IGC Cities that Work

Cities Spatial Model

This user manual describes how to install R and use the open source IGC.CSM package – a Quantitative Spatial Equilibrium (QSE) model toolkit.

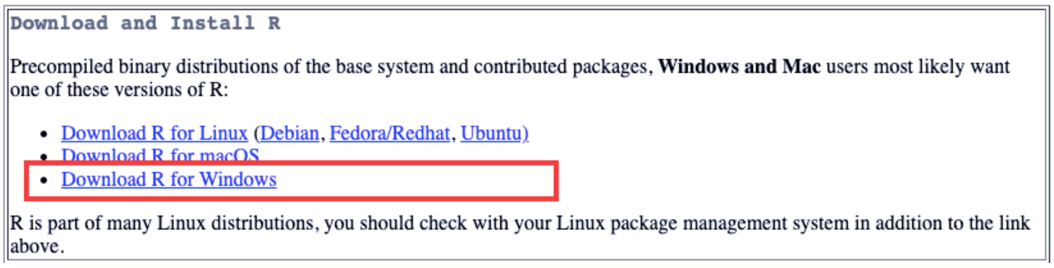
QSE models offer valuable insights for policymakers seeking to assess the impacts of hypothetical urban interventions. The IGC.CSM package aims to bridge the gap between policymakers and cutting-edge advancements in the urban economics literature by fostering a better understanding of QSE models, and facilitating their practical utilisation.

It requires only basic programming knowledge, and a dummy dataset is provided.

Note: this user manual does not include guidance on organising and cleaning the data necessary to apply the model to your own city, nor does it include guidance on visualisation of the results.

To Install R

1. Go to www.cloud.r-project.org.
2. Select the installer you need for your operating system. The rest of these instructions showcase the approach for Windows.



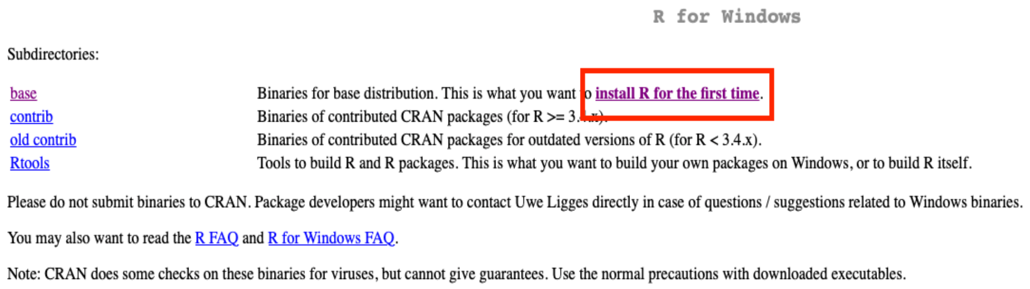
Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Download R for Linux (Debian, Fedora/Redhat, Ubuntu)
- Download R for macOS
- Download R for Windows

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

3. Click on the “install R for the first time” link.



R for Windows

Subdirectories:

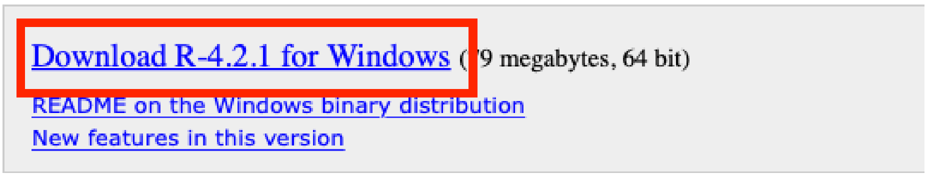
base	Binaries for base distribution. This is what you want to install R for the first time.
contrib	Binaries of contributed CRAN packages (for R >= 3.4.x).
old contrib	Binaries of contributed CRAN packages for outdated versions of R (for R < 3.4.x).
Rtools	Tools to build R and R packages. This is what you want to build your own packages on Windows, or to build R itself.

Please do not submit binaries to CRAN. Package developers might want to contact Uwe Ligges directly in case of questions / suggestions related to Windows binaries.

You may also want to read the R FAQ and R for Windows FAQ.

Note: CRAN does some checks on these binaries for viruses, but cannot guarantee. Use the normal precautions with downloaded executables.

4. Click on the Download R 4.2.1 for Windows link.

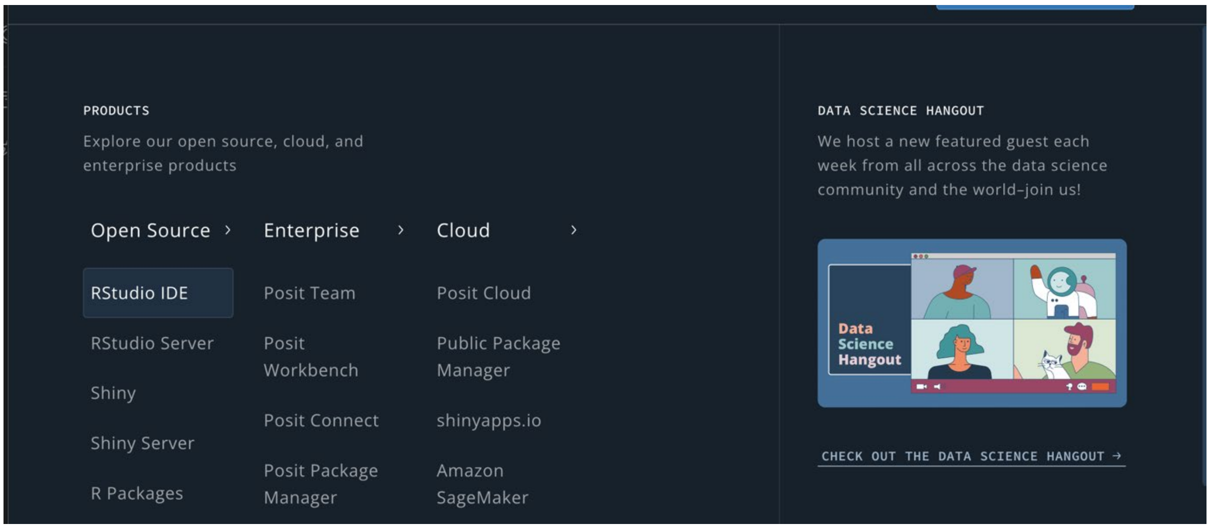


Download R-4.2.1 for Windows (79 megabytes, 64 bit)
README on the Windows binary distribution
New features in this version

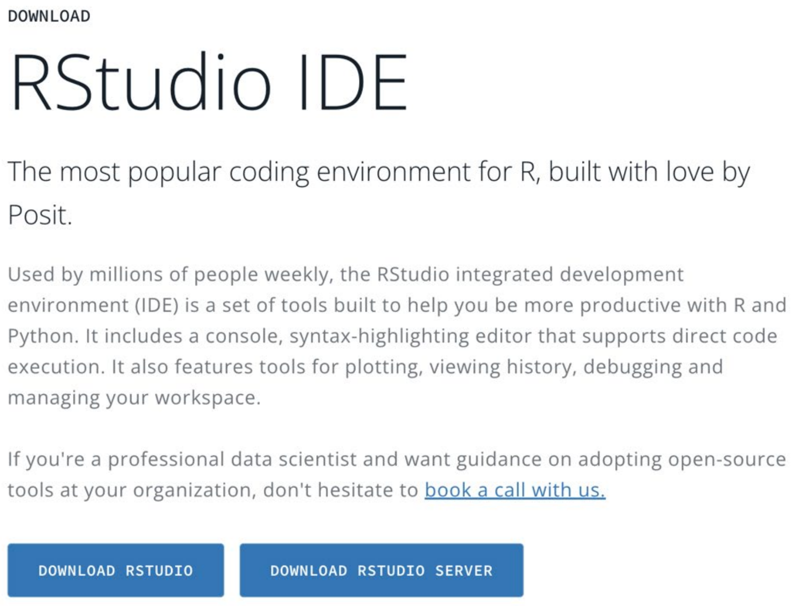
5. Save the .pkg file on your computer. Double-click it to open it, and follow the installation instructions.
6. After installing R, you need to download and install RStudio.

To Install RStudio

1. Go to https://posit.co/
2. At the top of the page, click on "Products" and then "RStudio IDE" under "OPEN SOURCE.



3. At the next page, click the "DOWNLOAD RSTUDIO" button in blue.


DOWNLOAD

RStudio IDE

The most popular coding environment for R, built with love by Posit.

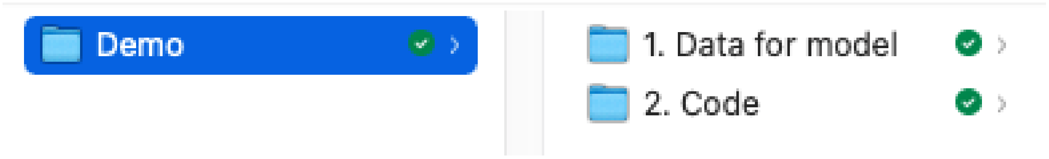
Used by millions of people weekly, the RStudio integrated development environment (IDE) is a set of tools built to help you be more productive with R and Python. It includes a console, syntax-highlighting editor that supports direct code execution. It also features tools for plotting, viewing history, debugging and managing your workspace.

If you're a professional data scientist and want guidance on adopting open-source tools at your organization, don't hesitate to book a call with us.

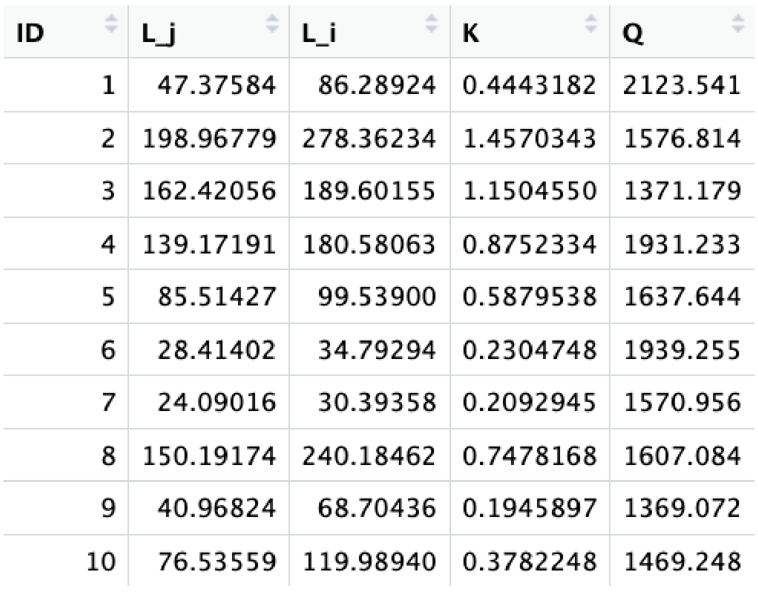
4. Select the installer you need for your operating system. Save the .dmg file on your computer, double-click it to open it, and follow the installation instructions.

To use the IGC.CSM package

Open this link and download the folder named "Demo". It has two subfolders: one with the code that allows us to install and use the package, and another one with the data we will use as input for the model.



1. **Data for model:** In the second subfolder from the Demo folder, you will find two .csv files with the minimum data requirements of the model.
    a. *Chars.csv* contains the main characteristics of each location. Recall from the model documentation that for each spatial unit, L_i and L_j correspond to its total number of residents and workers, K represents its total area, and Q is the average floorspace price. The package does not require the variables to be in any particular units. For example, the total area K could be measured in squared kilometres or squared yards.



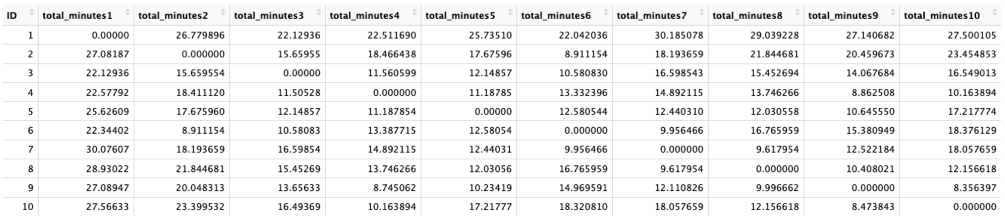
ID	L_j	L_i	K	Q
1	47.37584	86.28924	0.4443182	2123.541
2	198.96779	278.36234	1.4570343	1576.814
3	162.42056	189.60155	1.1504550	1371.179
4	139.17191	180.58063	0.8752334	1931.233
5	85.51427	99.53900	0.5879538	1637.644
6	28.41402	34.79294	0.2304748	1939.255
7	24.09016	30.39358	0.2092945	1570.956
8	150.19174	240.18462	0.7478168	1607.084
9	40.96824	68.70436	0.1945897	1369.072
10	76.53559	119.98940	0.3782248	1469.248

The latter example uses 10 locations, but you can add as many locations as you want when using the code with your own dataset. We suggest not to change any of the variable's names in the input file, but if you do, remember that you should change the names accordingly in the code file "Main.R".

    b. *MatrixTravelTimes_mins.csv* is a matrix of the travel times across all the locations. Each one of its values indicates the time it takes to travel between two locations.

For example, the value placed in column one and row three represents the time it takes to go from location three to location one.

This matrix will always be squared and have zeros on its main diagonal. However, the matrix does not have to be symmetrical.



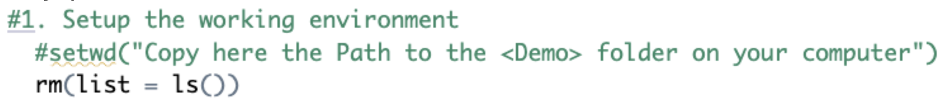
ID	total_minutes1	total_minutes2	total_minutes3	total_minutes4	total_minutes5	total_minutes6	total_minutes7	total_minutes8	total_minutes9	total_minutes10
1	0.00000	26.779896	22.12936	22.511690	25.73510	22.042036	30.185078	29.039228	27.140682	27.500105
2	27.08187	0.000000	15.65955	18.466438	17.67596	8.911154	18.193659	21.844681	20.459673	23.454853
3	22.12936	15.659554	0.00000	11.560599	12.14857	10.580830	16.598543	15.452694	14.067684	16.549013
4	22.57792	18.411120	11.50528	0.000000	11.18785	13.332396	14.892115	13.746266	8.862508	10.163894
5	25.62609	17.675960	12.14857	11.187854	0.00000	12.580544	12.440310	12.030558	10.645550	17.217774
6	22.34402	8.911154	10.58083	13.387715	12.58054	0.000000	9.956466	16.765959	15.380949	18.376129
7	30.07607	18.193659	16.59854	14.892115	12.44031	9.956466	0.000000	9.617954	12.522184	18.057659
8	28.93022	21.844681	15.45269	13.746266	12.03056	16.765959	9.617954	0.000000	10.408021	12.156618
9	27.08947	20.048313	13.65633	8.745062	10.23419	14.969591	12.110826	9.996662	0.000000	8.356397
10	27.56633	23.399532	16.49369	10.163894	17.21777	18.320810	18.057659	12.156618	8.473843	0.000000

The position of locations in files Chars.csv and MatrixTravelTimes_mins.csv must coincide. For example, if a location corresponds to row three in the file Chars.csv, it must be placed on column three and row three in the travel times matrix.

It is important to ensure that none of the variables have missing values or values that are equal to zero. Otherwise, it will not be possible to implement the following steps.

2. **Code:** Inside the code folder, you will find a file named "Main.R". This is the only file you will need to use the package. However, make sure to have an active internet connection since you will need to download the package from an online repository. Then, open the "Main.R" file using Rstudio.

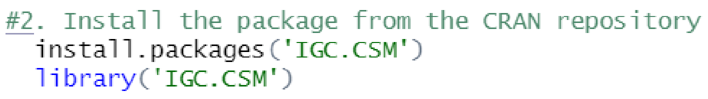
    a. First, you must change the working directory to the folder containing the data and clear the working environment from any previous values.

    ```
    #1. Setup the working environment
      #setwd("Copy here the Path to the <Demo> folder on your computer")
      rm(list = ls())
    ```

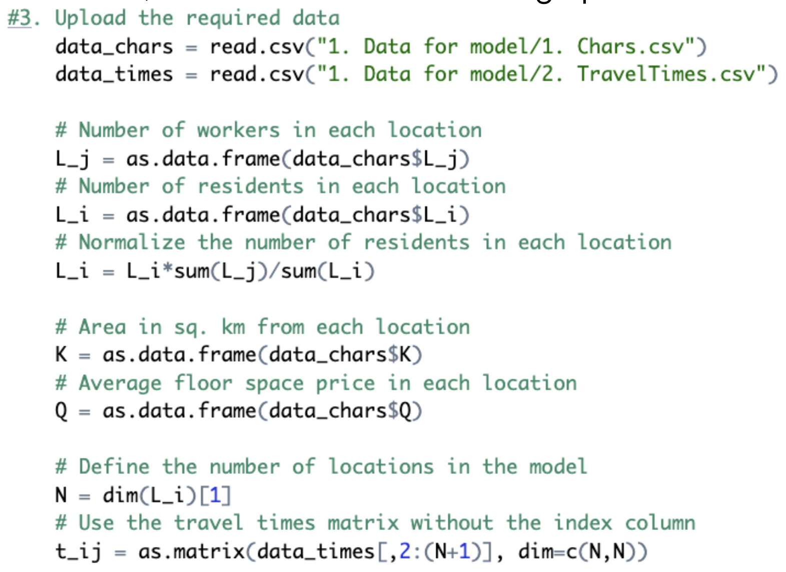
    b. Then, you will need to download the package from the CRAN repository.

    Once you have installed the package for the first time, you will not need to re-install it and you should only run the second line below:

    ```
    #2. Install the package from the CRAN repository
      install.packages('IGC.CSM')
      library('IGC.CSM')
    ```

c. In the third step, you will load the two CSV files (explained in section 1) into the RStudio working space.



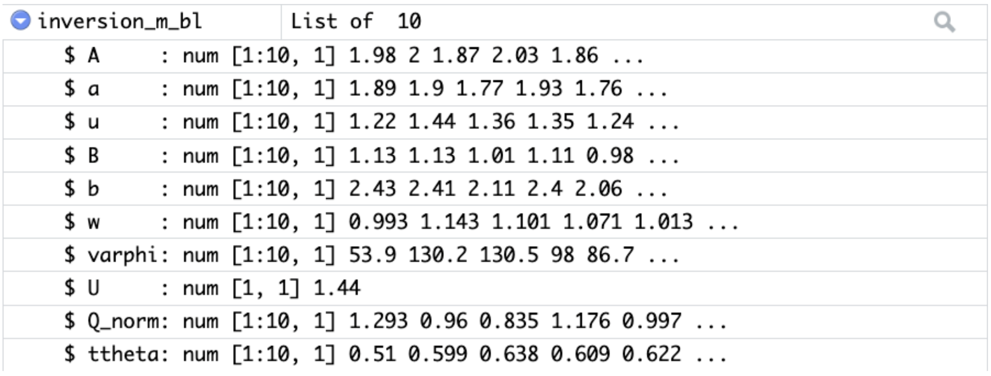
```
#3. Upload the required data
    data_chars = read.csv("1. Data for model/1. Chars.csv")
    data_times = read.csv("1. Data for model/2. TravelTimes.csv")

    # Number of workers in each location
    L_j = as.data.frame(data_chars$L_j)
    # Number of residents in each location
    L_i = as.data.frame(data_chars$L_i)
    # Normalize the number of residents in each location
    L_i = L_i*sum(L_j)/sum(L_i)

    # Area in sq. km from each location
    K = as.data.frame(data_chars$K)
    # Average floor space price in each location
    Q = as.data.frame(data_chars$Q)

    # Define the number of locations in the model
    N = dim(L_i)[1]
    # Use the travel times matrix without the index column
    t_ij = as.matrix(data_times[,2:(N+1)], dim=c(N,N))
```

d. Once you have completed steps a-c, you are ready to use the main functions of the package. First, using "inversionModel", you will save a dataset named "inversion_m_bl" in the R workspace, which contains the following results of the inversion of the model under the baseline scenario:

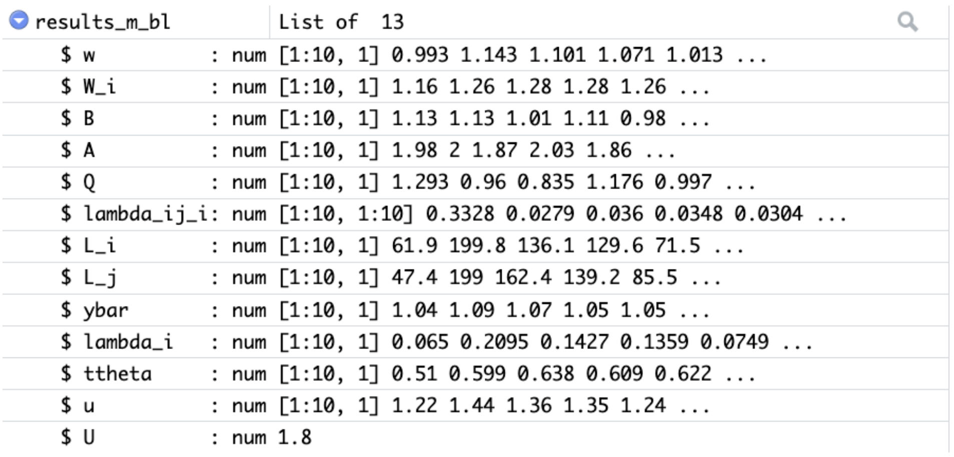


```
inversion_m_bl          List of  10
    $ A     : num [1:10, 1] 1.98 2 1.87 2.03 1.86 ...
    $ a     : num [1:10, 1] 1.89 1.9 1.77 1.93 1.76 ...
    $ u     : num [1:10, 1] 1.22 1.44 1.36 1.35 1.24 ...
    $ B     : num [1:10, 1] 1.13 1.13 1.01 1.11 0.98 ...
    $ b     : num [1:10, 1] 2.43 2.41 2.11 2.4 2.06 ...
    $ w     : num [1:10, 1] 0.993 1.143 1.101 1.071 1.013 ...
    $ varphi: num [1:10, 1] 53.9 130.2 130.5 98 86.7 ...
    $ U     : num [1, 1] 1.44
    $ Q_norm: num [1:10, 1] 1.293 0.96 0.835 1.176 0.997 ...
    $ ttheta: num [1:10, 1] 0.51 0.599 0.638 0.609 0.622 ...
```

- *A* corresponds to the productivity level in each location after incorporating the agglomeration forces.
- *a* corresponds to the productivity level in each location.
- *u* represents the real income measure in each location without the idiosyncratic shock.
- *U* corresponds to the average welfare.
- *b* corresponds to the amenities in each location.
- *B* corresponds to the amenities in each location after incorporating the congestion forces.
- *w* corresponds to the wages that equalise the labour supply predicted in the model with the one observed in the data for each location.

- *varphi* corresponds to the level of development density in each location.
- *Q_norm* corresponds to the normalised floor space prices in each location.
- *ttheta* corresponds to the share of the floor space that is used commercially.

e. Then, with the variables obtained previously, you can solve the model to get an equilibrium. Specifically, using the development density, wages, utility, floorspace prices, and the share of floorspace used commercially, the function “solveModel” yields the number of workers and residents in each location and their real income.



```
results_m_bl        List of  13
    $ w          : num [1:10, 1] 0.993 1.143 1.101 1.071 1.013 ...
    $ W_i        : num [1:10, 1] 1.16 1.26 1.28 1.28 1.26 ...
    $ B          : num [1:10, 1] 1.13 1.13 1.01 1.11 0.98 ...
    $ A          : num [1:10, 1] 1.98 2 1.87 2.03 1.86 ...
    $ Q          : num [1:10, 1] 1.293 0.96 0.835 1.176 0.997 ...
    $ lambda_ij_i: num [1:10, 1:10] 0.3328 0.0279 0.036 0.0348 0.0304 ...
    $ L_i        : num [1:10, 1] 61.9 199.8 136.1 129.6 71.5 ...
    $ L_j        : num [1:10, 1] 47.4 199 162.4 139.2 85.5 ...
    $ ybar       : num [1:10, 1] 1.04 1.09 1.07 1.05 1.05 ...
    $ lambda_i   : num [1:10, 1] 0.065 0.2095 0.1427 0.1359 0.0749 ...
    $ ttheta     : num [1:10, 1] 0.51 0.599 0.638 0.609 0.622 ...
    $ u          : num [1:10, 1] 1.22 1.44 1.36 1.35 1.24 ...
    $ U          : num 1.8
```

- *w* corresponds to the wages that equalise the labour supply and the labour demand according to the exogenous parameters in the model.
- *W_i* corresponds to the market access measure from each location.
- *B* corresponds to the amenities in each location after incorporating the congestion forces.
- *A* corresponds to the productivity level in each location after incorporating the agglomeration forces.
- *Q* corresponds to the floor space prices in each location.
- *lambda_ij_i* corresponds to the commuting share from location i to j, conditional on living in location i
- *L_i* corresponds to the total number of residents in each location after solving the model.
- *L_j* corresponds to the total number of workers in each location after solving the model.
- *u* corresponds to the welfare level in each location, which represents a real income measure.
- *ybar* corresponds to the average income in each location.

- *lambda_i* corresponds to the share of workers that decide to live in location i.
- *ttheta* corresponds to the share of the floor space that is used commercially after the shock.
- *U* corresponds to the aggregate welfare level in the city.

All the results have the same order as the input data. Therefore, if a location was in the second row in the data from section 1, its results will also be in that row.

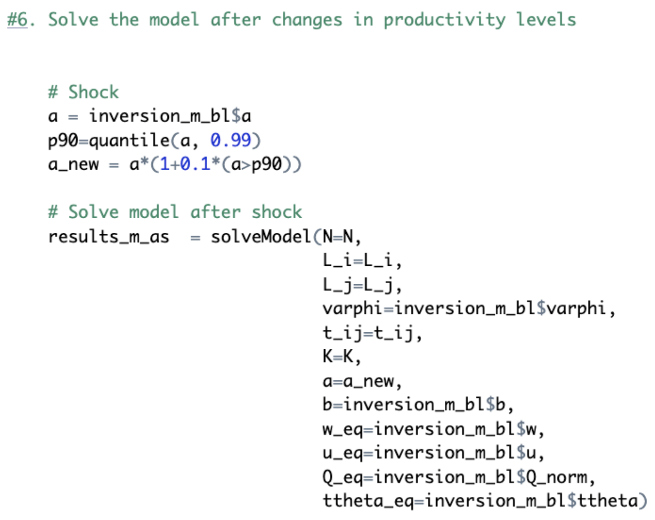
In the baseline scenario, the output variables from the function *inversionModel* and the function *solveModel* coincide. In the next step, you will see that when the model incorporates some of the policy changes, the variables differ from those obtained in the baseline scenario.

Similarly, you can use the function *solveModel* to find the model's equilibrium after incorporating changes in some of the model variables that reflect policies implemented by the government.

To implement the policy changes in the code, you need to change the variables that the *solveModel* function uses as inputs. The table below explains which variables should be changed for a couple of policy interventions.

Policy intervention	Change
Transport infrastructure improvements	t_ij: the travel times across locations
Business development	a_i: the productivity levels
Revitalisation and building projects	b_i: the level of amenities
Housing subsidies	Q: the floorspace price
Wage subsidies	w: the salaries
Land development	K: the total area available
Migration policies	L_i and L_j: the number of workers and residents

For example, after assuming an increase in the baseline productivity, the function *SolveModel* can compute the same variables from the previous step and save them in the data frame "results_m_as".



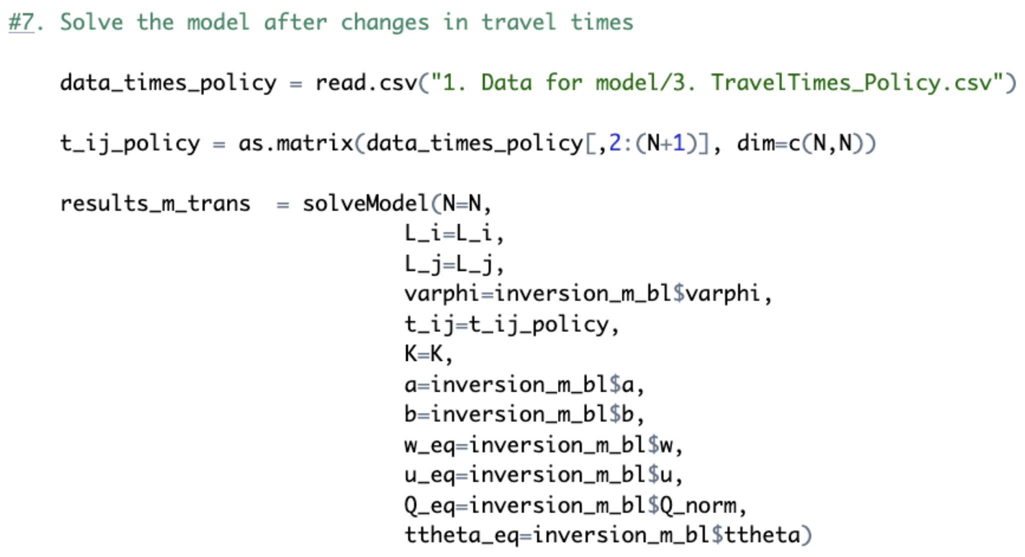
```
#6. Solve the model after changes in productivity levels


    # Shock
    a = inversion_m_bl$a
    p90=quantile(a, 0.99)
    a_new = a*(1+0.1*(a>p90))

    # Solve model after shock
    results_m_as  = solveModel(N=N,
                               L_i=L_i,
                               L_j=L_j,
                               varphi=inversion_m_bl$varphi,
                               t_ij=t_ij,
                               K=K,
                               a=a_new,
                               b=inversion_m_bl$b,
                               w_eq=inversion_m_bl$w,
                               u_eq=inversion_m_bl$u,
                               Q_eq=inversion_m_bl$Q_norm,
                               ttheta_eq=inversion_m_bl$ttheta)
```

By comparing the variables obtained in the baseline scenario with those obtained after incorporating the policy changes, this toolkit can be used to estimate their potential impact.

Similarly, after assuming a change in the travel times, the function SolveModel can compute a new equilibrium and store the result variables in a data frame.



```
#7. Solve the model after changes in travel times

    data_times_policy = read.csv("1. Data for model/3. TravelTimes_Policy.csv")

    t_ij_policy = as.matrix(data_times_policy[,2:(N+1)], dim=c(N,N))

    results_m_trans  = solveModel(N=N,
                                  L_i=L_i,
                                  L_j=L_j,
                                  varphi=inversion_m_bl$varphi,
                                  t_ij=t_ij_policy,
                                  K=K,
                                  a=inversion_m_bl$a,
                                  b=inversion_m_bl$b,
                                  w_eq=inversion_m_bl$w,
                                  u_eq=inversion_m_bl$u,
                                  Q_eq=inversion_m_bl$Q_norm,
                                  ttheta_eq=inversion_m_bl$ttheta)
```