Modeling and Simulation in Python

Chapter 5

Lab Author @ Joseph Simone

```
# Configure Jupyter so figures appear in the notebook
%matplotlib inline

# Configure Jupyter to display the assigned value after an assignment
%config InteractiveShell.ast_node_interactivity='last_expr_or_assign'

# import functions from the modsim.py module
from modsim import *
```

Reading data

Pandas is a library that provides tools for reading and processing data. `read_html` reads a web page from a file or the Internet and creates one `DataFrame` for each table on the page.

```
from pandas import read_html
```

The data directory contains a downloaded copy of https://en.wikipedia.org/wiki/World_population_estimates

The arguments of `read_html` specify the file to read and how to interpret the tables in the file. The result, `tables`, is a sequence of `DataFrame` objects; `len(tables)` reports the length of the sequence.

```
filename = 'data/World_population_estimates.html'
tables = read_html(filename, header=0, index_col=0, decimal='M')
len(tables)
```

```
6
```

We can select the `DataFrame` we want using the bracket operator. The tables are numbered from 0, so `tables[2]` is actually the third table on the page.

`head` selects the header and the first five rows.

```
table2 = tables[2]
table2.head()
```

United States Census Bureau (2017)[28]

Population Reference Bureau (1973–2016)[15]

United Nations Department of Economic and Social Affairs (2015)[16]

Maddison (2008)[17]

HYDE (2007)[24]

Tanton (1994)[18]

Biraben (1980)[19]

McEvedy & Jones (1978)[20]

Thomlinson (1975)[21]

Durand (1974)[22]

Clark (1967)[23]

Year

1950

2557628654

2.516000e+09

2.525149e+09

2.544000e+09

2.527960e+09

2.400000e+09

2.527000e+09

2.500000e+09

2.400000e+09

NaN

2.486000e+09

1951

2594939877

NaN

2.572851e+09

2.571663e+09

NaN

NaN

NaN

NaN

NaN

NaN

NaN

1952

2636772306

NaN

2.619292e+09

2.617949e+09

NaN

NaN

NaN

NaN

NaN

NaN

NaN

1953

2682053389

NaN

2.665865e+09

2.665959e+09

NaN

NaN

NaN

NaN

NaN

NaN

NaN

1954

2730228104

NaN

2.713172e+09

2.716927e+09

NaN

NaN

NaN

NaN

NaN

NaN

NaN

`tail` selects the last five rows.

```
table2.tail()
```

United States Census Bureau (2017)[28]

Population Reference Bureau (1973–2016)[15]

United Nations Department of Economic and Social Affairs (2015)[16]

Maddison (2008)[17]

HYDE (2007)[24]

Tanton (1994)[18]

Biraben (1980)[19]

McEvedy & Jones (1978)[20]

Thomlinson (1975)[21]

Durand (1974)[22]

Clark (1967)[23]

Year

2012

7013871313

7.057075e+09

7.080072e+09

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

2013

7092128094

7.136796e+09

7.162119e+09

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

2014

7169968185

7.238184e+09

7.243784e+09

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

2015

7247892788

7.336435e+09

7.349472e+09

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

2016

7325996709

7.418152e+09

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

Long column names are awkard to work with, but we can replace them with abbreviated names.

```python
table2.columns = ['census', 'prb', 'un', 'maddison',
                  'hyde', 'tanton', 'biraben', 'mj',
                  'thomlinson', 'durand', 'clark']
```

Here's what the DataFrame looks like now.

```python
table2.head()
```

census

prb

un

maddison

hyde

tanton

biraben

mj

thomlinson

durand

clark

Year

1950

2557628654

2.516000e+09

2.525149e+09

2.544000e+09

2.527960e+09

2.400000e+09

2.527000e+09

2.500000e+09

2.400000e+09

NaN

2.486000e+09

1951

2594939877

NaN

2.572851e+09

2.571663e+09

NaN

NaN

NaN

NaN

NaN

NaN

NaN

1952

2636772306

NaN

2.619292e+09

2.617949e+09

NaN

NaN

NaN

NaN

NaN

NaN

NaN

1953

2682053389

NaN

2.665865e+09

2.665959e+09

NaN

NaN

NaN

NaN

NaN

NaN

NaN

1954

2730228104

NaN

2.713172e+09

2.716927e+09

NaN

NaN

NaN

NaN

NaN

NaN

NaN

The first column, which is labeled `Year`, is special. It is the **index** for this `DataFrame`, which means it contains the labels for the rows.

Some of the values use scientific notation; for example, `2.544000e+09` is shorthand for $2.544 \cdot 10^9$ or 2.544 billion.

`NaN` is a special value that indicates missing data.

Series

We can use dot notation to select a column from a `DataFrame`. The result is a `Series`, which is like a `DataFrame` with a single column.

```
census = table2.census
census.head()
```

```
Year
1950    2557628654
1951    2594939877
1952    2636772306
1953    2682053389
1954    2730228104
Name: census, dtype: int64
```

```
census.tail()
```

```
Year
2012    7013871313
2013    7092128094
2014    7169968185
2015    7247892788
2016    7325996709
Name: census, dtype: int64
```

Like a `DataFrame`, a `Series` contains an index, which labels the rows.

`1e9` is scientific notation for $1 \cdot 10^9$ or 1 billion.

From here on, we will work in units of billions.

```
un = table2.un / 1e9
un.head()
```

```
Year
1950    2.525149
1951    2.572851
1952    2.619292
1953    2.665865
1954    2.713172
Name: un, dtype: float64
```

```
census = table2.census / 1e9
census.head()
```

```
Year
1950    2.557629
1951    2.594940
1952    2.636772
1953    2.682053
1954    2.730228
Name: census, dtype: float64
```

Here's what these estimates look like.

```
plot(census, ':', label='US Census')
plot(un, '--', label='UN DESA')

decorate(xlabel='Year',
         ylabel='World population (billion)')

savefig('figs/chap05-fig01.pdf')
```

```
Saving figure to file figs/chap05-fig01.pdf
```

The following expression computes the elementwise differences between the two series, then divides through by the UN value to produce relative errors, then finds the largest element.

So the largest relative error between the estimates is about 1.3%.

```
max(abs(census - un) / un) * 100
```

```
1.3821293828998855
```

Exercise: Break down that expression into smaller steps and display the intermediate results, to make sure you understand how it works.

1. Compute the elementwise differences, `census - un`
2. Compute the absolute differences, `abs(census - un)`
3. Compute the relative differences, `abs(census - un) / un`
4. Compute the percent differences, `abs(census - un) / un * 100`

```
elementwise_value = max(census - un)
```

```
0.0324796540000003
```

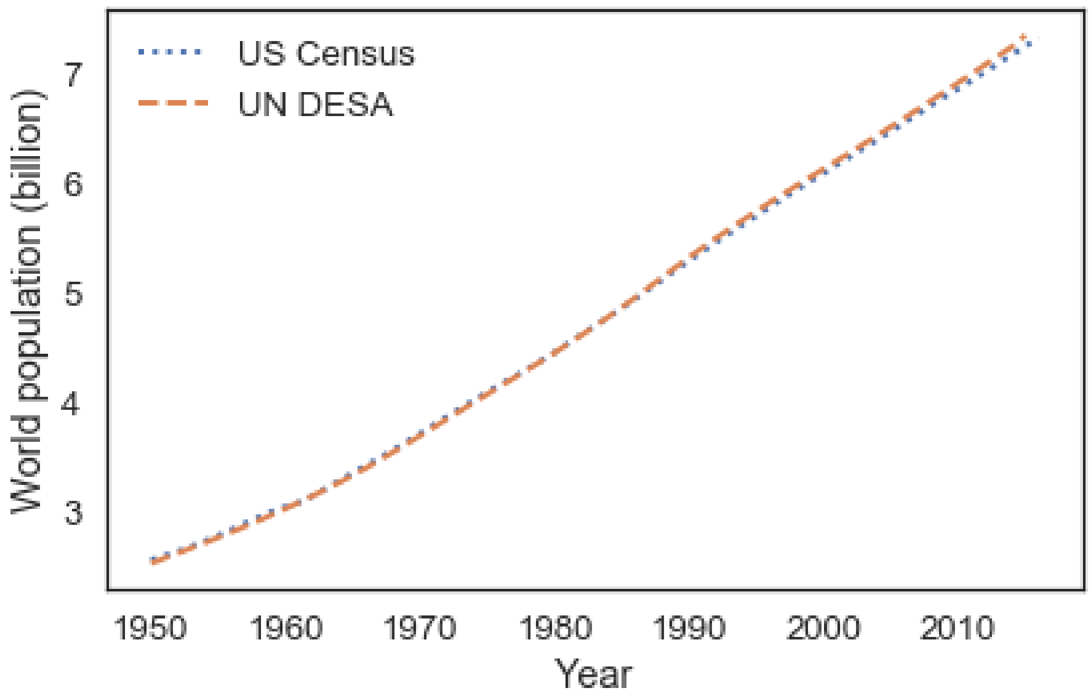

Figure 1: png

```
absolute_value = max(abs(census - un))
```

```
0.10157921199999986
```

```
relative_differences_value = max(abs(census - un) / un )
```

```
0.013821293828998854
```

```
percent_difference = max(abs(census - un)/un*100)
```

```
1.3821293828998855
```

`max` and `abs` are built-in functions provided by Python, but NumPy also provides version that are a little more general. When you import `modsim`, you get the NumPy versions of these functions.

Constant growth

We can select a value from a `Series` using bracket notation. Here's the first element:

```
census[1950]
```

```
2.557628654
```

And the last value.

```
census[2016]
```

```
7.325996709
```

But rather than "hard code" those dates, we can get the first and last labels from the `Series`:

```
t_0 = get_first_label(census)
```

```
1950
```

```
t_end = get_last_label(census)
```

```
2016
```

```
elapsed_time = t_end - t_0
```

```
66
```

And we can get the first and last values:

```
p_0 = get_first_value(census)
```

```
2.557628654
```

```
p_end = get_last_value(census)
```

```
7.325996709
```

Then we can compute the average annual growth in billions of people per year.

```
total_growth = p_end - p_0
```

```
4.768368055
```

```
annual_growth = total_growth / elapsed_time
```

```
0.07224800083333333
```

TimeSeries

Now let's create a `TimeSeries` to contain values generated by a linear growth model.

```
results = TimeSeries()
```

values

Initially the `TimeSeries` is empty, but we can initialize it so the starting value, in 1950, is the 1950 population estimated by the US Census.

```
results[t_0] = census[t_0]
results
```

values

1950

2.557629

After that, the population in the model grows by a constant amount each year.

```
for t in linrange(t_0, t_end):
    results[t+1] = results[t] + annual_growth
```

Here's what the results looks like, compared to the actual data.

```
plot(census, ':', label='US Census')
plot(un, '--', label='UN DESA')
plot(results, color='gray', label='model')

decorate(xlabel='Year',
         ylabel='World population (billion)',
         title='Constant growth')

savefig('figs/chap05-fig02.pdf')
```

```
Saving figure to file figs/chap05-fig02.pdf
```

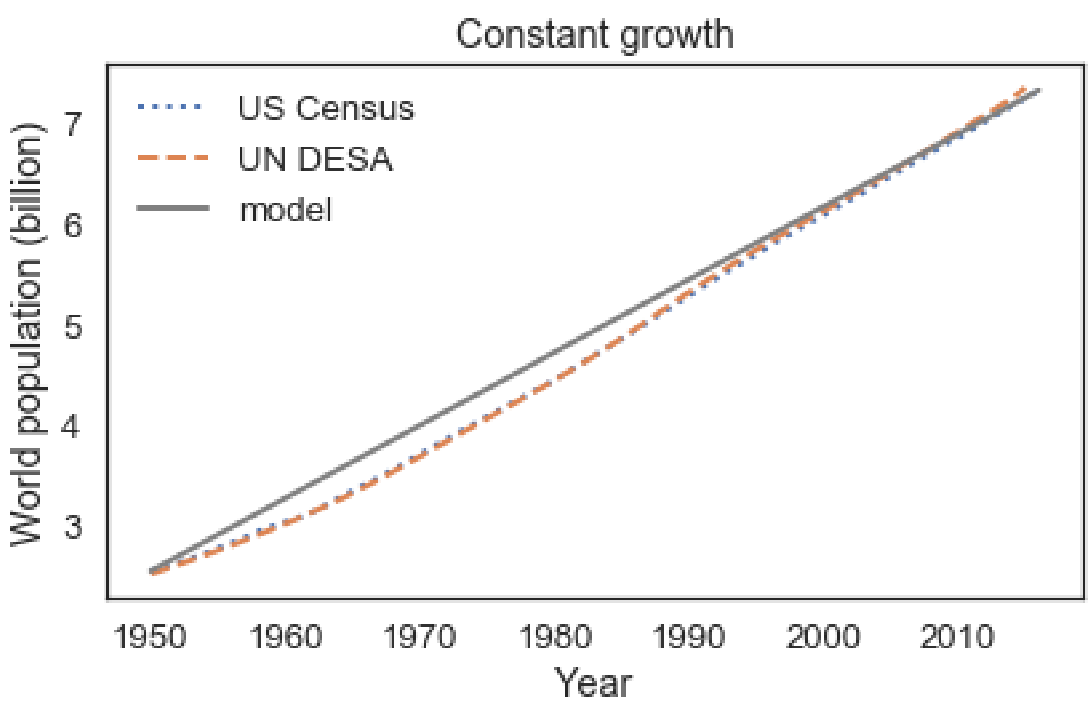

Figure 2: png

The model fits the data pretty well after 1990, but not so well before.

Exercises

Optional Exercise: Try fitting the model using data from 1970 to the present, and see if that does a better job.

Hint:

1. Copy the code from above and make a few changes. Test your code after each small change.
2. Make sure your `TimeSeries` starts in 1950, even though the estimated annual growth is based on later data.
3. You might want to add a constant to the starting value to match the data better.

```
p1 = census[1970]
```

```
3.712697742
```

```
p_end1 = get_last_value(census)
```

```
7.325996709
```

```
year_range = census.loc[1960:1970]
```

```
Year
1960    3.043002
1961    3.083967
1962    3.140093
1963    3.209828
1964    3.281201
1965    3.350426
1966    3.420678
1967    3.490334
1968    3.562314
1969    3.637159
1970    3.712698
Name: census, dtype: float64
```

```
t_0 = get_last_label(year_range)
```

```
1970
```

```
t_end = get_last_label(census)
```

```
2016
```

```
elapsed_time = t_end - t_0
```

```
46
```

```python
total_growth = p_end1 - p1
```

```
3.613298967
```

```python
annual_growth = total_growth / elapsed_time
```

```
0.07854997754347826
```

```python
result = TimeSeries()
```

values

```python
result[t_0] = census[t_0]
result
```

values

1970

3.712698

```python
for n in linrange(t_0, t_end):
    result[n+1] = result[n] + annual_growth
```

```python
plot(census, ':', label='US Census')
plot(un, '--', label='UN DESA')
plot(result, color='gray', label='model')

decorate(xlabel='Year',
         ylabel='World population (billion)',
         title='Constant growth')

savefig('figs/chap05-fig02.pdf')
```

```
Saving figure to file figs/chap05-fig02.pdf
```

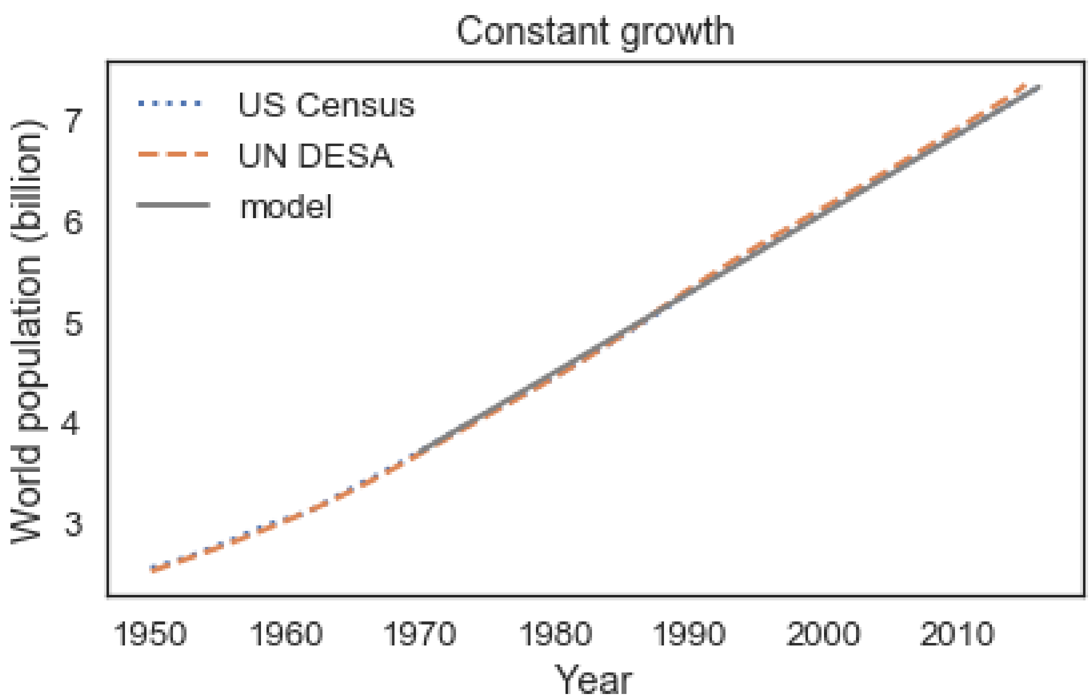
Constant growth
US Census
UN DESA
model
World population (billion)
3
4
5
6
7
1950
1960
1970
1980
1990
2000
2010
Year

Figure 3: png