FEDERAL STATE AUTONOMOUS
EDUCATIONAL INSTITUTION OF HIGHER EDUCATION
ITMO UNIVERSITY

Report on the
Discrete mathematical models laboratory task No. 1

“Modeling of demographic processes.”

Performed by

Danila Baklazhenko

J41321c

Accepted by

Dr Ivanov Sergey

St. Petersburg

2022

Goal

Define, formulate, and implement discrete mathematical model that simulates population dynamic of Russian Federation.

Formulation of the problem

Discrete mathematical models deal with discrete objects and processes, rather that continuous. Human population throughout the years may be considered as system of interconnected discrete objects. Hence, it may be simulated with the discrete mathematical model in order to simulate future population dynamic and obtain valuable data.

Solution method

All programming implemented with the Python programming language.

To implement discrete mathematical model of population we need to solve several tasks:

1. Determine the «survival» rates independently for men and women for all age groups (“0-4” -> “5-9” -> “10-14” ...) according to 2000-2005 years (data for Russia or any other country)
2. Determine the fertility rate for women in the age category “20- ... -39”
3. Calculate boys/girls ratio for newborn children
4. Predict the change in the country's population and demographic profile for 100 years and compare with existing prediction

As the starting point and historical data of the model the data from the “population.un.org” was taken as basic model. Also, from the same dataset we obtain results of existing predictions.

To calculate all necessary values three function were defined:

Survival_rate – calculates survival rates for all age groups.

Fertility_rate – calculate fertility rate throughout specified years.

Sex_ration – calculates boy/girls ration for newborns.

The implementation of functions may be seen in Listing 1.

Then, we should define the model of population itself. To achieve that we need to specify some input parameters: historical data, starting year, fertility rates, all survival rates and sex ration. We calculate all parameters with the use of defined in Listing 1 functions and pass them into the model.

The model itself defined as Python class with three methods:

__init__ - class object constructor

simulate_period_constant – method for calculating all parameters and moving population dynamic with constant pre-defined coefficients

simulate_period_dynamic - method for calculating all parameters and moving population dynamic with dynamically changing coefficients

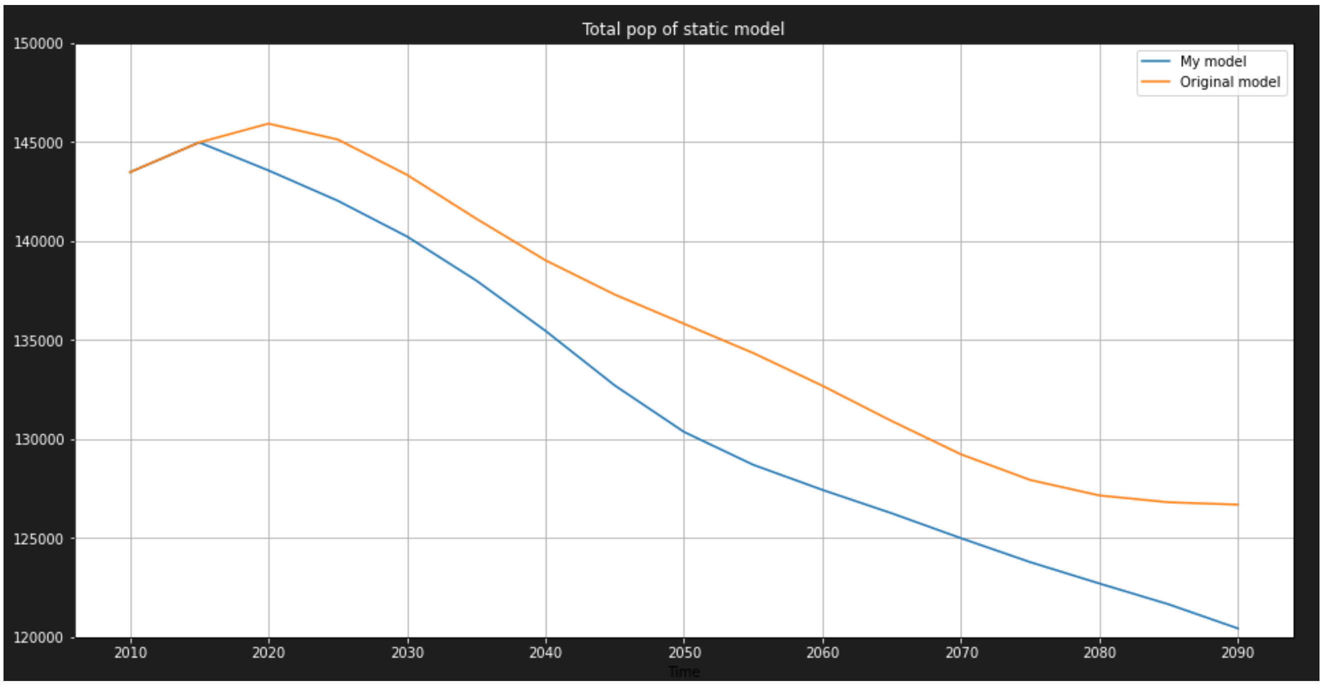
The equations of model movements defined as follows:

$$TotPop = \begin{cases} fertFemale * fertRate & \text{if, } Age = "0-4" \\ oldPop * survRate & for\ all\ age\ groups \end{cases}$$

Each time interval is 5 years since the defined age groups equal to 5 years. With each move of population the year of the model changes as well. The implementation of population model shown in Listing 2.

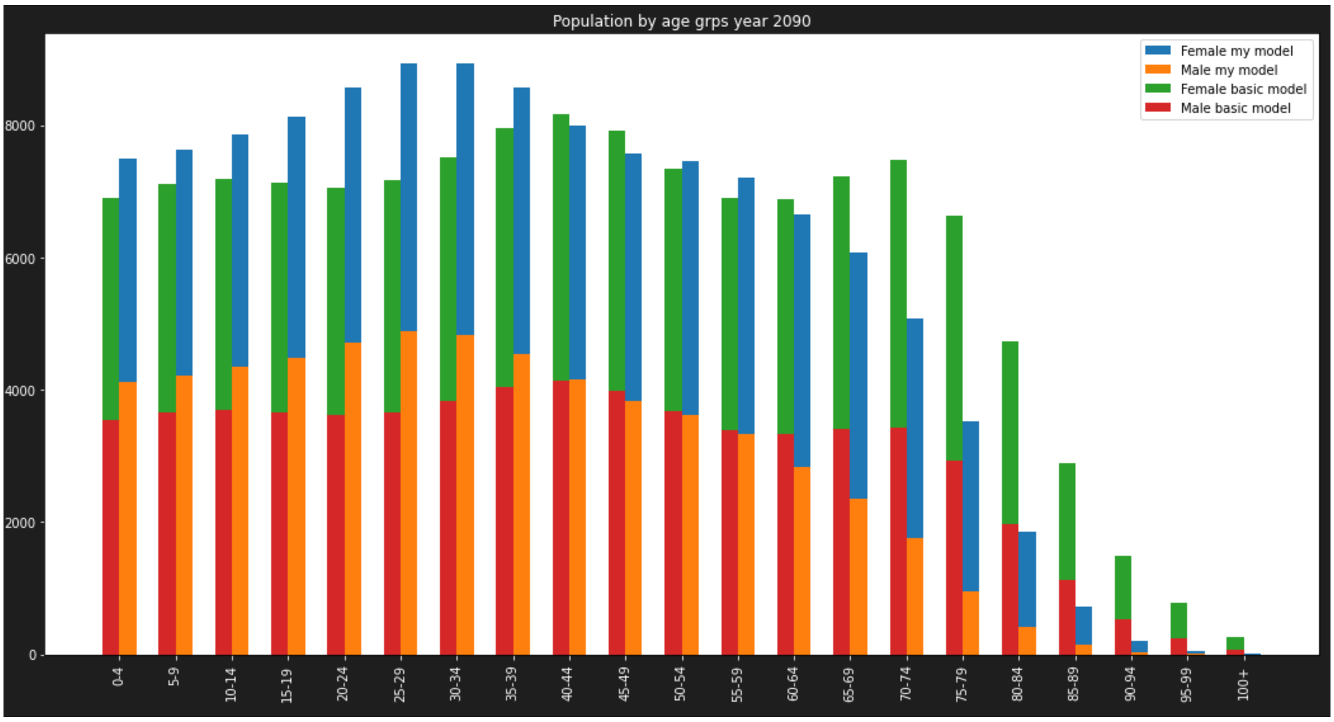
Results

For defined model the forecast was made for 90 years and compared with the basic model. The results was plotted as chart and shown in picture below.



As we can see the population of our model not that much differs from the basic model. The differences may be explained by the different initial point and chosen historical data for coefficient calculations.

To check the validity of the model, the bar chart with population distribution by age groups were plotted. Below you can see the chart for year 2090.



As we can see from the plot the population distribution seems sensible, however slightly different from the basic model.

Conclusions

In this laboratory task the discrete mathematical model for the population was created, defined, and implemented in Python programming language. For this model we gathered the data from "population.un.org", calculate necessary coefficients and made forecasts for the 100 years, as well as compared developed model with the basic model presented as "population.un.org". From the results we can say that the model is sensible, however slightly differs from the basic model.

Appendix

Listing of code and charts in form of Python notebook: GitHub

Listing 1

```python
def sex_ratio(data, year):
    format_data = data[(data["Time"] == year) & (data["AgeGrp"] == '0-4')]
    sex_ratio = format_data.PopMale.iloc[0] / format_data.PopFemale.iloc[0]
    return sex_ratio

def fertility_rate(data, year, fertilyty_ages):
    format_data = data[(data["Time"] == year)]

    female_pop = format_data[(format_data["Time"]==year) &
(format_data["AgeGrp"].isin(fertilyty_ages))].PopFemale.sum()
    born_pop = format_data[(format_data["Time"]==year) & (format_data["AgeGrp"]
== '0-4')].PopTotal.sum()

    fert_rate = born_pop/female_pop

    return fert_rate

def survival_rate(data, period_start = 2000, period_end = 2005):
    format_data = data[data["Time"].isin([period_start, period_end])]

    AgeGrps = format_data.AgeGrp.unique()

    surv_rate_male = []
    surv_rate_fmale = []

    for age in AgeGrps:

        if age != '0-4':
            surv_rate_male.append(format_data[(format_data['AgeGrp'] == age) &
(format_data['Time'] == period_end)].PopMale.iloc[0] /
                                    format_data[(format_data['AgeGrp'] == prev_age) &
(format_data['Time'] == period_start)].PopMale.iloc[0])

            surv_rate_fmale.append(format_data[(format_data['AgeGrp'] == age) &
(format_data['Time'] == period_end)].PopFemale.iloc[0] /
                                    format_data[(format_data['AgeGrp'] == prev_age) &
(format_data['Time'] == period_start)].PopFemale.iloc[0])
        prev_age = age

    return surv_rate_male, surv_rate_fmale
```

Listing 2

```python
# Population model
class Population:
    def __init__(self, pop_data, year, fert_rate, surv_rate_male,
surv_rate_fmale, sex_ratio):
        self.pop_data = pop_data
        self.fert_rate = fert_rate
        self.year = year
        self.surv_rate_male = surv_rate_male
        self.surv_rate_fmale = surv_rate_fmale
        self.sex_ratio = sex_ratio

        AgeGrps = rus_data.AgeGrp.unique()
        self.fertilyty_ages = AgeGrps[4:10]

    def simulate_period_constant(self):
        # Calculate population movement
        #fertility_range = 3
        #new_pops = self.pop_data[(self.pop_data["Time"] == self.year) &
(self.pop_data["AgeGrp"].isin(self.fertilyty_ages))].PopFemale.sum() *
self.fert_rate / fertility_range

        new_pops = self.pop_data[(self.pop_data["Time"] == self.year) &
(self.pop_data["AgeGrp"].isin(self.fertilyty_ages))].PopFemale.sum() *
self.fert_rate

        current_pop = self.pop_data[(self.pop_data["Time"] == self.year)].copy()
        current_pop['surv_rate_m'] = [*self.surv_rate_male, 0]
        current_pop['surv_rate_f'] = [*self.surv_rate_fmale, 0]

        for index, row in current_pop.iterrows():
            if row['AgeGrp'] == '0-4':
                current_pop.loc[index, 'PopMale'] = new_pops * (0.5 - (1 -
self.sex_ratio))
                current_pop.loc[index, 'PopFemale'] = new_pops * (0.5 + (1 -
self.sex_ratio))
            else:
                current_pop.loc[index, 'PopMale'] = prev_row['PopMale'] *
prev_row['surv_rate_m']
                current_pop.loc[index, 'PopFemale'] = prev_row['PopFemale'] *
prev_row['surv_rate_f']

            current_pop.loc[index, 'PopTotal'] = current_pop.loc[index,
'PopMale'] + current_pop.loc[index, 'PopFemale']

            current_pop.loc[index, 'Time'] += 5
            prev_row = row

        self.pop_data = pd.concat([self.pop_data, current_pop],
ignore_index=True)
        self.year += 5
    def simulate_period_dynamic(self):
        # Culculate population movement
        fertility_range = 4.5

        new_pops = self.pop_data[(self.pop_data["Time"] == self.year) &
(self.pop_data["AgeGrp"].isin(self.fertilyty_ages))].PopFemale.sum() *
self.fert_rate / fertility_range

        current_pop = self.pop_data[(self.pop_data["Time"] == self.year)].copy()
        current_pop['surv_rate_m'] = [*self.surv_rate_male, 0]
        current_pop['surv_rate_f'] = [*self.surv_rate_fmale, 0]

        for index, row in current_pop.iterrows():
            if row['AgeGrp'] == '0-4':
                current_pop.loc[index, 'PopMale'] = new_pops * (0.5 - (1 -
self.sex_ratio))
                current_pop.loc[index, 'PopFemale'] = new_pops * (0.5 + (1 -
self.sex_ratio))
            else:
                current_pop.loc[index, 'PopMale'] = prev_row['PopMale'] *
prev_row['surv_rate_m']
                current_pop.loc[index, 'PopFemale'] = prev_row['PopFemale'] *
prev_row['surv_rate_f']

            current_pop.loc[index, 'PopTotal'] = current_pop.loc[index,
'PopMale'] + current_pop.loc[index, 'PopFemale']

            current_pop.loc[index, 'Time'] += 5
            prev_row = row

        self.pop_data = pd.concat([self.pop_data, current_pop],
ignore_index=True)

        # Recalculate coeficients
        fert_rates = [fertility_rate(self.pop_data, year, self.fertilyty_ages)
for year in range(self.year-5, self.year+5, 5)]
        self.fert_rate = sum(fert_rates)/len(fert_rates)

        self.surv_rate_male, self.surv_rate_fmale = survival_rate(self.pop_data,
self.year, self.year+5)

        sex_ratios = [sex_ratio(self.pop_data, year) for year in range(self.year-
5, self.year+5, 5)]
        self.sex_ratio = sum(sex_ratios)/len(sex_ratios)

        self.year += 5
```