cv19

Mar 26, 2020

Contents

El propósito de este proyecto es:

Autores:

A python virtual environment is suggested to work with this project. Requirements are listed in the project home directory file: `requirements.txt`.

CHAPTER 1

Science case

1.1 Proposal

Servicio para recabar datos de tablas y hacer una predicción de diferentes parámetros relacionados con la epidemia de Covid19

Los datos a recabar son: - configuracion del experimento, a partir de parámetros ajustados a los datos disponibles - datos demográficos

Resultado: evolución con el tiempo de los siguientes parámetros, con sus respectios errores:

- Cantidad de personas infectadas
- Cantidad de camas necesarias

El experimento es probabilístico y hace uso de algoritmos de simulación

---

El planteo se basa en el gráfico de flujo de individuos entre los diferentes estados en que puede estar respecto de el contagio con el virus:

I: Número de Infectados

C: Número de Confirmados

H: Número de confirmados en tratamiento domiciliario

B: Número de confirmados en cama común en un centro de salud

U: Número de confirmados en una unidad de terapia intensiva

R: Número de confirmados con recuperación registrada

D: Número de confirmados difuntos

Las relaciones entre las variables I, C y (B+H+U) están dadas por los modelos de propagación de infecciones, por ejemplo el modelo SEIR.

En este caso se hace una implementación numérica para analizar variables que no se pueden (?) analizar con el modelo, por ejemplo:

- Usar distribuciones de probabilidad para los diferentes parámetros
- Considerar la incorporación de casos importados
- Calcular la necesidad de camas
- etc (?!)

La transferencia de individuos entre los diferentes estados está determinada por dos cosas:

- La probabilidad de transferencia (f)
- El tiempo de retardo (T)
- Los parámetros propios del individuo

Por ejemplo, la probabilidad de transferencia del estado U al D es la tasa de mortalidad, y ocurre cierto tiempo después de que el paciente ingresa al estado U.

El modelo se puede representar por:

Fig. 1: Representación del modelo como un diagrama de flujo

Modelo donde solo se puede pasar a D desde el estado U	Modelo donde se puede pasar de los estados H, B y U al D

En una simulación, se puede implementar de manera muy simple el modelo considerando (pequeños) incrementos. Por ejemplo, para una variable W que depende de otras variables $V_i,\ i = 1, 2, ..., n$

$$\frac{dV}{dt} = f(V_1, V_2, ..., V_n)$$

y si la función es lineal:

$$\frac{dV}{dt} = a_1 \cdot V_1 + a_2 \cdot V_2 + ... + a_n \cdot V_n$$

se puede integrar de manera numérica, en primera aproximación, como:

$$V(t + dt) = V(t) + (a_1 \cdot V_1 + a_2 \cdot V_2 + ... + a_n \cdot V_n) \cdot dt$$

1.2 References

Tools to analize covid19 data

1.2.1 Resources

Data sources:

- Clean dat from OpenBlender
- Analyzing Coronavirus (Covid-19) Data Using Pandas and Plotly
- Notebook in RMOTR

- MathWorks blog

Pandemic:

- COVID-19 Daily Tracker
- The handbook project
- Article in KDnuggets
- Infection Trajectory in VisualCapitalist
- Visualization (fastcompany)
- Harvard Medical School curriculum

1.2.2 Notebooks

Hasta ahora hay dos notebooks que son mas que nada exploratorios de los datos compilados por John Hopkins CSSE.

world.ipynb: carga datos de la poblacion y area de los paises para construir una tabla con esos dos datos. Luego se usará para normalizar las curvas de contagio de los diferentes países.

load_data.ipynb: Carga los datos de JHU CSSE y analiza las curvas de contagio de algunos países.

1.2.3 DATA

Los datos actualizados son leidos directamente de GitHub, no hace falta bajarlos.

Los datos sobre poblacion y area de los países:

table-1.csv: source: https://en.wikipedia.org/wiki/List_of_countries_and_dependencies_by_area preprocessed with https://wikitable2csv.ggor.de/

table-2.csv: source: https://en.wikipedia.org/wiki/List_of_countries_by_population_(United_Nations) preprocessed with https://wikitable2csv.ggor.de/

world_area.csv: Tabla limpia con las areas

world_population.csv: Tabla limpia con las poblaciones

world.ods: Archivo ODS con las dos tablas (para verificar a ojo)

pop_area.csv: Tabla con las columnas de poblacion y area combinadas

CHAPTER 2

Getting Started

2.1 Getting started

2.1.1 Preparing a virtual environment

```
virtualenv MyVE
source MyVE/bin/activate
pip install -r requirements.txt
```

2.2 API Usage

- instalation through pypy not yet implemented
- make setup.py installer
- from a python script, call import cv19

This project is organized as an API to be used from a python prompt.

Steps:

- Complete the configuration of the experiment
- All the settings of the experimets are parsed from the configuration files using configparser.

2.2.1 Prerequisites

- Put data files on the `dat` directory.
- Complete the names of the data files in the configuration file

Notebooks

Hasta ahora hay dos notebooks que son mas que nada exploratorios de los datos compilados por Johns Hopkins CSSE

world.ipynb: carga datos de la poblacion y area de los paises para construir una tabla con esos dos datos. Luego se usará para normalizar las curvas de contagio de los diferentes países.

load_data.ipynb: Carga los datos de JHU CSSE y analiza las curvas de contagio de algunos países.

DATA

Los datos actualizados son leidos directamente de GitHub, no hace falta bajarlos.

Los datos sobre poblacion y area de los países:

table-1.csv: source: https://en.wikipedia.org/wiki/List_of_countries_and_dependencies_by_area preprocessed with https://wikitable2csv.ggor.de/

table-2.csv: source: https://en.wikipedia.org/wiki/List_of_countries_by_population_(United_Nations) preprocessed with https://wikitable2csv.ggor.de/

world_area.csv: Tabla limpia con las areas

world_population.csv: Tabla limpia con las poblaciones

world.ods: Archivo ODS con las dos tablas (para verificar a ojo)

pop_area.csv: Tabla con las columnas de poblacion y area combinadas

Data is stored in the *dat* directory.

filename	contents
covid_df.csv	Covid-19 world data
pop_area.csv	Population of countries
table-1.csv	
table-2.csv	
table_age.csv	Age distribution table
table_clean.csv	
table_population_cordoba.csv	Age distribution in Cordoba
table_population_world.csv	
world.ods	
world_area.csv	
world_population.csv	

2.2.2 Configuration files

2.2.3 Command line usage

For a simple test, go to src and run:

2.2.4 API usage

To use functionalities, import the *cv19* module:

```
import cv19
```

First, we must parse the configuration parameters from the .ini file.

All parameters with an assigned value must be read with the configparser module. The ConfigParser class is inherited in *cv19.parser*.

Variables can be accessed using the names of the sections and the names of the fields. For example, conf['clinical']['bed_stay'].

```
conf = cv19.parser()
conf.check_file(argv)
conf.read_config_file()
conf.load_filenames()
conf.load_parameters()
```

Finally, the simulation is made with the *cv19.InfectionCurve* class, where the function *cv19.InfectionCurve.compute()* makes the computations.

```
c = cv19.InfectionCurve()
t, I = c.compute(conf.p)
c.plt_IC_n(t, [I], conf.filenames.fname_infected)
```

CHAPTER 3

Developer Documentation

3.1 Model implementation

En esta sección se describe la implementación del modelo numérico para hacer la integración de las variables.

3.1.1 Implementación del diagrama de flujo

El diagrama de flujo de las variables:

Se puede plantear como un grafo bidireccional dirigido, donde los nodos contienen los valores de las variables a medida que se hace evolucionar el sistema y las aristas contienen diferentes valores, por ejemplo las probabilidades de transición o los tiempos de retardo.

Este tipo de objeto se construye instanciando la clase *cv19.Graph*.

Ejemplo de uso:

```
from cv19 import Graph

g = Graph()

for i, inode in enumerate(['A','B','C','D']):
    print(i)
    g.add_node(inode, 0)

nms = ['x', 'y']
g.add_edge('A', 'B', nms, [1, 100])
g.add_edge('A', 'C', nms, [2, 200])
g.add_edge('B', 'D', nms, [3, 300])
g.add_edge('D', 'B', nms, [4, 400])
g.add_edge('D', 'C', nms, [5, 500])
g.add_edge('C', 'C', nms, [6, 600])
```

(continues on next page)

(continued from previous page)

```
g.add_edge('B', 'D', nms, [333, 333])

g.show()
```

3.1.2 Modelo de contagios con grafos

Con la clase *cv19.Graph* se puede crear un grafo y agragar un estado en cada nodo, con valor inicial cero para todos los estados menos para el estado I, que comienza con un valor de por lo menos uno:

```
g = Graph()

for node in ['I','C','R','H','B','U','D',]:
    g.add_node(node, 0)

g.set_node('I', p.N_init)
```

Ahora podemos agregar en las aristas las cosas que necesitamos. Para cada arista vamos a agregar un *lag* y una probabilidad de transición. Para el estado I el *lag* es cero y la probabilidad es la tasa de transmisión. El contador del estado I disminuye cuando hay transferencias del estado I al R.

```
g.add_edge('I', 'I', nms, [p.R,  0])
```

Esto nos va a permitir actualizar el valor de I según:

$$I(t+dt) = I(t) + I(t) * R * dt$$

En esta versión I siempre crece, asi que habría que restarle los casos que se recuperan o fallecen. Es decir, hay que incorporar la trnasición $I \to C$.

$$I(t+dt) = I(t) + I(t) * R * dt - C(t) * P(C|I,t) * dt$$

En general, dada una variable V, su variación se incrementa por todas las variables U que tienen transiciones $U \to V$ y disminuye por todas las variables W que tienen transiciones $I \to W$.

$$V(t+dt,q) = V(t,q) + \sum_i U(t,q) \cdot P(V|U,q) \cdot dt - \sum_i W(t,q) \cdot P(V|W,q) \cdot dt$$

Por ejemplo, si tenemos un grafo como este:

La variacion del nodo V está dada por:

$$\begin{aligned} V(t+dt,q) = V(t,q) &+ U_1(t,q) \cdot P(V|U_1,q) \cdot dt + \\ &+ U_2(t,q) \cdot P(V|U_2,q) \cdot dt + \\ &+ U_3(t,q) \cdot P(V|U_3,q) \cdot dt - \\ &- W_1(t,q) \cdot P(W_1,V,q) \cdot dt - \\ &- W_2(t,q) \cdot P(W_2,V,q) \cdot dt \end{aligned}$$

Aquí por ejemplo $P(V|U_1,q)$ significa probabilidad de que haya una transición del estado U_1 al estado V.

Teniendo en cuenta que:

- los parámetros parseados del archivo .ini están en la lista de parámetros p,

- los tiempos de lag están en unidades de días (si no, hay que pasarlos)
- todos los infectados acusan síntomas al día p.t_incubation

resulta el código para llevar la cuenta de I así:

```
g = Graph()

for node in ['I','C','R','H','B','U','D',]:
    g.add_node(node, 0)

nms = ['prob','lag']
g.set_node('I', p.N_init)
g.add_edge('I', 'I', nms, [p.R,  0])
g.add_edge('I', 'C', nms, [1. p.t_incubation])

t = 0.
time_steps = 0

while t < p.t_max:

    time_steps = time_steps + 1

    t_prev = t
    t = t + p.dt
    ts.append(t)

    prob_II = g.get_edge('I', 'I', 'prob')
    lag_II  = g.get_edge('I', 'I', 'lag')
    prob_IC = g.get_edge('I', 'C', 'prob')
    lag_IC  = g.get_edge('I', 'C', 'lag')

    n_I = I[-1] + I[-lag_II] * prob_II * p.dt - \
          C[-lag_IC] * prob_IC * p.dt
    I.append(n_I)
```

3.1.3 Implementación de un modelo

Para el modelo de la figura

Las transiciones son:

estado	suma	resta
I	I	C
C	I	B, H, U
H	C, B, U	R, B, U
B	C, H, U	H, U
U	C, B, H	D, B, H
R	H	
D	U	

Para este modelo, la actualización de un paso temporal para I es:

$$I(t+dt) = I(t) + I(t-\tau_{II}) \cdot P(I|I) \cdot dt - C(t-\tau_{IC}) \cdot P(C|I) \cdot dt$$

donde τ_{II} es el time lag para la transición $I \to I$, y τ_{IC} es el time lag para la transición $I \to C$. Los valores P son las probabilidades de transición, es decir, $P(C(t)|I(t-\tau_{IC}))$ es la probabilidad de que un individuo pase al estado C en el tiempo t si estaba en el estado I en el tiempo $t - \tau_{IC}$. Por supuesto, $P(I(t)|I(t-\tau_{II}))$ es la probabilidad de permanecer en el estado I durante el tiempo τ_{II}.

Para el contador de "confirmados", C, tenemos que:

$$\begin{aligned} C(t+dt) = C(t) \quad &+ \quad I(t-t_{CC}) \cdot P(C(t)|I(t-\tau_{IC}) \cdot dt - && (3.1)\\ &- \quad C(t-\tau_{CH}) \cdot P(H(t)|C(t-\tau_{CH}) \cdot dt - && (3.2)\\ &- \quad C(t-\tau_{CB}) \cdot P(B(t)|C(t-\tau_{CB}) \cdot dt - && (3.3)\\ &- \quad C(t-\tau_{CU}) \cdot P(U(t)|C(t-\tau_{CU}) \cdot dt && (3.4) \end{aligned}$$

porque sólo se puede llegar al estado C desde el estado I, y se puede pasar desde el estado C a los estados H, B o U con diferentes probabilidades. Los tiempos de retardo desde C son cero, porque se pueden pensar a los estados H, B y U como subestados de C, es decir,

$$C = H + B + U$$

De la misma forma, para las transiciones $B \to H$, $B \to U$, $H \to R$, $H \to B$, $H \to U$, $U \to H$, $U \to B$ y $U \to D$, tenemos las siguientes relaciones:

$$\begin{aligned} B(t+dt) = B(t) \quad &+ \quad C(t-t_{BB}) \cdot P(B(t)|C(t-\tau_{CB}) \cdot dt + && (3.5)\\ &+ \quad H(t-\tau_{HB}) \cdot P(B(t)|C(t-\tau_{HB}) \cdot dt + && (3.6)\\ &+ \quad U(t-\tau_{UB}) \cdot P(B(t)|C(t-\tau_{UB}) \cdot dt - && (3.7)\\ &- \quad U(t-\tau_{UB}) \cdot P(U(t)|B(t-\tau_{UB}) \cdot dt && (3.8) \end{aligned}$$

y análogamente para los otros nodos. Como se puede ver, hay una estructura que persiste en todos los casos, y se puede escribir de manera general. Para un nodo V, la variación en un paso temporal está dada por:

$$\begin{aligned} V(t+dt) = V(t) \quad &+ \quad \sum_{i \in incoming} W_i(t-\tau) \cdot P(V(t)|W_i(t-\tau_{W_i V}) \cdot dt - && (3.9)\\ &- \quad \sum_{i \in outgoing} V(t) \cdot P(W_i(t)|V(t-\tau_{V W_i}) \cdot dt && (3.10) \end{aligned}$$

3.1.4 Versión probabilística

En la version probabilística, los cambios en los valores de los estados dependen de las características de la población en esos estados. Por ejemplo, las transiciones $B \to U$ dependen de la edad de los pacientes. El modelo anterior se puede extender para una versión probabilística de la siguiente forma:

$$V(t+dt, q) = V(t, q) + \sum_i U(t, q) \cdot P(V|U, q) \cdot dt - \sum_i W(t, q) \cdot P(V|W, q) \cdot dt$$

3.1.5 Tabulaciones de las distribuciones de probabilidad

Tasas de mortalidad como función de la edad

Distribución de la edad poblacional

Tiempos de permanencia en UTI como función de la edad

etc. . . .

3.2 CV19 module

class cv19.**Graph**
Bases: object

class Graph This class is used to create and manipulated graphs It makes a heavy use of the node class A graph is made of nodes and edges. This class allows to store a value for each node and different "weights" for each edge. Also, edges are directed.

Exampe of usage: g = Graph()

for i, inode in enumerate(['A','B','C','D']): print(i) g.add_node(inode, 0)

nms = ['x', 'y'] g.add_edge('A', 'B', nms, [1, 100]) g.add_edge('A', 'C', nms, [2, 200]) g.add_edge('B', 'D', nms, [3, 300]) g.add_edge('D', 'B', nms, [4, 400]) g.add_edge('D', 'C', nms, [5, 500]) g.add_edge('C', 'C', nms, [6, 600])

A node can be connected to itself. g.add_edge('B', 'B', nms, [333, 333])

g.show()

vert_dict: dict a dict containing the vertices

num_vertices [int] the number of nodes (or vertices) in a graph

add_node(node, value) get_node() get_node_value() set_node() get_node_ids() get_nodes_to() get_nodes_from()

add_edge(*frm*, *to*, *names=[]*, *values=0*)
warning: does not verify if edge already exists

add_node(*nnode*, *value*)
method: add_node

Adds a node to a graph. The node must have a value.

get_edge(*frm*, *to*, *field*)

get_node(*n*)
method: get_node

n: str

node: a node object

get_node_ids()

get_node_value(*n*)
method: get_node_value

n: str

value: float

get_nodes_from(*nnode*)

get_nodes_to(*nnode*)

node_activation(*nnode*, *key*)

node_upgrade(*nnode*, *key*)

set_node(*n*, *value*)
method: set_node

n: str The ID or name of the node

value: float The value to be assigned to the node

updates the Graph

show_connections()

show_weights()

class cv19.**InfectionCurve**

Bases: object

compute(*p*)

InfectionCurve(self, p): computes the Infection Curve based on a probabilistic model implemented in a simulation

Args: config object from ParseConfig

Raises:

Returns: Time series for the curves of: - Infected - Confirmed - Recovered - Confirmed at home - Confirmed at hospital - Confirmed at ICU - Dead

model_SEIR()

model_SIR(*p*)

plt_IC()

plots the infection curve

Args: ic: time series fplot: filename for the plot

Raises:

Returns: Nothing, just save the plot.

plt_IC_n()

plots the infection curve

Args: ic: time series fplot: filename for the plot

Raises:

Returns: Nothing, just save the plot.

class cv19.**node**(*nnode*, *value*)

Bases: object

class node This class is used to create and manipulated nodes.

add_neighbor(*neighbor*, *names*, *values*)

be_neighbor(*neighbor*, *names*, *values*)

get_connections()

get_id()

get_weight(*neighbor*)

class cv19.**parser**(*defaults=None*, *dict_type=<class 'collections.OrderedDict'>*, *allow_no_value=False*, ***, *delimiters=('=', ':')*, *comment_prefixes=('#', ';')*, *inline_comment_prefixes=None*, *strict=True*, *empty_lines_in_values=True*, *default_section='DEFAULT'*, *interpolation=<object object>*, *converters=<object object>*)

Bases: configparser.ConfigParser

parser class manipulation of parser from ini files

check_file(*sys_args*)
chek_file(args): Parse paramenters for the simulation from a .ini file

Args: filename (str): the file name of the map to be read

Raises:

Returns: readmap: a healpix map, class ?

load_filenames()
load_filenames(self): make filenames based on info in config file

Args: None

Raises:

Returns: list of filenames

load_parameters()
load_parameters(self): load parameters from config file

Args: None

Raises:

Returns: list of parameters as a named tuple

read_config_file()
chek_file(args): Parse paramenters for the simulation from a .ini file

Args: filename (str): the file name of the map to be read

Raises:

Returns: readmap: a healpix map, class ?

class cv19.**table_draw**
Bases: object

add_clean_column(*columnname*, *column*)

load(*filename*)

random_gen(*xinfname*, *xsupname*, *yname*)

method: random_gen Generates a random sample from a given probability density The probability density must be in the table, in the table_draw class.

save_clean_CDF(*filename*)

tabular(*x*, *y*)

test_random_gen(*nran*, *fplot*)

3.3 To do list

3.3.1 Team work

- Organizar códigos. Tratar de armar un solo repositorio, e ir poniendo los códigos "limpios" en *src*

3.3.2 Development

- Complete documentation: it is generated automatically from metacomments, using Sphinx.

  Asserts:

  - What to do is data files are missing
  - Prevent variable overflows and division by zero

3.3.3 TOX

Use TOX to:

- check if package installs correctly with different Python versions and interpreters
- run tests in each of the environments
- act as a frontend to Continuous Integration servers (TRAVIS)

3.4 Testing

Make several tests to reach a good code coverage and verify if results are as expected.

3.4.1 Proposed tests to develop

- check parsing
- read a synthetic data and verify theoretical behaviour
- Check paths and files
- Verify means in statistical distributions
- Try different statistical distributions
- Compare results with other tools

3.4.2 Testing tools and procedures

In order to make testing, we should use any of the following tools:

- pytest
- hypothesis

pytest examples

"pytest will run all files of the form **test_***.py or *_test.py in the current directory and its subdirectories." So, simply go to the tst directory, and run pytest.

In the environment:

In the code (example from pytest documentation):

How to run test:

From the CLI, write:

** coverage **

Desde el entorno, instalar los paquetes coverage y pytest-cov:

Para calcular la cobertura de código, correr:

Se puede integrar el pytest con el codecov:

3.5 Documenting the project

Sphinx: automatic generation of documents from docstrings

This documentation has been generated using Sphinx.

In order to generate HTML docs locally, go to `doc/` and run:

```
(python-env) make html
```

Editting docs

Just edit .rst files.

Generate the documentation on line in readthedocs

1. log in into readthedocs account
2. go to project
3. build

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index