Modeling and Simulation in Python

Chapter 8

Lab Author @ Joseph Simone

```python
# 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 *
```

Functions from the previous chapter

```python
def plot_results(census, un, timeseries, title):
    """Plot the estimates and the model.

    census: TimeSeries of population estimates
    un: TimeSeries of population estimates
    timeseries: TimeSeries of simulation results
    title: string
    """
    plot(census, ':', label='US Census')
    plot(un, '--', label='UN DESA')
    plot(timeseries, color='gray', label='model')

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

```python
def run_simulation(system, update_func):
    """Simulate the system using any update function.

    system: System object
    update_func: function that computes the population next year

    returns: TimeSeries
    """
    results = TimeSeries()
    results[system.t_0] = system.p_0

    for t in linrange(system.t_0, system.t_end):
        results[t+1] = update_func(results[t], t, system)

    return results
```

Reading the data

```
def read_table2(filename = 'data/World_population_estimates.html'):
    tables = pd.read_html(filename, header=0, index_col=0, decimal='M')
    table2 = tables[2]
    table2.columns = ['census', 'prb', 'un', 'maddison',
                      'hyde', 'tanton', 'biraben', 'mj',
                      'thomlinson', 'durand', 'clark']
    return table2
```

```
#table2 = read_table2()
#table2.to_csv('data/World_population_estimates2.csv')
```

```
table2 = pd.read_csv('data/World_population_estimates2.csv')
table2.index = table2.Year
table2.head()
```

Year

census

prb

un

maddison

hyde

tanton

biraben

mj

thomlinson

durand

clark

Year

1950

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

1951

2594939877

NaN

2.572851e+09

2.571663e+09

NaN

NaN

NaN

NaN

NaN

NaN

NaN

1952

1952

2636772306

NaN

2.619292e+09

2.617949e+09

NaN

NaN

NaN

NaN

NaN

NaN

NaN

1953

1953

2682053389

NaN

2.665865e+09

2.665959e+09

NaN

NaN

NaN

NaN

NaN

NaN

NaN

1954

1954

2730228104

NaN

2.713172e+09

2.716927e+09

NaN

NaN

NaN

NaN

NaN

NaN

NaN

```
un = table2.un / 1e9
census = table2.census / 1e9
plot(census, ':', label='US Census')
plot(un, '--', label='UN DESA')

decorate(xlabel='Year',
             ylabel='World population (billion)',
             title='Estimated world population')
```

Running the quadratic model

Here's the update function for the quadratic growth model with parameters `alpha` and `beta`.

```
def update_func_quad(pop, t, system):
    """Update population based on a quadratic model.

    pop: current population in billions
    t: what year it is
    system: system object with model parameters
    """
    net_growth = system.alpha * pop + system.beta * pop**2
    return pop + net_growth
```

Extract the starting time and population.

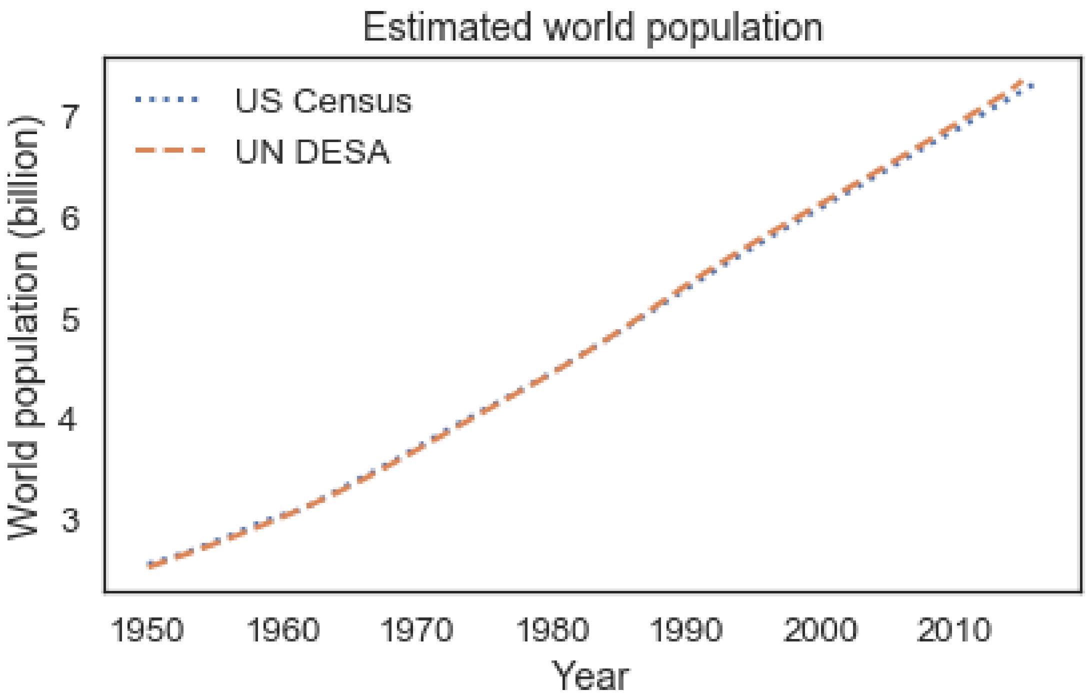

Figure 1: png

```
t_0 = get_first_label(census)
t_end = get_last_label(census)
p_0 = get_first_value(census)
```

```
2.557628654
```

Initialize the system object.

```
system = System(t_0=t_0,
                t_end=t_end,
                p_0=p_0,
                alpha=0.025,
                beta=-0.0018)
```

values

t_0

1950.000000

t_end

2016.000000

p_0

2.557629

alpha

0.025000

beta

-0.001800

Run the model and plot results.

```
results = run_simulation(system, update_func_quad)
plot_results(census, un, results, 'Quadratic model')
```

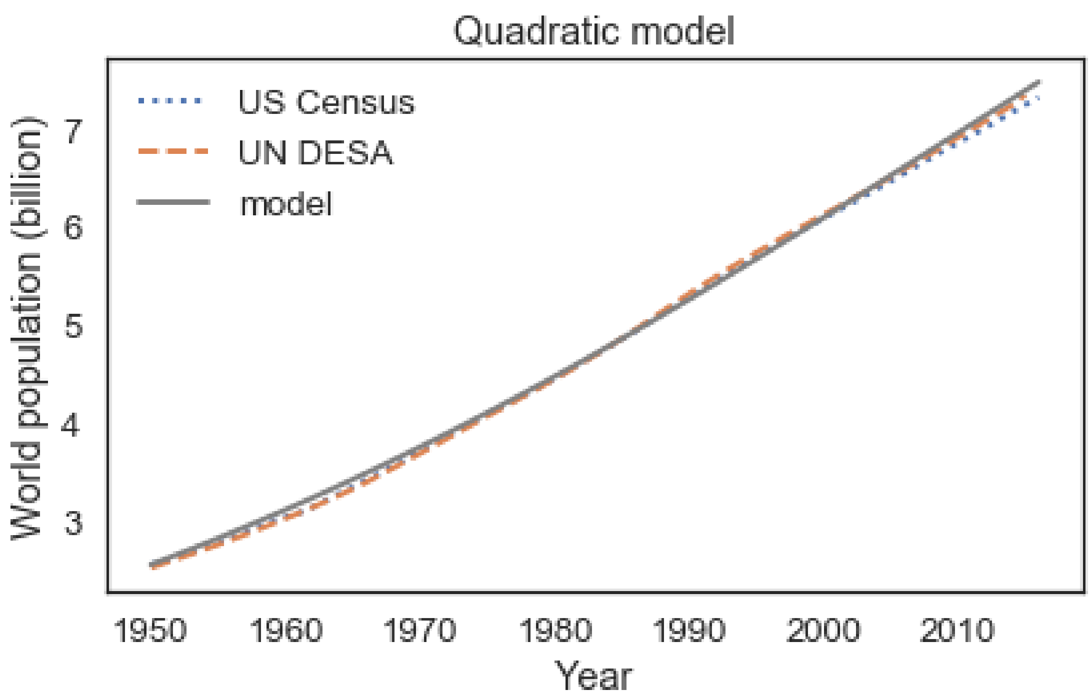

Figure 2: png

Generating projections

To generate projections, all we have to do is change `t_end`

```
system.t_end = 2250
results = run_simulation(system, update_func_quad)
plot_results(census, un, results, 'World population projection')
savefig('figs/chap08-fig01.pdf')
```

```
Saving figure to file figs/chap08-fig01.pdf
```

The population in the model converges on the equilibrium population, `-alpha/beta`

```
results[system.t_end]
```

```
13.856665141368708
```

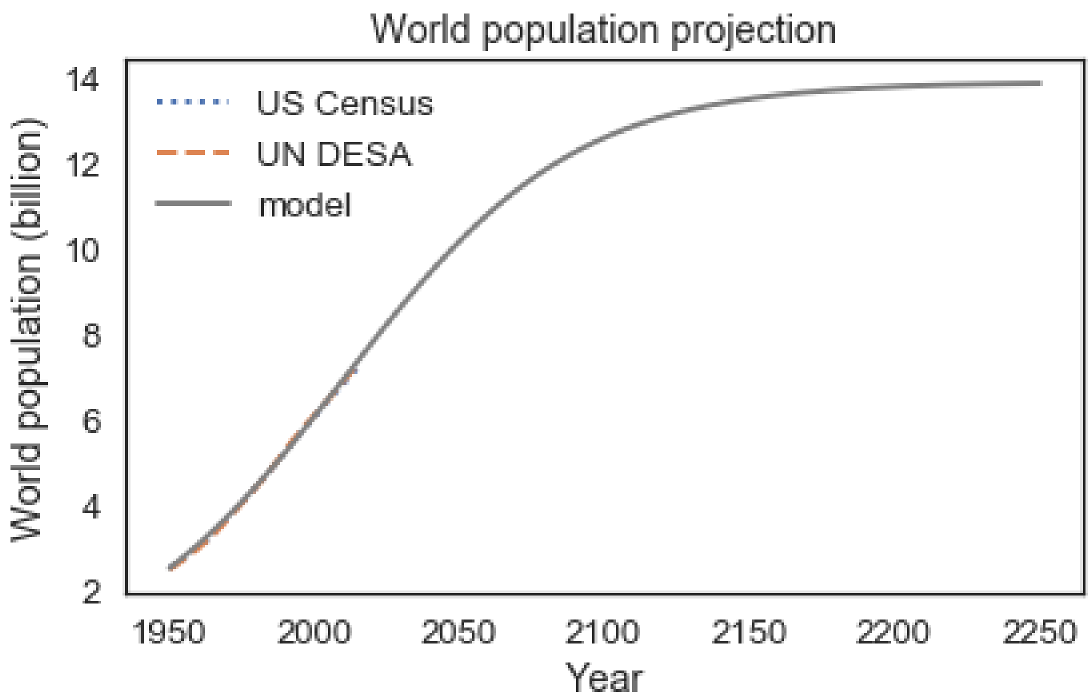

Figure 3: png

```
-system.alpha / system.beta
```

```
13.88888888888889
```

Exercise: What happens if we start with an initial population above the carrying capacity, like 20 billion? Run the model with initial populations between 1 and 20 billion, and plot the results on the same axes.

```
t_0 = get_first_label(census)
t_end = get_last_label(census)
p_0 = 1
p_end = 20
system = System(t_0=t_0,
                t_end=t_end,
                p_0=p_0,
                p_end = p_end,
                alpha=0.025,
                beta=-0.0018)
```

values

t_0

1950.0000

t_end

2016.0000

p_0

1.0000

p_end

20.0000

alpha

0.0250

beta

-0.0018

```
results = run_simulation(system, update_func_quad)
plot_results(census, un, results, 'Quadratic model')
```

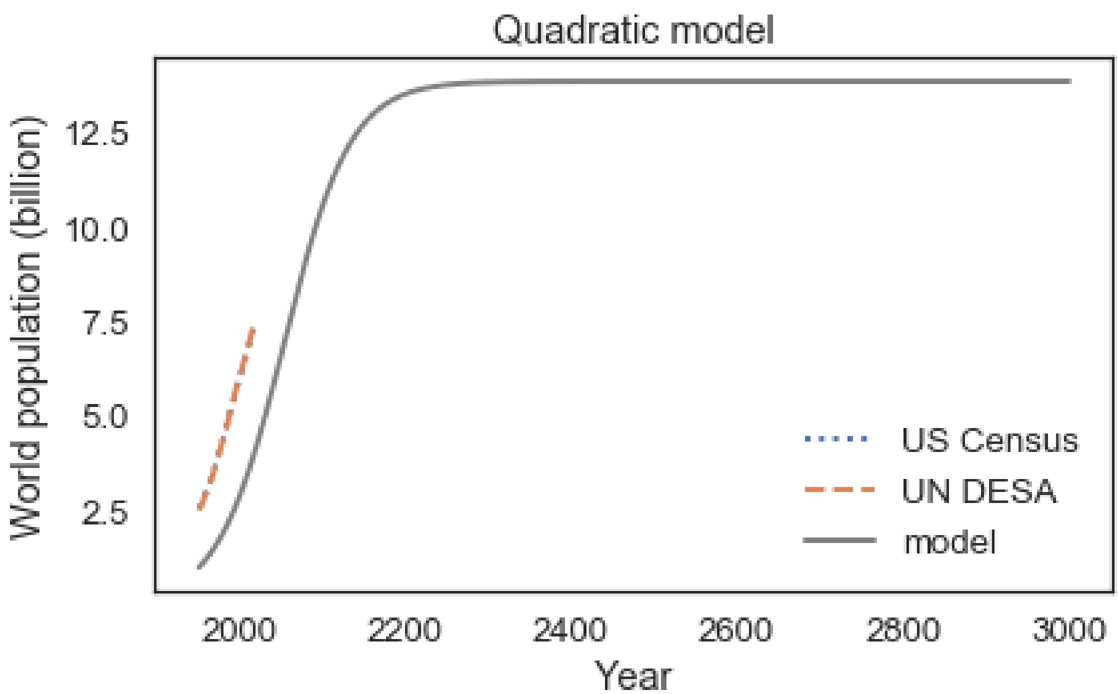

Figure 4: png

The ModSim Package assumes that the population growth has been limited due to a number of factors. The model is based on the assumption that population growth is limited by the access to resources. Meaning as the population appraches the carrying capacity, resources become less available.

Comparing projections

We can compare the projection from our model with projections produced by people who know what they are doing.

```
def read_table3(filename = 'data/World_population_estimates.html'):
    tables = pd.read_html(filename, header=0, index_col=0, decimal='M')
    table3 = tables[3]
    table3.columns = ['census', 'prb', 'un']
    return table3
```

```python
#table3 = read_table3()
#table3.to_csv('data/World_population_estimates3.csv')
```

```python
table3 = pd.read_csv('data/World_population_estimates3.csv')
table3.index = table3.Year
table3.head()
```

Year

census

prb

un

Year

2016

2016

7.334772e+09

NaN

7.432663e+09

2017

2017

7.412779e+09

NaN

NaN

2018

2018

7.490428e+09

NaN

NaN

2019

2019

7.567403e+09

NaN

NaN

2020

2020

7.643402e+09

NaN

7.758157e+09

`NaN` is a special value that represents missing data, in this case because some agencies did not publish projections for some years.

This function plots projections from the UN DESA and U.S. Census. It uses `dropna` to remove the `NaN` values from each series before plotting it.

```
def plot_projections(table):
    """Plot world population projections.

    table: DataFrame with columns 'un' and 'census'
    """
    census_proj = table.census / 1e9
    un_proj = table.un / 1e9

    plot(census_proj.dropna(), ':', color='C0', label='US Census')
    plot(un_proj.dropna(), '--', color='C1', label='UN DESA')
```

Run the model until 2100, which is as far as the other projections go.

```
t_0 = get_first_label(census)
t_end = get_last_label(census)
p_0 = get_first_value(census)
```

```
2.557628654
```

```
system = System(t_0=t_0,
                t_end=2100,
                p_0=p_0,
                alpha=0.025,
                beta=-0.0018)
```

values

t_0

1950.000000

t_end

2100.000000

p_0

2.557629

alpha

0.025000

beta

-0.001800

```
results = run_simulation(system, update_func_quad)

plt.axvspan(1950, 2016, color='C0', alpha=0.05)
plot_results(census, un, results, 'World population projections')
plot_projections(table3)
savefig('figs/chap08-fig02.pdf')
```

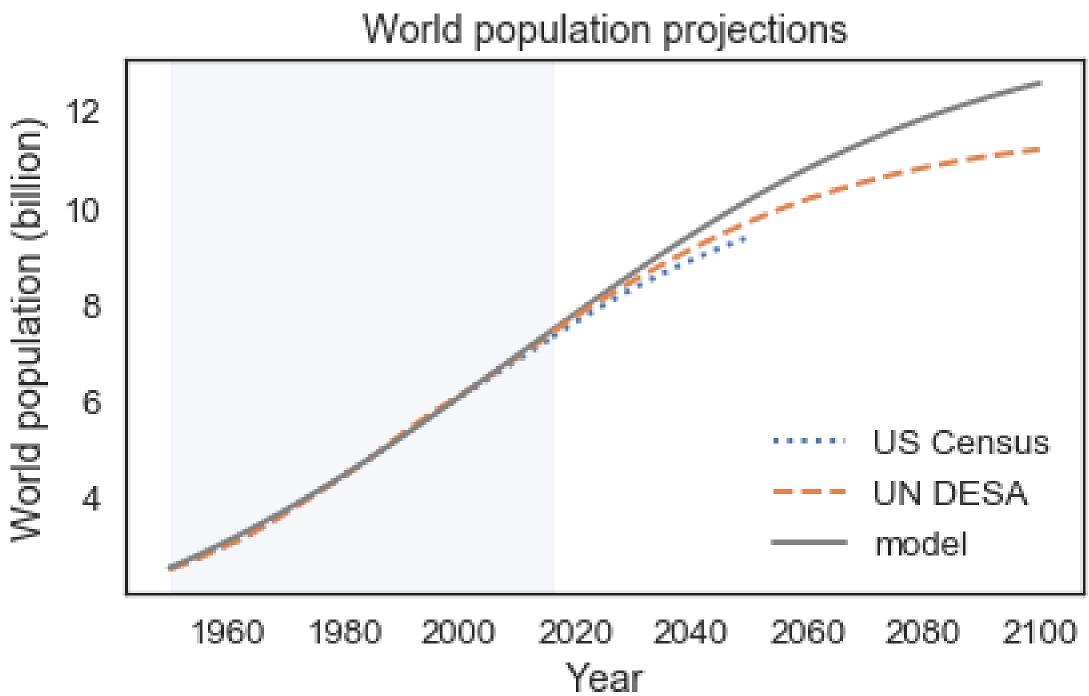

Figure 5: png

```
Saving figure to file figs/chap08-fig02.pdf
```

People who know what they are doing expect the growth rate to decline more sharply than our model projects.

Exercises

Exercise: The net growth rate of world population has been declining for several decades. That observation suggests one more way to generate projections, by extrapolating observed changes in growth rate.

The `modsim` library provides a function, `compute_rel_diff`, that computes relative differences of the elements in a sequence.

Here's how we can use it to compute the relative differences in the `census` and `un` estimates:

```
alpha_census = compute_rel_diff(census)
plot(alpha_census, label='US Census')

alpha_un = compute_rel_diff(un)
plot(alpha_un, label='UN DESA')

decorate(xlabel='Year', label='Net growth rate')
```

Other than a bump around 1990, net growth rate has been declining roughly linearly since 1965. As an exercise, you can use this data to make a projection of world population until 2100.

1. Define a function, `alpha_func`, that takes `t` as a parameter and returns an estimate of the net growth rate at time `t`, based on a linear function `alpha = intercept + slope * t`. Choose values of `slope` and `intercept` to fit the observed net growth rates since 1965.

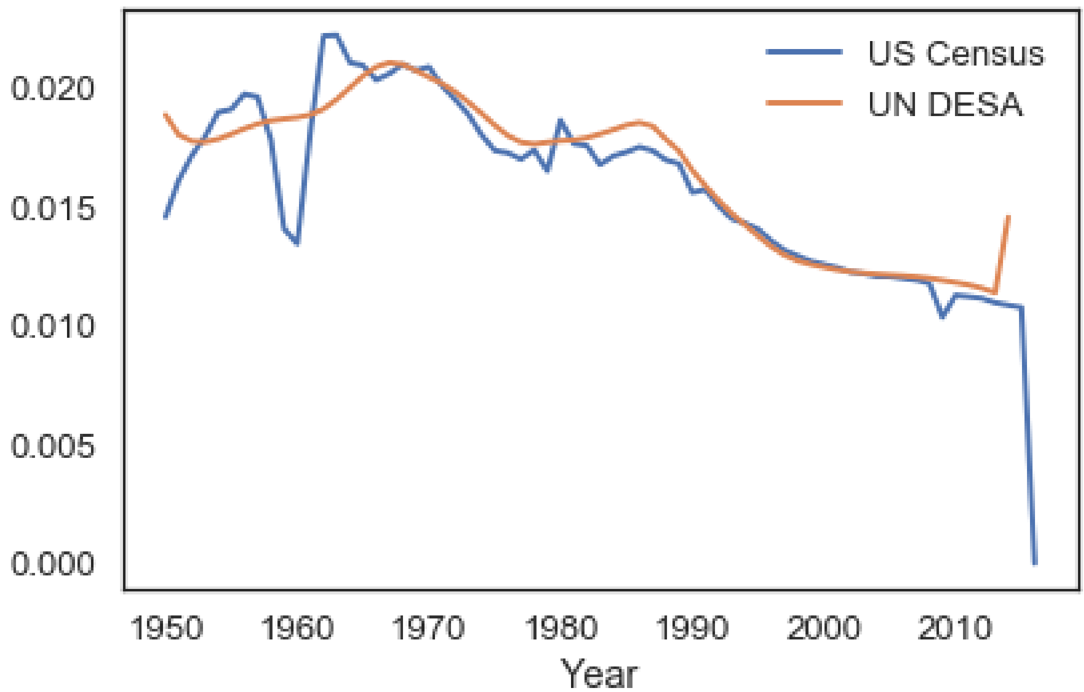

Figure 6: png

2. Call your function with a range of `ts` from 1960 to 2020 and plot the results.
3. Create a `System` object that includes `alpha_func` as a system variable.
4. Define an update function that uses `alpha_func` to compute the net growth rate at the given time `t`.
5. Test your update function with `t_0 = 1960` and `p_0 = census[t_0]`.
6. Run a simulation from 1960 to 2100 with your update function, and plot the results.
7. Compare your projections with those from the US Census and UN.

```
def alpha_func(t):
    intercept = 0.02
    slope = -0.00022
    y = 1970
    return intercept + slope * (t-y)
```

```
ts = linrange(1960, 2021)
```

```
array([1960, 1961, 1962, 1963, 1964, 1965, 1966, 1967, 1968, 1969, 1970,
       1971, 1972, 1973, 1974, 1975, 1976, 1977, 1978, 1979, 1980, 1981,
       1982, 1983, 1984, 1985, 1986, 1987, 1988, 1989, 1990, 1991, 1992,
       1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003,
       2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014,
       2015, 2016, 2017, 2018, 2019, 2020], dtype=int32)
```

```
test_model = TimeSeries(alpha_func(ts), ts)
plot(test_model, color='gray', label='alpha_test_model')
plot(alpha_census)
plot(alpha_un)

decorate(xlabel='Year', ylabel='Net growth rate')
```

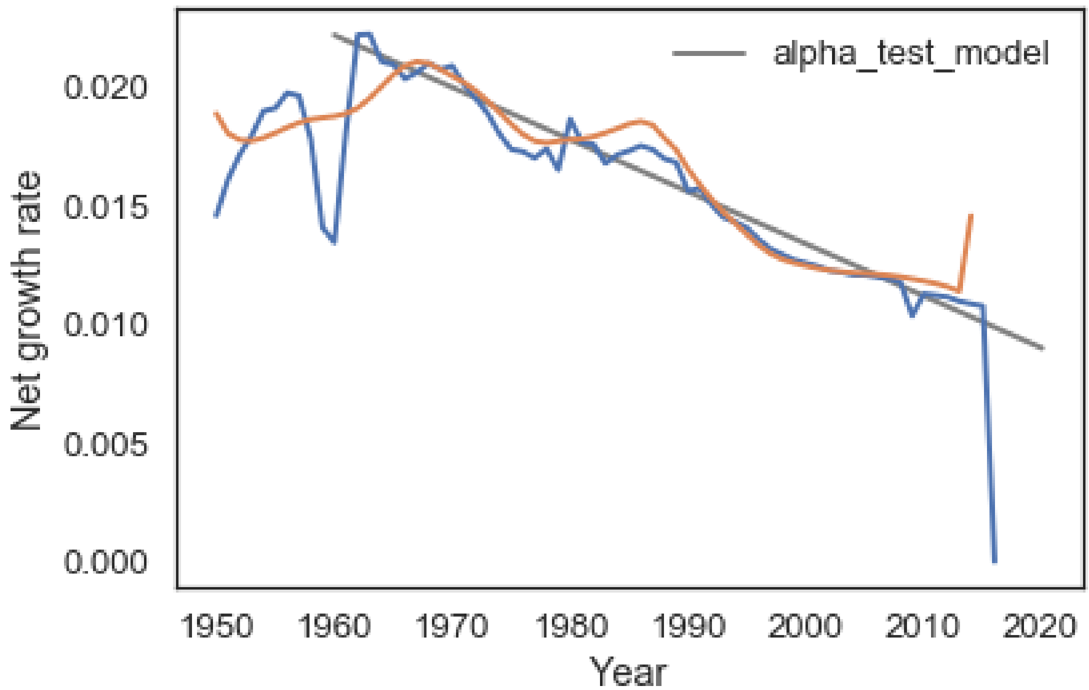

Figure 7: png

```
t_0 = 1960
p_0 = census[t_0]
```

```
3.043001508
```

```
system = System(t_0=t_0,
                t_end=2100,
                p_0=p_0,
                alpha_func=alpha_func)
```

values

t_0

1960

t_end

2100

p_0

3.043

alpha_func

<function alpha_func at 0x000001CD4601F5E8>

```
def update_func_alpha(pop, t, system):
    net_growth = system.alpha_func(t) * pop
    return pop + net_growth
```

```
update_func_alpha(p_0, t_0, system)
```

```
3.1105561414776
```

```
results = run_simulation(system, update_func_alpha);
```

```
plot_results(census, un, results, 'World population projections')
plot_projections(table3)
```

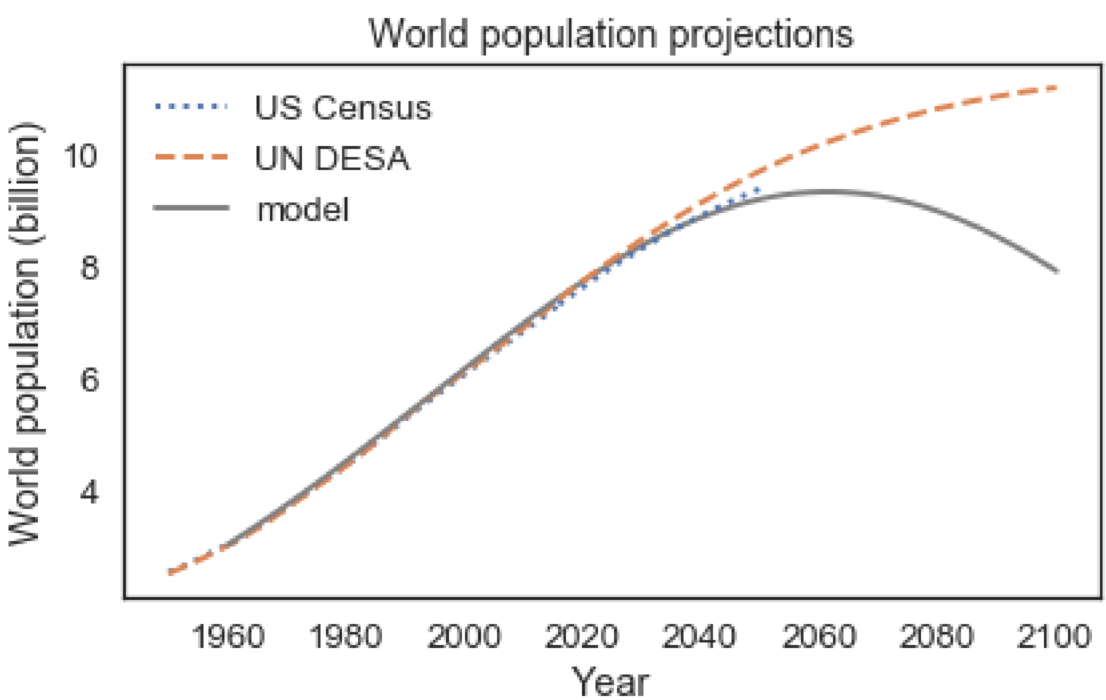

Figure 8: png

Related viewing: You might be interested in this video by Hans Rosling about the demographic changes we expect in this century.