

INSTITUTO TECNOLÓGICO AUTÓNOMO DE MÉXICO



Una tesis extendida ($\overline{\text{tesis}}$)

TESIS

QUE PARA OBTENER EL TÍTULO DE

LICENCIADO EN MATEMÁTICAS APLICADAS

PRESENTA

VALERIA AURORA PÉREZ CHÁVEZ

ASESOR: ERNESTO JUVENAL BARRIOS ZAMUDIO

«Con fundamento en los artículos 21 y 27 de la Ley Federal del Derecho de Autor y como titular de los derechos moral y patrimonial de la obra titulada “**Una tesis extendida ($\overline{\text{tesis}}$)**”, otorgo de manera gratuita y permanente al Instituto Tecnológico Autónomo de México y a la Biblioteca Raúl Baillères Jr., la autorización para que fijen la obra en cualquier medio, incluido el electrónico, y la divulguen entre sus usuarios, profesores, estudiantes o terceras personas, sin que pueda percibir por tal divulgación una contraprestación.»

VALERIA AURORA PÉREZ CHÁVEZ

FECHA

FIRMA

Agradecimientos

Agradezco a facu por ser tan chingona y a Mike por pasarme el formato. Salu2.

EL SERCH

Índice general

1. Introducción	1
2. Julia	2
2.1. Instalación	2
2.1.1. Windows	3
2.1.2. Mac	3
2.2. Atom	3
2.2.1. Instalación en Windows	4
2.2.2. Conexión con Julia	5
2.3. Símbolo del sistema	6
2.3.1. <i>Multithreading</i>	6
2.4. Jupyter	7
2.5. Básicos de Julia	8
2.5.1. Operaciones básicas	9
2.5.2. <i>Strings</i> (secuencias de caracteres)	10
2.5.3. Funciones	11
2.5.4. Vectores y Matrices	12
2.5.5. <i>DataFrames</i>	14
2.5.6. Regresiones	17
2.5.7. Paquete Distributions	22

3. <i>Polynomials</i>	23
3.1. El problema	23
3.2. Método Gauss-Newton	24
3.3. Ejemplo	24
4. R	26
4.1. Paquete JuliaCall	26
4.1.1. install_julia	26
4.1.2. julia_setup	26
4.1.3. Núcleos y threads	27
4.1.4. JuliaObject	27
4.1.5. julia_assign	27
4.1.6. julia_eval	27
4.1.7. julia_package	27
4.2. Ejemplo	27
5. Computadoras virtuales	28
5.1. Amazon	28
5.2. Docker	28
A. Extras	29

Índice de algoritmos

Índice de tablas

Índice de figuras

2.1. Ventana principal de Atom	4
2.2. Ventana inicial de Jupyter	8
2.3. Sintaxis básica de una función en Julia.	11

Capítulo 1

Introducción

Historia de Julia como programa, historia de R. También hay que decir que esto no es un manual de Julia ni de R ni de Python.

Capítulo 2

Julia

Julia es un lenguaje de programación gratis cuyo propósito general es ser tan rápido como C, pero manteniendo la facilidad de lenguaje de R o Python. Es una combinación de sintaxis simple con alto rendimiento computacional. Su slogan es "Julia se ve como Python, se siente como Lisp, corre como Fortran"(Carrone et al., 2021). Esta combinación de características hace que Julia sea un lenguaje de programación que ha tomado mucha fuerza en la comunidad científica. Por lo tanto, en esta sección explicaré los básicos de Julia, desde su instalación hasta análisis de regresiones multivariadas.

2.1. Instalación

Al momento de escritura de esta tesis la versión de Julia disponible es la v1.6.3 hasta el 23 de Septiembre del 2021. Para descargar Julia, hay que entrar al siguiente link: <https://julialang.org/downloads/>.

2.1.1. Windows

Para esta versión de Julia, las opciones disponibles de descarga son un instalador de 64-bits o uno de 32-bits. Para saber el tipo de Sistema que tiene tu ordenador de Windows, debes seleccionar el botón de **Start**, después **Configuración > Sistema > Acerca de**. En esta opción puedes ver el tipo de sistema que tiene tu computadora. Con esta información, eliges el instalador para tu computadora. Es importante seleccionar el **installer** y no el **portable**. Una vez descargado, seleccionas el archivo **.exe** y sigues los pasos de instalación.

2.1.2. Mac

Vale: No voy a abarcar tanto MAC porque no tengo Mac jaja

Para el sistema operativo de Mac, solo hay una versión disponible de descarga. Selecciona la opción de 64-bit. Una vez descargado, mueve el archivo de Julia al folder de “Aplicaciones”. Para verificar que está bien descargado, haz click en el ícono de Julia y debe salir una terminal como la siguiente:

2.2. Atom

Atom es un editor de texto de programación. Su meta es *no dificultar su uso ni posibilidad de modificación* (??, ATO). Busca ser lo suficientemente fácil para que cualquier persona, ya sea principiante o experto en programación lo pueda utilizar. Por lo tanto, es una buena herramienta para escribir, editar y documentar el trabajo que uno hace en Julia.

Vale:
Puedo copiar una imagen que no sea mía, sino de una página web? referenciando obvio

Vale:
No estoy segura de como arreglar

2.2.1. Instalación en Windows

Por supuesto, el primer paso es instalar el programa. Para hacerlo, hay que ir a la página <https://atom.io/> y hacer clic en el botón de Download.

Una vez descargado el archivo ejecutable solo hay que darle clic. Este proceso puede tardar un par de minutos. Sin embargo, una vez que termine el programa se ejecutará de manera automática. La pantalla de inicio se ve de la siguiente manera:

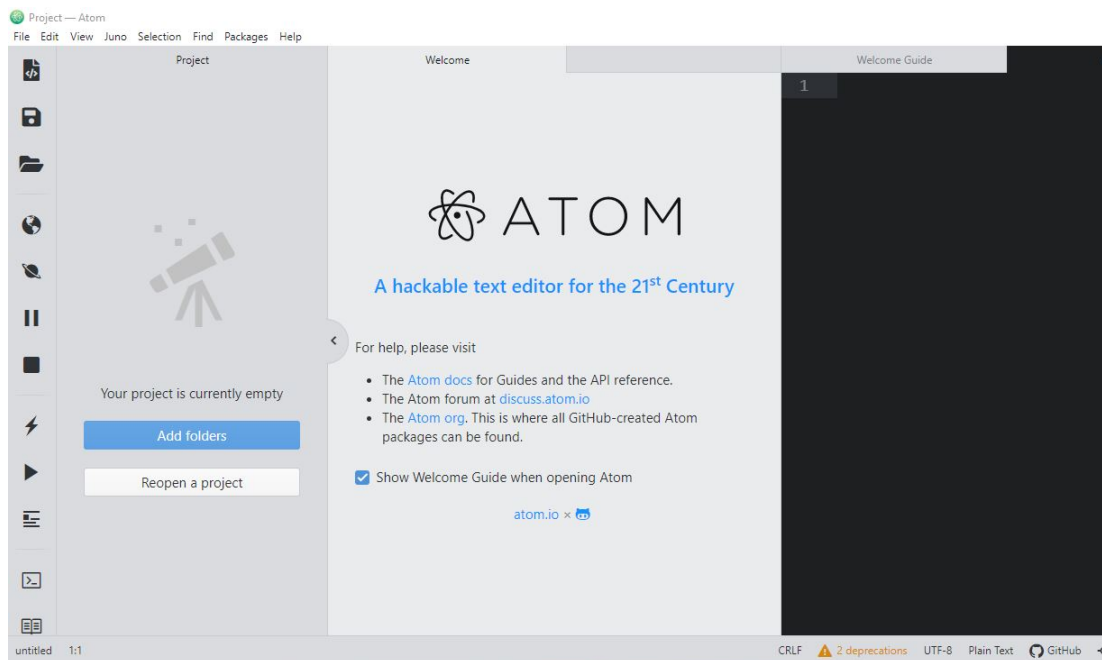


Figura 2.1. Ventana principal de Atom

El usuario puede modificar la pantalla de Atom con diferentes colores y temas siguiendo las instrucciones que vienen en la página <https://flight-manual.atom.io/getting-started/sections/atom-basics/>.

Sin embargo, esa información no es relevante para esta tesis por lo que se va a omitir.

2.2.2. Conexión con Julia

Ahora es momento de enlazar Julia con Atom. Para hacer esto, es necesario instalar el paquete Juno.

1. Abre Atom y selecciona File seguido de Settings.
2. En el panel izquierdo selecciona la opción de Install y en el buscador de Install Packages escribe *uber-juno*.
3. Asegurate de que sea el paquete generado por JunoLab y haz click en el botón de Install.

Una vez instalado Juno, va a aparecer una opción llamada Juno en la parte superior de opciones entre View y Selection. Para correr Julia dentro de Atom debes seleccionar Juno y después Open REPL. Aparecerá una nueva ventana en la parte inferior de la pantalla que te dirá que oprimas enter para iniciar una sesión de Julia. Después de hacerlo, Julia iniciará y podrás usarlo de manera normal.

Una ventaja de Atom es utilizarlo por su función principal: ser un editor de texto de programación. Es una forma fácil y rápida de crear código de Julia y el usuario puede decidir si correrlo dentro de Atom o directo en Julia. Para crear un nuevo archivo, hay que seleccionar la opción de File en la parte superior izquierda y después la opción de New File. Para que Atom reconozca ese archivo como texto del lenguaje Julia hay que guardarlo con el nombre deseado agregando `.jl` al final del nombre.

2.3. Símbolo del sistema

Otra opción (mi favorita) es correr Julia desde el símbolo del sistema (también conocido como *Command Prompt* o *cmd*). De ahora en adelante, se usarán por igual los términos símbolo del sistema y *cmd*. Para facilitar el uso y trabajo en Julia, recomiendo agregar Julia a un *PATH*. Las instrucciones para hacerlo en Windows 10 están en la página <https://julialang.org/downloads/platform/#windows>.

2.3.1. *Multithreading*

Una de las razones por las que Julia es un lenguaje de programación muy rápido es que tiene la capacidad para multihilo (*multithreading* en inglés). Esto significa que puede correr diferentes tareas de manera simultánea en varios hilos. De la manera más simple: la meta de los autores de Julia fue hacer un lenguaje de programación con un rendimiento tan alto que pudiera hacer varias cosas a la vez. Debido a que uno de los objetivos de esta tesis es medir la eficiencia y velocidad de Julia, es crucial conocer la característica del *multithreading* y como utilizarla.

Juno automáticamente pone la misma cantidad de hilos que la cantidad de núcleos de procesadores disponibles (Bezanson et al., 2019). Sin embargo, esto se puede modificar entrando a **File > Settings > Packages**. Después, buscar el paquete *julia-client* y seleccionar los **Settings** del paquete. Finalmente, dentro del apartado de **Julia Options** buscar **Number of Threads** y poner el número deseado.

Por otro lado, si se está usando el símbolo de sistema el camino es diferente. Antes de ejecutar *julia*, es necesario modificar la cantidad de hilos que se van a utilizar. En Windows, esto se hace escribiendo `set JULIA_NUM_THREADS=4` (??, Jul) En este ejemplo, se cambiaron los

hilos a 4, pero se puede poner cualquier número. Sin embargo, se recomienda que el número no exceda de la cantidad de procesadores físicos de la computadora. Después de hacer esta modificación, ahora sí podemos ejecutar Julia.

Para observar que el cambio se ejecutó de manera correcta (en cualquiera de las dos opciones) basta con correr el comando `Threads.nthreads()` y observar que la respuesta sea el número deseado.

2.4. Jupyter

Otra forma de correr Julia es utilizando Jupyter. Jupyter es un programa que existe para desarrollar software de manera pública en decenas de lenguajes de programación. Una de las ventajas de utilizar Jupyter para esta tesis es que se pueden utilizar los tres lenguajes de programación (R, Python y Julia) en un mismo software.

La manera más fácil para obtener Jupyter es instalando Anaconda. Anaconda es una interfaz gráfica que permite manejar y administrar aplicaciones, paquetes, ambientes y canales sin necesidad de usar comandos en el cmd. Para instalar Anaconda en Windows hay que ir a esta página <https://docs.anaconda.com/anaconda/install/windows/> y seguir las instrucciones de instalación. La instalación de este programa puede tomar unos cuantos minutos.

Una vez instalado Anaconda, basta con ejecutarlo y después seleccionar **Launch** debajo del ícono de Jupyter Notebook. Se abrirá una ventana en tu navegador como la siguiente:

Para poder trabajar con Julia en Jupyter, hay que instalar el paquete `IJulia` como en las instrucciones

Vale:

Viene de la página oficial de Jupyter que todavía no tengo referencia

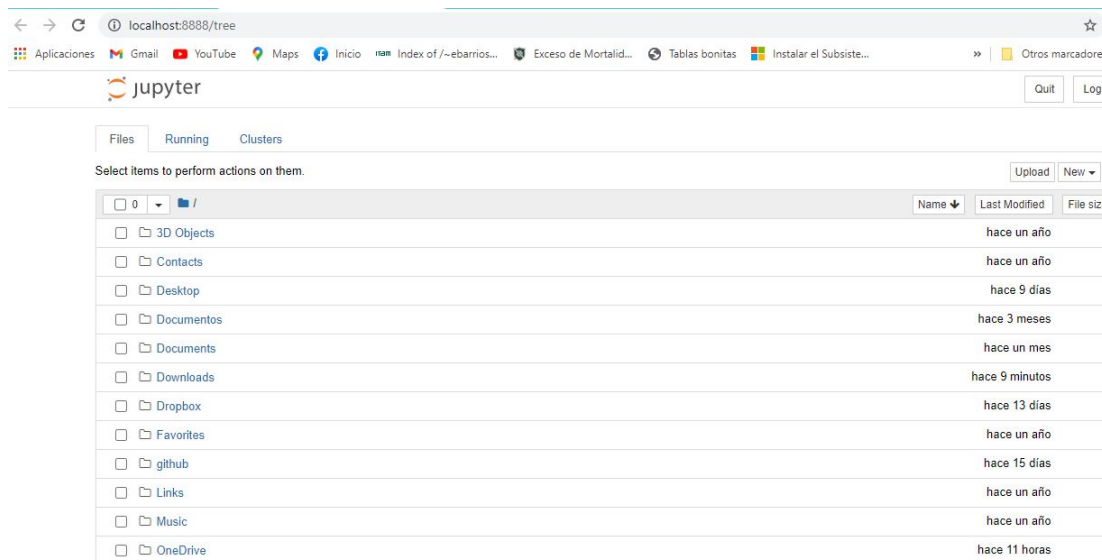


Figura 2.2. Ventana inicial de Jupyter

Vale: No me gustaría agregar en esta parte la instalación de paquetes pero si se tiene que hacer pues ni modo

Para confirmar que la instalación está bien hecha hay que abrir Jupyter, seleccionar New y debe aparecer la opción de Julia 1.6.3.

2.5. Básicos de Julia

Como ya se mencionó, Julia busca ser un lenguaje de programación sencillo e intuitivo. Por lo tanto, su sintáxis es bastante sencilla. La asignación de variables se hace con un signo de igualdad $=$. El ejemplo más sencillo de esto sería $x = 2$ donde denotamos a x el valor de 2. Toda la información sobre la sintaxis que utiliza Julia viene del manual oficial de Julia ([??](#), [Jul](#)).

2.5.1. Operaciones básicas

La siguiente tabla muestra la sintaxis usada para las operaciones básicas en Julia.

Expresión	Nombre	Descripción
$+x$	suma unaria	la operación identidad
$-x$	resta unaria	asigna a los valores sus inversos aditivos
$x + y$	suma binaria	realiza adición
$x - y$	resta binaria	realiza sustracción
$x * y$	multiplicación	realiza multiplicación
x / y	división	realiza divisiones
$x \div y$	división de enteros	x/y truncado a un entero
$x \setminus y$	división inversa	equivalente a dividir y / x
$x \wedge y$	potencia	eleva x a la potencia y
$x \% y$	residuo	equivalente a <code>rem(x,y)</code>

La siguiente tabla muestra los operadores booleanos en Julia.

$!x$	negación
$x \& \& y$	operador de circuito corto <i>and</i>
$x \parallel y$	operador de circuito corto <i>or</i>

Vale:

Se ven horribles los `& &` pero ncon verbatim no funciona

Operaciones básicas en vectores

En Julia, para cada operación binaria existe su correspondiente operación punto (*dot operation* en inglés). Estas funciones están definidas para efectuarse elemento por elemento en vectores y matrices. Para llamarse, basta agregar un punto antes del operador binario. Por ejemplo, `[1 9 9 7] .^ 2` eleva cada uno de los elementos del vector al cuadrado.

También es importante mencionar que Julia maneja los números

imaginarios utilizando el sufijo `im`. Sin embargo, como no utilizaran en esta tesis, se omitirá mayor explicación.

2.5.2. *Strings* (secuencias de caracteres)

Además de números, Julia puede asignar caracteres a variables. Esto se hace utilizando las comillas dobles. Por ejemplo,

```
string = "Esta tesis es genial"
```

```
"Esta tesis es genial"
```

Similar a otros lenguajes de programación, podemos acceder a caracteres específicos de un string utilizando corchetes `[]` y a cadenas seguidas de caracteres usando dos puntos `:`. Continuando con el ejemplo anterior,

Vale: Como pongo esto que se vean los resultados?

```
julia> string[6]
't': ASCII/Unicode U+0074 (category Ll: Letter, lowercase)

julia> string[4:8]
"a tes"
```

Además, Julia también cuenta con la opción de concatenación de múltiples strings. Esto se hace utilizando un asterisco `*` para separar cada uno de los strings. Por ejemplo,

```
julia> grado = "licenciada"
"licenciada"

julia> nexco = "en"
"en"

julia> carrera = "matematicas aplicadas"
"matematicas aplicadas"

julia> espacio = " "
" "

julia> grado*espacio*nexo*espacio*carrera
"licenciada en matematicas aplicadas"
```

2.5.3. Funciones

En Julia, una función es un objeto que asigna una tupla de argumentos a un valor de retorno (`??`, `Jul`). La sintaxis básica para definir funciones en Julia es:

```
julia> function f(x,y)
    x + y
end
f (generic function with 1 method)
```

Figura 2.3. Sintaxis básica de una función en Julia.

Además, puedes agregar la palabra *return* para que la función regrese un valor. Por ejemplo, si quisieramos tener una función a la que le des dos números y te regrese el número mayor, la función sería la siguiente:

```

function numero_mayor(x, y)
    if (x > y)
        return x
    else
        return y
    end
end

```

Para llamar a la función bastaría con escribir `numero_mayor(x, y)` asignando o sustituyendo valores por x y y .

2.5.4. Vectores y Matrices

Definición 1. *Un vector columna de n componentes se define como un conjunto ordenado de n números escritos de la siguiente manera:*

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

En Julia, para definir un vector columna se hace uso de los corchetes cuadrados `[]` y comas. Por ejemplo,

```

julia> A = [1, 9, 9, 7]
4-element Vector{Int64}:
 1
 9
 9
 7

```

```
A = [1, 9, 9, 7]
```

da como resultado un vector de 4 elementos de tipo Int64.

Si quisiera definir un vector renglón, se hace exactamente igual solamente omitiendo el uso de las comas. Sin embargo, es importante señalar que ahora Julia tomó el objeto A como una matriz, no como un vector.

Vale: Me interesa mucho ver lo de los tipos

Definición 2. Una matriz A de $m \times n$ es un arreglo rectangular de mn números dispuestos en m renglones y n columnas.

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1j} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2j} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{i1} & a_{i2} & \dots & a_{ij} & \dots & a_{in} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mj} & \dots & a_{mn} \end{pmatrix}$$

En Julia, hay dos formas de definir matrices. La primera es utilizando los corchetes cuadrados [] para comenzar y terminar la matriz. Las columnas están separadas por espacios y las filas por punto y coma. La segunda opción similar a la primera con la única diferencia de que en lugar de punto y coma se cambia de renglón. Esta opción puede ser un poco tediosa ya que requieren que las columnas estén alineadas. Sin embargo, es una forma más visual de ver las matrices. En la siguiente tabla se pueden ver ambas opciones.

```
julia> A_1 = [1 2 3; 4 5 6]
2×3 Matrix{Int64}:
 1  2  3
 4  5  6

julia> A_2 = [1 2 3
              4 5 6]
2×3 Matrix{Int64}:
 1  2  3
 4  5  6
```

```
A_1 = [1 2 3; 4 5 6]
A_2 = [1 2 3
       4 5 6]
```

De manera análoga con los vectores, para llamar un solo elemento de la matriz se utilizan los corchetes cuadrados. Continuando con el ejemplo anterior, para obtener el número 5 de la matriz A_2, se programaría el código `A_2[2, 2]`.

Vale: No estoy segura de si poner las operaciones básicas de matrices (con sus definiciones, claro) ni la concatenación de matrices

2.5.5. *DataFrames*

Ya que conocemos los arreglos básicos que se pueden manejar en Julia es momento de conocer y manejar los *dataframes*. Un dataframe es una tabla estructurada de dos dimensiones que se usa para tabular distintos tipos de datos. Julia tiene un paquete llamado *DataFrames* que permite trabajar con dataframes que uno crea en Julia o que viene de alguna base de datos externa.

Instalación del paquete

Similar a otros lenguajes de programación, cuando se usa un paquete por primera vez hay que instalarlo. En las ocasiones siguientes donde se quiera volver a usar el paquete basta con llamarlo con la palabra **using**. Para instalar el paquete, hay que poner las siguientes líneas de código.

```
using Pkg
Pkg.add("DataFrames")
using DataFrames
```

```
using Pkg
Pkg.add("DataFrames")
using DataFrames
```

(2.1)

Vale:

Lo malo de esta versión es que no encuentro como enumerarlo

La primera línea de comando le está diciendo a Julia que use el paquete llamado **Pkg**. Este paquete viene por default en la instalación de Julia. La segunda línea le está indicando a Julia que agregue el paquete llamado **DataFrames**. Una vez instalado el paquete de **DataFrames**, hay que hacerle saber a Julia que lo queremos usar en esta ocasión. Para eso, utilizamos la tercera línea del código. En caso de que Julia no marque ningún error, el paquete está listo para utilizarse.

Crear un dataframe

Aunque de manera general los dataframes se utilicen para manejar grandes cantidades de información exportada de otros formatos, es importante saber como se crea y manipula un dataframe desde cero en Julia. Para crear un dataframe, hay que escribir la palabra **DataFrame** y abrir un paréntesis. Después, se escribe el nombre de la primera

columna, un signo de igualdad y los datos que corresponden a esa variable. Se repite lo mismo con la cantidad de columnas que se requieran. Por ejemplo, para hacer un dataframe con las claves únicas y nombres de cinco mujeres el código sería el siguiente:

```
df = DataFrame(id = 1:5,  
nombre = ["Valeria", "Paula", "María José",  
"Sofía", "Mónica"])
```

Es importante nombrar las columnas del dataframe ya que de esta forma basta con escribir `df.col` para referirnos a la columna llamada `col` del dataframe llamado `df`. De la misma manera, si se quisiera agregar una columna nueva basta con asignarle datos a `df.colNueva`. Por ejemplo, si quisiéramos agregar una columna llamada `color` al dataframe del ejemplo anterior, el código sería el siguiente:

```
df.color = ["morado", "azul", "verde", "negro", "rojo"]
```

```
df.color = ["morado", "azul", "verde",  
"negro", "rojo"]
```

Como cualquier otro objeto en Julia, los dataframes tienen diferentes funciones como seleccionar un subgrupo de datos, agregar datos por renglones, modificar y eliminar datos, etc. Sin embargo, eso no es de relevancia para esta tesis por lo que se omitirá.

Importar datos en un dataframe

Como se mencionó anteriormente, los dataframes son utilizados para contener grandes cantidades de información. Usualmente, esta

información no es generada en Julia por lo que hay importarla. Una de las formas más rápidas es usando el paquete CSV. Para instalarlo, hay que seguir los pasos como en la 2.1 cambiando DataFrames por CSV. Una vez instalado el programa, basta utilizar el comando CSV.read y la ruta de la ubicación del archivo para exportar los datos. Por ejemplo,

```
df = CSV.read("C:/Users/Valeria/Documents/ITAM/Tesis/Julia con R/ejemplo_csv.csv", DataFrame)
```

```
julia> df = CSV.read("C:/Users/Valeria/Documents/ITAM/Tesis/Julia con R/ejemplo_csv.csv", DataFrame)
5x6 DataFrame

```

Row	id	nombre	color	deporte	lugar_residencia	estatura
	Int64	String15	String7	String15	String31	Float64
1	1	Valeria	morado	atletismo	Nuevo Leon	1.7
2	2	Paula	azul	hiking	Estado de Mexico	1.75
3	3	Maria Jose	verde	atletismo	Ciudad de Mexico	1.63
4	4	Sofia	negro	funcional	Oaxaca	1.66
5	5	Monica	rojo	baile	Veracruz	1.58

2.5.6. Regresiones

Vale: Buen link de ayuda <https://www.machinelearningplus.com/linear-regression-in-julia/>

Una vez que ya podemos manejar relativamente grandes volúmenes de información en un dataframe en Julia, podemos empezar con la parte verdaderamente estadística. ¿Cuál es el punto de tener una muestra de tamaño significativo si no sabemos analizarla? Una de las maravillas que nos regala la estadística es el uso de regresiones para intentar darle algún sentido o explicación a los datos.

Regresión es un método que permite a los investigadores resumir como predicciones o valores promedio de un resultado varian a través

de variables individuales definidas como predictores. (Gelman et al., 2021)

En pocas palabras, una regresión es una fórmula que intenta explicar como una variable depende otras. Como es de esperarse, Julia tiene un paquete llamado `GLM` que calcula modelos lineales. Para instalarlo y utilizarlo, hay que seguir las instrucciones que vienen en la tabla 2.1 con este paquete. Para ajustar un modelo lineal generalizado, hay que usar la función `glm(formula, data, family, link)` donde

- **formula:** usa los nombres de las columnas del dataframe de datos para referirse a las variables predictoras
- **data:** el dataframe que contenga los predictores de las formula
- **family:** podemos elegir entre `Bernoulli()`, `Binomial()`, `Gamma()`, `Normal()`, `Poisson()` o `NegativeBinomial()`
- **link:** _____

Vale: Viene del manual de GLM que todavia no tengo referencia

Regresión lineal simple

El modelo de regresión lineal más simple es el que tiene un solo predictor:

$$y = a + bx + \epsilon \quad (2.2)$$

Para este modelo, use los datos de (Gelman et al., 2021) sobre el modelo creado por Douglas Hibbs para predecir las elecciones de Estados Unidos basandose solamente en el crecimiento económico. Los datos se ven de la siguiente manera: _____

Vale:
Todavía
no
entiendo
muy bien
esto

Vale: no
entiendo
muy bien
a que se
refiere
esto, creo
que está
fuera del
alcance de
la tesis

Vale:
Como le
hago para

```
julia> elections = CSV.read("C:/Users/Valeria/Documents/ITAM/Tesis/Julia con R/Regression_and_other_stories/ROS-Examples-master/ROS-Examples-master/ElectionsEconomy/data/hibbs.csv", DataFrame)
16x5 DataFrame
Row   year   growth   vote   inc_party_candidate   other_candidate
Int64  Float64  Float64  String15              String15
1     1952    2.4     44.6   Stevenson            Eisenhower
2     1956    2.89    57.76  Eisenhower           Stevenson
3     1960    0.85    49.91  Nixon                 Kennedy
4     1964    4.21    61.34  Johnson               Goldwater
5     1968    3.02    49.6   Humphrey              Nixon
6     1972    3.62    61.79  Nixon                 McGovern
7     1976    1.08    48.95  Ford                  Carter
8     1980   -0.39    44.7   Carter                Reagan
9     1984    3.86    59.17  Reagan                Mondale
10    1988    2.27    53.94  Bush, Sr.             Dukakis
11    1992    0.38    46.55  Bush, Sr.             Clinton
12    1996    1.04    54.74  Clinton               Dole
13    2000    2.36    50.27  Gore                  Bush, Jr.
14    2004    1.72    51.24  Bush, Jr.             Kerry
15    2008    0.1     46.32  McCain                Obama
16    2012    0.95    52.0   Obama                 Romney
```

El modelo que buscamos es que el voto sea resultado del crecimiento económico. Por lo tanto, el modelo programado en Julia se ve de la siguiente manera:

```
julia> elections_lm = lm(@formula(vote ~ growth), elections)
StatsModels.TableRegressionModel{LinearModel{GLM.LmResp{Vector{Float64}}, GLM.DensePredChol{Float64, LinearAlgebra.CholeskyPivoted{Float64, Matrix{Float64}}}}, Matrix{Float64}}

vote ~ 1 + growth

Coefficients:

```

	Coef.	Std. Error	t	Pr(> t)	Lower 95%	Upper 95%
(Intercept)	46.2476	1.62193	28.51	<1e-13	42.769	49.7263
growth	3.06053	0.696274	4.40	0.0006	1.56717	4.55389

Al resultado de la regresión es el modelo $y = 46.3 + 3.1x$.

Regresión lineal múltiple

Vale: Todo esto lo saque de (Gelman et al., 2021)

El modelo de regresión lineal clásico para múltiples predictores es

$$y_i = \beta_1 X_{i1} + \dots + \beta_k X_{ik} + \epsilon_i, \text{ para } i = 1, \dots, n$$

Vale: Solo para que quede claro, es lo mismo que viene en el libro de Gelman

donde los errores ϵ_i son independientes e idénticamente distribuidos de manera normal con media 0 y varianza σ^2 . La representación matricial equivalente es

$$y_i = X_i\beta + \epsilon_i, \text{ para } i = 1, \dots, n$$

donde X es una matriz de $n \times k$ con renglón X_i .

En primer lugar, analicé un modelo que incluye una relación entre dos predictores. Para este modelo, usé los datos de (Gelman et al., 2021) sobre la relación entre los resultados de exámenes de niños (`kid_score`), el coeficiente intelectual IQ de sus madres (`mom_iq`) y si sus madres terminaron o no la preparatoria (`mom_hs`).

La relación que intento modelar es si el coeficiente intelectual y la educación de las madres afectan los resultados de los exámenes de los niños. Por lo tanto, los predictores son las variables en relación con la madre mientras que la respuesta es el desempeño de los niños. El código en Julia se ve de la siguiente manera

```
using DataFrames, GLM, CSV

data_kid = CSV.read("C:/Users/Valeria/Documents/ITAM/Tesis/Julia con R/data_kid.csv")

fm = @formula(kid_score ~ mom_hs + mom_iq + mom_hs*mom_iq)

kidscore_lm = lm(fm, data_kid)
```

Lo cual da como resultado el modelo

Vale: OJO que el coeficiente de `momiq` sale en R como 1.1 y aquí como 0.97

$$kid_score = -11.48 + 51.26 * mom_hs + 0.97 * mom_iq - 0.48 * mom_hs * mom_iq + \epsilon$$

Por lo tanto, para incluir la relación entre dos predictores, basta usar un asterisco entre ellos en la formula de la regresión.

Vale: Me acabo de dar cuenta de eso del paquete CSV. Cambio todo para que ahora sea CSVFiles?? O solo lo menciono??

Como segundo ejemplo, analicé un modelo que tiene una variable categórica para mostrar su aplicación en Julia. A diferencia de otros lenguajes, en Julia no hay que especificar si el predictor es una variable categórica. Sin embargo, si la base de datos tiene valores faltantes (*missing values*) es mejor utilizar el paquete llamado **CSVFiles**. La instalación y uso del paquete es igual que como se muestra en las instrucciones 2.1.

Tomando en cuenta lo anterior, utilicé datos de una encuesta hecha a 1816 personas que buscaba predecir sus ganancias anuales tomando en cuenta aspectos como su altura, peso, sexo, etnicidad, educación, educación de los padres, etc. Para este ejemplo en específico, los predictores son la altura centrada en el promedio (**cHeight**), el sexo (**male**) y la etnicidad (**ethnicity**). La variable respuesta es el peso (**weight**). La razón por la que se centró la altura es para mejorar la comprensión del modelo. El código en Julia para este modelo es el siguiente:

```
using DataFrames, GLM, CSVFiles
```

```
earnings_data = DataFrame(load("C:/Users/Valeria/Documents/ITAM/Tests/earnings_data.csv"))
earnings_data.cHeight = earnings_data.height .- 66
```

```
fm = @formula(weight ~ cHeight + male + ethnicity)
earnings_lm = lm(fm, earnings_data)
```

El resultado es la regresión

Vale: Otra vez, los resultados son levemente diferentes que en el libro

Vale: Se ve bien así en tabla, no?

	Coeficiente	Error Estándar
cHeight	154.33	2.23
male	3.85	0.25
ethnicity: Hispanic	-6.15	3.56
ethnicity: Other	-12.26	5.18
ethnicity: White	-5.19	2.27

Gráfica de resultados

Histograma

Gráfica cuantil-cuantil

Análisis de residuales

Tabla de análisis de varianza

2.5.7. Paquete Distributions

Para aprender como funcionan las distribuciones en Julia. El manual del paquete viene aquí <https://juliastats.org/Distributions.jl/v0.14/index.html>

Capítulo 3

Polynomials

Polynomials es un paquete de Julia que proporciona aritmética básica, integración, diferenciación, evaluación, raíces para polinomios univariados `Polynomials.jl` (`pol`). Para instalar el paquete hay que seguir los comandos 2.1. Esta sección va a explicar como ajustar un polinomio a un conjunto de datos.

3.1. El problema

Supongamos que tenemos un conjunto de datos con solamente dos variables x y y . Buscamos ajustarlos a un polinomio de grado k . Es decir, buscamos ajustar los datos al modelo

$$y = \sum_{j=0}^k \beta_j x^j + \epsilon \quad (3.1)$$

El problema consiste en encontrar los mejores coeficientes que cumplan la ecuación anterior. Para hacerlo, el paquete Polynomials utiliza una implementación del método de Gauss-Newton para resolver un problema de mínimos cuadrados no lineal.

Vale: Ya sé que dice que es para problemas

3.2. Método Gauss-Newton

Vale: Supongo que lo tengo que explicar???

3.3. Ejemplo

Para probar la función `fit` del paquete `Polynomials` elegí utilizar el conjunto de datos llamado *Filip* del Laboratorio de Información Tecnológica perteneciente al Instituto Nacional de Estándares y Tecnología (NIST por sus siglas en inglés). Los datos y el resultado de la estimación de coeficientes están en <https://www.itl.nist.gov/div898/strd/lls/data/LINKS/DATA/Filip.dat>.

El código en Julia para obtener el resultado es el siguiente:

```
using Polynomials
filip = CSV.read("filip_data.csv", DataFrame)
coef_filip = CSV.read("coeficientes_filip.csv", DataFrame)
x = filip.x
y = filip.y
as = coef_filip.coeficiente
p = Polynomials.fit(x,y, 10)
```

Los datos con las 82 observaciones en forma de pares ordenados es `filip_data.csv`, mientras que los datos con los resultados de los coeficientes dados por el NIST es `coeficientes_filip.csv`. Dichos documentos fueron obtenidos de la página mencionada anteriormente. El resultado obtenido en Julia es

Vale:

Lo voy a dejar así aunque se ve pésimo, solo es para que se vean


```
-1467.4896770751313 - 2772.179712060834x  
- 2316.3711834983233x^2 - 1127.973991485157x^3 -  
354.478249903365x^4 - 75.12420525407894x^5 -  
10.875318557918678x^6 - 1.0622150384213702x^7 -  
0.06701911888098762x^8 - 0.0024678109131484544x^9 -  
4.0296254716574806e-5x^10
```

Vale: OJO, no me da el error de la estimación, ni valor t ni nada

Para comparar los resultados con las estimaciones reales de los coeficientes, uso el siguiente código de Julia:

```
as = coef_filip.coeficiente  
q = Polynomial(as)  
maximum(abs(p(x) - q(x)) for x in range(0,1, length=1000))
```

Es decir, creo un polinomio con las estimaciones de los coeficientes de NIST y después hago la diferencia en valor absoluto con la estimación que yo obtuve. El resultado es

```
0.0003723332774825394
```

Por lo tanto, en la peor estimación de β_i , Julia acierta hasta 3 dígitos.

Capítulo 4

R

Porque juntar Julia con R. Usar documentación oficial:
<https://cran.r-project.org/web/packages/JuliaCall/JuliaCall.pdf>.
También en esta sección incluir muchos ejemplos prácticos

4.1. Paquete JuliaCall

Como instalarlo en R

4.1.1. `install_julia`

4.1.2. `julia_setup`

De este no estoy segura, tengo que checar en las cosas de Rodrigo

4.1.3. Núcleos y threads

4.1.4. JuliaObject

4.1.5. julia_assign

4.1.6. julia_eval

4.1.7. julia_package

4.2. Ejemplo

Capítulo 5

Computadoras virtuales

5.1. Amazon

Como hacer una computadora virtual en Amazon que me haga todo esto. El tutorial oficial de como instalarlo en Amazon está aquí: https://d1.awsstatic.com/whitepapers/julia-on-sagemaker.pdf?did=wp_card&trk=wp_card

5.2. Docker

También se podría usar Docker pero te quita espacio en tu computadora y puede crashear como la mía. Para Windows como que no está muy bonito el asunto pero al parecer para Linux sí. También podría ser opción idk

Apéndice A

Extras

No se olviden cambiar toda la información. Los quiero un chingo

Bibliografía

Atom flight manual. <https://flight-manual.atom.io/>. Accessed: 2021-10-06.

Julia 1.6 documentation. <https://docs.julialang.org/en/v1/>. Accessed: 2021-10-07.

Polynomials.jl. <https://juliamath.github.io/Polynomials.jl/stable/>. Accessed: 2021-11-04.

Bezanson, J., J. Nash, and K. Pamnany (2019, Jul). Announcing composable multi-threaded parallelism in julia.

Carrone, F., M. Nicolini, and H. Obst Demaestri (2021). *Data Science in Julia for Hackers*. <https://datasciencejuliahackers.com/> (visited 06-10-2021).

Gelman, A., J. Hill, and A. Vehtari (2021). *Regression and other Stories*. Cambridge University Press.

Una tesis extendida (\overline{tesis}), escrito por Valeria
Aurora Pérez Chávez, se terminó de imprimir
de madrugada,
con mucha cafeína en las venas
y ojeras en la cara.