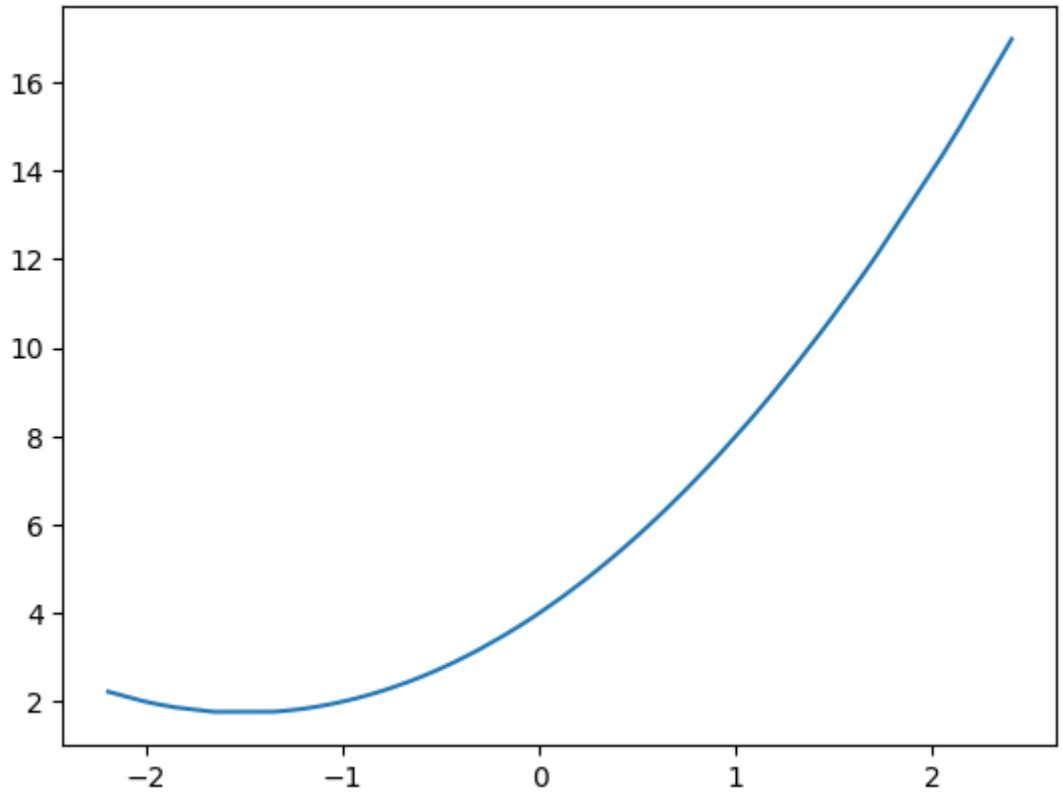
In [11]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [12]:
```
f = lambda x: x**2 + 3*x + 4

x = np.random.normal(size=100)
x.sort()
```

In [13]:
```
plt.plot(x, f(x))
```

Out[13]: [<matplotlib.lines.Line2D at 0x10b015400>]



In [ ]:
```

```

3. Weekly Chellange

3.1 0721

3.1.1 Problem

你想測試一下 Doran (2023) 文章中所說的 *OLS as a Unifying Framework* 這一小節。

3.1.2 Solution

Theory. 一個一般的解線性方程式的公式如右 X'Xβ = X'y, 因此我們可以得到 β 的估計值為 $\bf \hat \beta=(X'X)^{-1}X'y$. 而 Doran (2023) 文章中提到，很多文章都是用 $\bf (X'X)^{-1}$. 然而更快速更穩定的方法不是用 $\bf (X'X)$ 的 invert 而是應該去找 $\bf (X'X)$ 的分解，$\bf (X'X)=LL'$，其中 L 是 Cholesky factor 的下三角。

3.1.3 See Also

- Doran, H. (2023). A collection of numerical recipes useful for building scalable psychometric applications. Journal of Educational and Behavioral Statistics, 48(1), 37-69.

In [17]:
```
X = matrix(rnorm(300), nrow=100)
B = rnorm(3)
y = X %*% B + rnorm(100)
cat('True B: ', B)
```

```
True B:  1.302446 1.219099 2.64538
```

In [18]:
```
## X'X
t(X) %*% X
```

A matrix: 3 × 3 of type dbl

108.376385	-8.716593	-6.680255
-8.716593	121.026511	-6.408469
-6.680255	-6.408469	83.483773

In [22]:
```
## find cholesky factor
L = t(X) %*% X |> chol() |> t()
L %*% t(L)
```

A matrix: 3 × 3 of type dbl

108.376385	-8.716593	-6.680255
-8.716593	121.026511	-6.408469
-6.680255	-6.408469	83.483773

In [27]:
```
## Estimated B
solve(t(X) %*% X) %*% t(X) %*% y
```

A matrix: 3 × 1 of type dbl

1.339048
1.107174
2.587583

In [24]:
```
# 正確的兩步求解
XTy = t(X) %*% y
z = forwardsolve(L, XTy)      # 解 L*z = X^T*y
B_hat = backsolve(t(L), z)    # 解 L^T*β = z

print(B_hat)
```

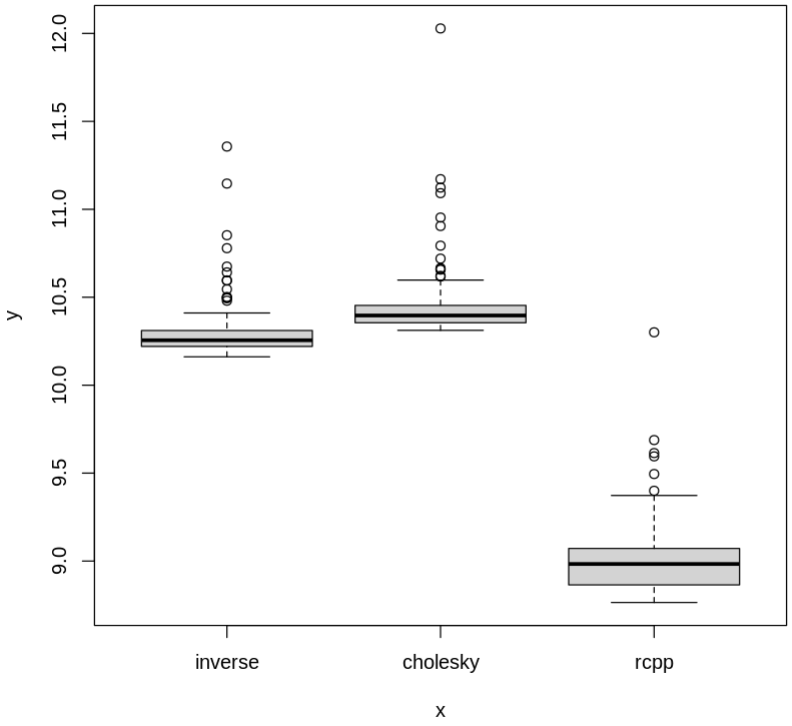
```
        [,1]
[1,] 1.339048
[2,] 1.107174
[3,] 2.587583
```

In [26]:
```
system("sudo apt install r-cran-microbenchmark")
```

In [28]:
```
library(microbenchmark)
```

In [85]:
```
bench <- microbenchmark(
  inverse = solve(t(X) %*% X) %*% t(X) %*% y,
  cholesky = {
    L = t(X) %*% X |> chol() |> t()
    XTy = t(X) %*% y
    z = forwardsolve(L, XTy)      # 解 L*z = X^T*y
    B_hat = backsolve(t(L), z)    # 解 L^T*β = z
  },
  rcpp = solve_two_steps(X,y),
  times = 100
)
```

In [89]:
```
plot(bench$expr, log(bench$time))
```



Step 2. (用 Rcpp 寫)

In [32]:
```
system("sudo apt install r-cran-rcpparmadillo")
```

In [49]:
```
library(Rcpp)
library(RcppArmadillo)
```

In [75]:
```
rcpp_chol <- '
#include <RcppArmadillo.h>
// [[Rcpp::depends(RcppArmadillo)]]

using namespace arma;

// [[Rcpp::export]]
arma::mat find_chol(arma::mat X){
  // find cholesky factor
  arma::mat XTX = X.t() * X;
  arma::mat L = chol(XTX);

  return L.t();
}

// [[Rcpp::export]]
arma::mat solve_two_steps(arma::mat X, arma::mat y){
  arma::mat XTy = X.t() * y;
  arma::mat L = find_chol(X);
  // 两步求解
  arma::mat z = solve(trimatl(L), XTy);
  arma::mat beta = solve(trimatu(L.t()), z);

  return beta;
}

'
```

In [76]:
```
sourceCpp(code = rcpp_chol)
```

In [77]:
```
find_chol(X)
```

A matrix: 3 × 3 of type dbl

10.4103979	0.0000000	0.000000
-0.8372968	10.9692955	0.000000
-0.6416906	-0.6331997	9.092363

In [78]:
```
solve_two_steps(X,y)
```

A matrix: 3 × 1 of type dbl

1.339048
1.107174
2.587583

3.2 0722

3.2.1 Problem

3.2.2 Solution

3.2.3 Discussion

3.2.4 See Also

3.3 0723

In [ ]:

3.4 0724

In [ ]:

4. R Performance Benchmark Results

- JW
- 2025-09-01

這是一個長期的比較結果。

4.1 Performance Summary (Sorted by Speed)

Rank	Device	Time (min)	Time (sec)	Speed vs M2 Mini	CPU	RAM	OS	R Version
1	**Mac mini M4**	0.563	33.8	**0.75x**	M4	16 GB	macOS Sequoia 15.6.1	R 4.5.1
2	MacBook Air M3	0.633	38.0	0.85x	M3	8 GB	macOS Ventura	R 4.4.0
3	**Mac mini M2** *(baseline)*	0.748	44.9	**1.00x**	M2	16 GB	macOS Ventura 13.3	R 4.3.1
4	Intel i5-12400 (Ubuntu)	1.068	64.1	1.43x	i5-12400	64 GB	Ubuntu 22.04.3 LTS	R 4.3.1
5	Intel i5-12400 (Windows)	1.278	76.7	1.71x	i5-12400	64 GB	Windows 11 x64	R 4.3.1
6	Intel i7-10700 (Windows)	1.885	113.1	2.52x	i7-10700 2.90GHz	16 GB	Windows 10 x64	R 4.3.1
7	Intel i5-10500 (Windows)	2.230	133.8	2.98x	i5-10500 3.10GHz	16 GB	Windows 10 x64	R 4.3.1
8	Google Colab	2.868	172.1	3.83x	x86_64	-	Ubuntu 22.04.4 LTS	R 4.5.1
9	MacBook Air (Intel)	5.563	333.8	7.44x	Intel i5 1.6GHz	8 GB	macOS Ventura 13.6	R 4.3.1

4.2 Key Insights

4.2.1 🏆 Top Performers

- **Mac mini M4**: 25% faster than M2, best overall performance
- **Apple Silicon dominance**: M4/M3/M2 占據前三名
- **M3 vs M2**: 僅管 M3 只有 8GB RAM，仍比 M2 (16GB) 快 15%

4.2.2 💻 Intel Performance

- **Same CPU, different OS**: i5-12400 在 Ubuntu 比 Windows 快 20%
- **RAM impact**: 64GB RAM 的 i5-12400 仍比 Apple Silicon 慢 43%+
- **Older Intel**: i7-10700/i5-10500 明顯落後於新 i5-12400