

# Population Synthesis with Quasirandom Integer Sampling

*Andrew P Smith, Robin Lovelace and Mark Birkin*

*23 March 2017*

---

Established methods for synthesising a population from geographically aggregated data are robust and well understood. However, most rely on the potentially detrimental process of “integerisation” if a whole-individual population is required, e.g. for use in Agent-Based Modelling. This paper investigates a sampling method of synthesis that is computationally efficient and results directly in integer populations. We showcase the concept of quasirandom sequences and demonstrate that they perform far better than pseudorandom numbers in this context. We demonstrate that sampling the marginal distributions *without replacement* is crucial, and perform some comparative statistical analyses of example populations. Finally we highlight the advantages and disadvantages of our method over the established ones. The implementation is available in an open-source R package `humanleague`.

---

## 1 Introduction

Iterative Proportional Fitting (IPF) is a popular and well-established technique for generating synthetic populations from marginal data. Compared with other methods of ‘population synthesis’ IPF is relatively fast, simple and easy to implement, as showcased in a practical introduction to the subject by Lovelace and Dumont (2016). Furthermore, a number of software packages aimed at facilitating IPF for spatial microsimulation and other applications have been published (e.g. Barthélemy and Suesse 2016; Jones, Lovelace, and Dumont 2016).

However, the method has various limitations from the perspective of spatial microsimulation for social simulation. For example, IPF generates fractional weights instead of integer populations, an issue tackled by ‘integerisation’ techniques (Lovelace and Ballas 2013); handles ‘empty cells’ poorly (Lovelace et al. 2015); and requires a representative individual-level ‘seed’ population. The most in-depth exploration of such limitations and the mathematical behaviour of IPF to date is an outstanding PhD thesis on the subject (Zaloznik 2011).

In this paper we investigate sampling techniques where a population matrix is built exclusively in the integer domain. The research reported in this paper is partly motivated by the need to generate individual simulated agents for an entire city. Journey patterns are imputed to each member of the simulated population using app-based mobility traces in order to generate travel demand schedules for an entire urban population in pseudo-real-time.

We show that integer sampling can only work effectively if a quasirandom (as opposed to a pseudorandom) number generator is used, and this in itself is not sufficient to work in all cases. We also show that this problem can be overcome by combining the quasirandom generator with a sampling-without-replacement algorithm. We then look at the statistical properties of the resulting populations, comparing them to populations generated with the IPF algorithm.

## 2 Theory

### 2.1 Degeneracy and Statistical Likelihood

In this work we define degeneracy as the number of possible different ways a given overall population  $P$  can be sampled into a table  $\mathbf{T}$  containing  $S$  possible states  $\mathbf{k}$ , with occupancy probability  $p_{\mathbf{k}}$ , the system having  $d$  degrees of freedom.<sup>1</sup> Making the assumption that marginals are uncorrelated, then the higher the degeneracy of the population, the more statistically likely the population is.

We measure the degeneracy of populations using a  $\chi^2$  statistic:

$$\chi^2 = \frac{\sum_{\mathbf{k}} (\mathbf{T}_{\mathbf{k}} - p_{\mathbf{k}} S)^2}{p_{\mathbf{k}} S}$$

from which we can estimate a p-value, which represents the statistical significance of the synthetic population.

$$p = 1 - \mathbb{P}[d/2, \chi^2]$$

where  $\mathbb{P}$  is the cumulative  $\chi^2$  density function and  $d$  the degrees of freedom of the system. The aim in this context is to generate populations with a high p-value, i.e. a low statistical significance and a high-degeneracy.

### 2.2 Quasirandom Numbers

Quasirandom numbers, often referred to as low discrepancy sequences, are preferential to pseudorandom numbers in many applications, despite not having some of the (appearance of) randomness that good pseudorandom generators possess. In this work we focus on the Sobol quasirandom sequence (Bratley and Fox 1988; Joe and Kuo 2003).

Figure 1 qualitatively illustrates the difference between pseudo- and quasirandom sampling. Each plot contains 2000 uniform points in two dimensions. The quasirandom samples fill the sample space far more evenly and display an obvious lack of randomness, clearly showing a lack of independence between samples. In contrast, the pseudorandom samples show no discernible pattern with clusters and gaps present in the sampling domain, suggesting independence between variates.

---

<sup>1</sup>These terms are defined more precisely in section 2.4.

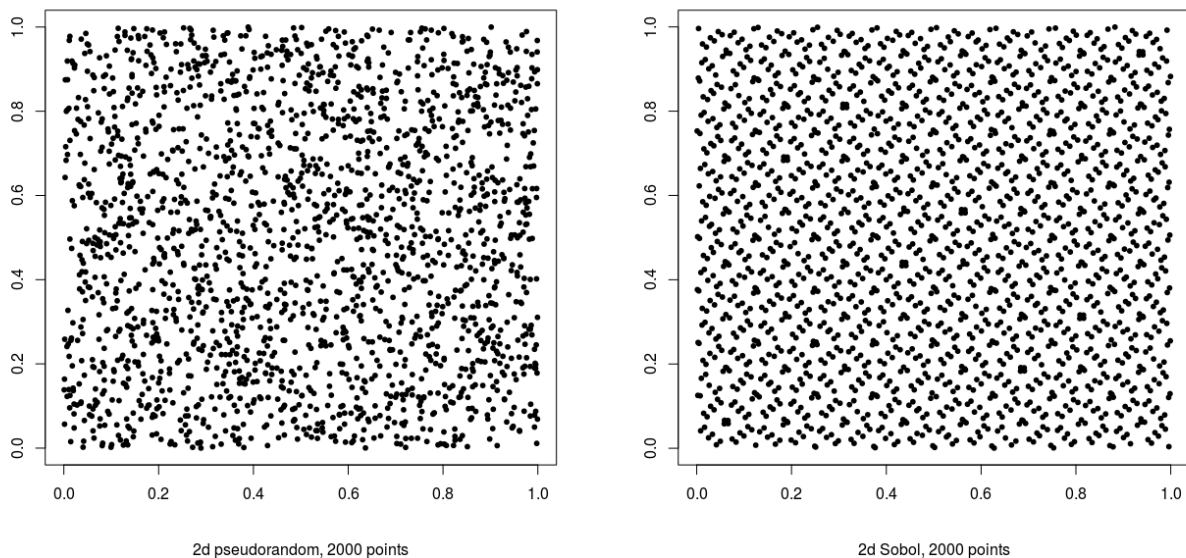


Figure 1: Bivariate uniform samples - pseudorandom (left) and quasirandom (right).

Sobol sequences, as with other quasirandom sequences, have an inherent dimensionality and a relatively short period - the sequence is exhausted once every integer in the sample space has been sampled. Successive samples evenly fill the sample space, and thus lack independence. Conversely, a good pseudorandom generator has no discernible dependence between variates often has a much longer period, allowing for very large samples to be taken.

For applications such as numerical integration, Sobol sequences converge at a rate of  $\approx 1/N$  (actually  $(\ln N)^D/N$  (Press et al. 2017) compared to  $\approx 1/\sqrt{N}$  for a pseudorandom generator, and thus require far fewer samples to achieve the same level of accuracy.

Quasirandom sequences are not seeded like pseudorandom generators. To avoid repetition, and for better degeneracy, it is recommended to discard a number of variates on initialisation (Joe and Kuo 2003), and on subsequent sampling continue the sequence from its previous position.

In the later analysis of our algorithm, for comparison, we use the MT19937 variant of the Mersenne Twister as our pseudorandom generator (Matsumoto and Nishimura 1998).

## 2.3 Sampling without Replacement

Given discretely distributed population  $P$  with integral frequencies  $\{f_1, f_2, \dots, f_n\}$ , a random sample  $i \in \{1 \dots n\}$  has probability

$$p(i) = \frac{f_i}{\sum_{j=0}^n f_j}$$

Once a sample  $i$  is taken the distribution is adjusted to  $\{f_1, f_2, \dots, f_i - 1, \dots, f_n\}$ , with resultant impact on  $p$ . Once  $f_i$  has reached zero no further samples can take the value  $i$  since  $p(i) = 0$ .

Since  $f_i$  cannot be oversampled, this implies that all the other states  $f_{j \neq i}$  cannot be undersampled. By extension, if all but one states  $f_{i \neq j}$  cannot be oversampled, then  $f_j$  cannot be undersampled. Thus each state can neither be under- nor oversampled: the distribution must be matched exactly.

## 2.4 Problem Statement

Population synthesis (Orcutt 1957) refers to the (re)generation of individual-level data from aggregate (marginal) data. For example, UK census data provides aggregate data across numerous socio-economic categories at various geographical resolutions - but it expressly does not provide data at an individual-level, for privacy reasons among others.

A mathematical statement of population synthesis is as follows:<sup>2</sup>

Given a population  $P$  and a set of  $D$  marginal vectors  $\mathbf{m}_i$  of length  $l_i$  such that we have  $S$  possible states

$$S = \prod_{i=1}^D l_i$$

Find a contingency table  $\mathbf{T}$  in  $D$  dimensions such that

$$\sum_{\mathbf{k}, k_i \text{ fixed}} \mathbf{T}_{\mathbf{k}} = \mathbf{m}_i$$

in other words each element of  $\mathbf{m}_i$  is the sum of all elements in  $\mathbf{T}$  for a fixed value of  $k_i$ . We define this as the *marginal constraint*.

Each marginal sum and the sum of the elements of contingency table must equal the population  $P$ :

$$\sum \mathbf{m}_i = \sum_{\mathbf{k}} \mathbf{T} = P$$

We define this as the *population constraint*. Finally, the *integral constraint* restricts the elements to the natural numbers:

$$\mathbf{T} \in \mathbb{N}^S, \mathbf{m}_i \in \mathbb{N}^{l_i}$$

The probability of a given state  $\mathbf{k}$  being occupied is thus the product of the marginal probabilities:

$$p_{\mathbf{k}} = \prod_{i=1}^D (\mathbf{m}_i)_{\mathbf{k}} / P$$

And the degrees of freedom  $d$  of the system (required in order to calculate the statistical significance of the population) is given by

$$d = \prod_i l_i - 1$$

In the general case there are not enough constraints to determine a unique solution. Hence there is a need to resort to iterative or optimisation-type solutions, such as IPF, simulated annealing, likelihood estimation, chi-squared fitting, or least-squares fitting.

---

<sup>2</sup>In the notation below, the index  $i$  is scalar and refers to a particular dimension. The index  $\mathbf{k}$  is a vector index  $\{k_1, k_2, \dots, k_D\}$  of length  $D$ , the dimensionality of the problem.

### 3 The Algorithm

The final version of the algorithm is outlined in this section, a description of the process by which we arrived at this algorithm follows in the Discussion.

1. Create an  $D$ -dimensional discrete without-replacement distribution using the marginals  $\mathbf{m}_i$
2. Sample  $P$  random variates from this distribution to create a population  $\mathbf{T}$ . Specifically, we sample a value of  $\mathbf{k}$  and increment the value of  $\mathbf{T}_{\mathbf{k}}$ , repeating until the distribution is fully depleted. Constructing the problem in this way automatically ensures that all constraints are automatically met, as explained in the previous section.
3. Compute occupation probabilities  $p_{\mathbf{k}}$  for each state  $\mathbf{k}$ .
4. Compute a  $\chi^2$  statistic and a p-value, which represents the statistical significance of the synthetic population (in this case we prefer higher p-values, i.e. statistically insignificant populations).

#### 3.1 The Implementation

The algorithm is implemented in an R package called **humanleague**, written in C++11 and C and exposed to R via Rcpp. We used open source implementations of the Sobol sequence generator (Johnson 2017) and the incomplete gamma function (Burkardt 2008).

One of the many useful features introduced in C++11 was a standard random number framework, which splits underlying (uniform) generators and the distributions, and defines APIs that these classes should implement (see e.g. <http://en.cppreference.com/w/cpp/numeric/random>). There is no native support for quasirandom generators nor for without-replacement distributions, but by conforming to the APIs they can be implemented to interoperate with the native generator and distribution implementations.

It should be noted that there are two instances where the new types extend the standard interfaces (see class definitions below):

- a quasirandom generator has inherent dimensionality, and thus should return that number of variates per sample. In our implementation we provided both the standard single-valued `operator()` and the extended vector `buf()` accessor functions.
- without-replacement sampling will eventually become exhausted, as a result we provided a boolean `empty()` function to check if the distribution is exhausted. If exhausted, the accessor functions will throw an exception.

The remainder of the code was developed by the authors. The Sobol sequence generator is implemented in such a way that it is not reset each time a population is requested, allowing different populations to be generated each time, up to the limit of the sequence being exhausted. Our implementation uses 32-bit unsigned integers, thus allowing for  $\approx 4 \times 10^9$  samples (in each dimension).

```
class Sobol
{
public:
    typedef uint32_t result_type;
    explicit Sobol(uint32_t dim, result_type nSkip = 0u);
    ~Sobol();
    const std::vector<result_type>& buf();
    result_type operator()();
    void skip(result_type n);
    result_type min() const;
    result_type max() const;
private:
    SobolData* m_s;
```

```

uint32_t m_dim;
std::vector<result_type> m_buf;
uint32_t m_pos;
};

template<typename I> // I must be an integral type
class discrete_distribution_without_replacement
{
public:
    typedef I result_type;
    // enforce integral types only
    static_assert(std::is_integral<I>::value,
        "discrete_distribution_without_replacement: only integral types supported");
    discrete_distribution_without_replacement(typename std::vector<I>::const_iterator b,
                                              typename std::vector<I>::const_iterator e);

    // std::distribution compatibility
    template<typename R>
    result_type operator()(R& rng);
    result_type operator()(result_type r);
    bool empty() const;
private:
    std::vector<I> m_freq;
    I m_sum;
};

```

The package is open source and available at <https://github.com/CatchDat/humanleague>. Plans are in progress to submit it the official CRAN repository.

The implementation supports dimensionalities up to 12, although this limit is arbitrary and could be increased if necessary. Input is simply a list of integer vectors representing the marginals. Marginal vectors must all sum to the population (to satisfy the population constraint)  $P$ .

The output is broadly compatible with the established `mipfp` (Barthelemy & Suesse 2016) R package:

- $D$ -dimensional population table  $\mathbf{T}_k$
- $D$ -dimensional occupancy probability array  $\mathbf{p}_k$
- boolean value indicating convergence
- maximum value of each residual vector
- $\chi^2$  statistic
- p-value

as can be seen by the example output below:

```

> x=c(10,10,10)
> p=humanleague::synthPop(list(x,x))
> p
$method
[1] "qiws"

$conv
[1] TRUE

$chiSq
[1] 1.2

$pValue

```

```

[1] 0.8780986

$error.margins
[1] 0 0

$p.hat
      [,1]      [,2]      [,3]
[1,] 0.1111111 0.1111111 0.1111111
[2,] 0.1111111 0.1111111 0.1111111
[3,] 0.1111111 0.1111111 0.1111111

$x.hat
      [,1] [,2] [,3]
[1,]    4    3    3
[2,]    4    3    3
[3,]    2    4    4

```

## 4 Comparison to Existing Methods

In this section we present results from the final version of the sampling algorithm. The process by which we arrived at this version is given in section 5.1.

### 4.1 Statistical Properties

For this we compared the QIWS algorithm to IPF using a number of two-dimensional test cases.

No integerisation was done to the IPF data (using e.g. the R package **raker**) due to that fact that integerisation can cause a mismatch in one or more marginals.

IPF was initialised with a random seed array, each element uniformly distributed in  $[0, 2)$ , so that for the p-value tests the initial total population would be approximately equal to the final total.

The first three tests are made-up examples. The remainder are marginal figures derived from 2011 UK census data at the MSOA level (mid-layer super output area), with one dimension representing person status in terms of age, sex, and economic activity, the other their workplace location (MSOA).

Pop	States	N	QIWS $\bar{c}$	QIWS $\bar{p}$	IPF $\bar{c}$	IPF $\bar{p}$
20	4	1000	1	0.79	1	0.48
125	25	1000	1	0.91	1	0.06
935	49	1000	1	0.86	1	0.03
4958	16032	1000	1	0.81	1	1
4098	11760	1000	1	0.8	1	1
4029	11904	1000	1	0.8	1	1
4989	14640	1000	1	0.79	1	1
5219	15168	1000	1	0.79	1	1

Table 1: Convergence behaviour of population synthesis algorithms.  $N$  is the number of times the algorithm was run,  $\bar{c}$  refers to the convergence success rate, and  $\bar{p}$  refers to the mean p-value.

A more detailed analysis was performed on two evenly distributed marginal vectors of length 10 and a population  $P = 100$  and number of states  $S = 100$ , sampling 10000 populations.

For reference, the histogram of p-values using IPF (with random seed values) yields the following rather

unsurprising result <sup>3</sup>:

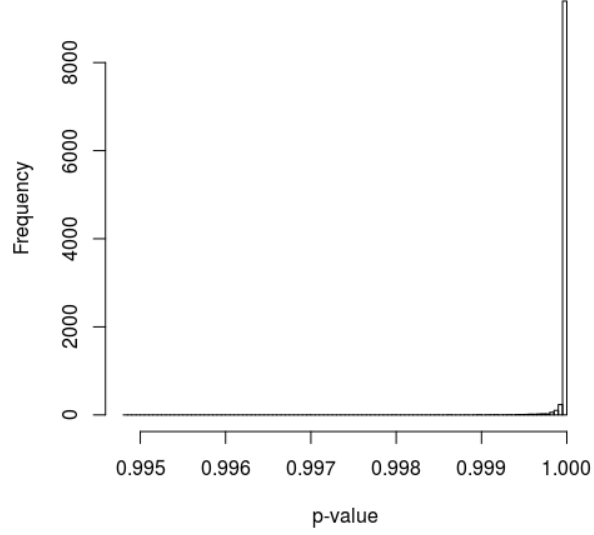


Figure 2: Distribution of p-values of IPF.

This is entirely expected, since the IPF algorithm

We compared the statistical performance of our integer without-replacement sampling using a pseudorandom generator and a quasirandom generator. The algorithm is guaranteed to work in both cases so we looked at the distribution of p-values, which can be seen in figure 3.

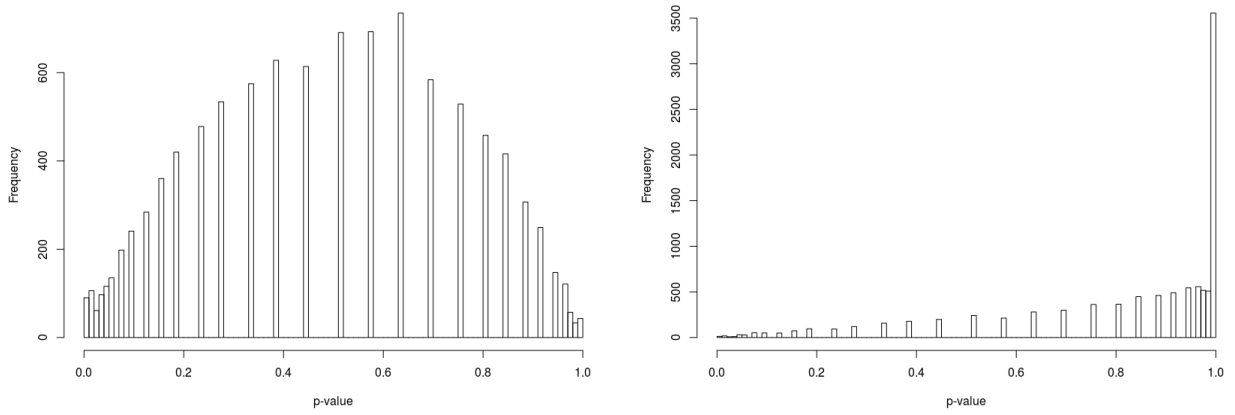


Figure 3: Distribution of p-values of QIWS using pseudorandom (left) and quasirandom (right) sampling.

---

<sup>3</sup>The IPF algorithm seeks to maximise entropy, resulting in a low  $\chi^2$  statistic (Lovelace et al. 2015).



## 4.2 Performance

Since computers vary widely in performance for many reasons, we present normalised performance values.

Pop	States	N	QIWS	IPF
20	4	1000	1.0	66.3
125	25	1000	1.3	96.7
935	49	1000	2.3	68.3
4958	16032	1000	30.7	1559.3
4098	11760	1000	23.0	1069.0
4029	11904	1000	22.7	1027.3
4989	14640	1000	28.3	1457.0
5219	15168	1000	28.7	1557.7

Table 2: Relative computation time of the algorithms, using the first case as a baseline.

## 5 Discussion

### 5.1 Evolution of the algorithm.

The motivation behind the development of this method came from the fact that IPF, in general, does not generate integral populations in each state. If the resulting population is used as a basis for Agent-Based Modelling (ABM), then an integerisation process must be applied to the synthesised population. Whilst this process preserves the overall population, it may not exactly preserve all of the marginal distributions.

Although integerisation may not introduce significant errors, the authors considered whether there was an alternative method of synthesis that would directly result in an integral population that exactly matched the marginal distributions.

We could not find any references in the literature on the use of quasirandom sequences for the purpose of population synthesis, and so decided to investigate the suitability of sampling algorithms using such sequences.

Firstly, it was demonstrated that naive pseudorandom sampling could not reliably generate populations matching the marginals. Since successive samples are ostensibly independent, there is no mechanism to reduce under- or oversampling of any particular state.

The initial algorithm simply sampled the marginal distributions (with replacement), using quasirandom numbers, to build a population. For “toy” problems this method often worked, but for more complex problems the algorithm often failed to exactly match all the marginals.

In general, the resulting population mismatched the marginals only slightly. A correction algorithm was implemented which, applied at most once for each dimension of the population, adjusted the states by the error in the marginal for that dimension. This corrected the overall population, and is reminiscent of an iteration in IPF.

It was also demonstrated that a correction step as described above could not be applied to a pseudorandomly generated population because the marginal errors were almost always too large to be able to apply a correction: such corrections would typically result in a negative population in one or more states.

Furthermore, it was shown that even using quasirandom sampling, it was not always guaranteed that a population could be corrected using the method described above without a negative population in one or more states. The sparser (the ratio of the total population  $P$  to the number of possible states  $S$ ) the population, the more often this would occur. Often the solution was simply to discard and resample the population until an exact or correctable population was generated. This raised a concern about the reliability (and efficiency)

of the algorithm, and in fact further testing did reveal cases where the algorithm the number of resamples required was unacceptably large, even exhausting the period of the generator.

This flaw drove the authors to come up with an alternative formulation of the algorithm, and it transpired that without-replacement sampling would guarantee that the population matched all marginals by design (as explained in section 2.2), eliminating the need for the correction step entirely. In fact, this method is guaranteed to work regardless of the choice of underlying random number generation.

The next phase of the work was to analyse generated populations statistically, focussing on the degeneracy of the resulting populations. The results from this work, presented in the previous section, are discussed below.

## 5.2 Statistical Properties

### 5.2.1 General Remarks

Neither IPF nor QIWS failed to converge on any of the test cases in Table 1.

One unexpected observation was that IPF performed least well on the simple tests, generating the lowest mean p-values. Since these are “toy” examples we do not attribute any significance to this, putting it down to our choice of random seed.

### 5.2.2 Randomness in Context

To put pseudorandom versus quasirandom sampling into context, consider a “random” number generator that simply always returns zero. If we used this as the basis for without-replacement sampling of a marginal distribution  $\{10, 10, 10\}$ , we would get a sequence consisting of ten ones, ten twos and ten threes as the samples exhausted each bin and moved to the next. In two dimensions, this would result in a population matrix

$$\begin{pmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 10 \end{pmatrix}$$

suggesting that the marginals are strongly correlated. This particular result has a low degeneracy - there are relatively few different ways at arriving at this result - and a p-value of effectively zero implying that it is vanishingly unlikely that this population resulted by chance.

### 5.2.3 Degeneracy

The  $\chi^2$  value resulting from IPF is likely to be slightly lower than QIWS due to the fact the former numbers are not generally integral. Figure 2 demonstrates that IPF almost always results in a p-value of 1. This is expected due to the manner in which that the IPF algorithm works.

QIWS does not achieve such high p-values in general, as seen in figure 3, and there is a marked difference in the distribution of p-values depending on the type of underlying random generator used. It can be seen clearly that quasirandom sampling generally yields more degenerate populations, and is much closer to IPF in this respect.

It should be borne in mind that one of the reasons that QIWS cannot achieve the high p-value levels as consistently as IPF is due to the simple fact that the population is integral, and this will generally result in a higher  $\chi^2$  statistic. We would caution users, however, to check the resulting p-value against some suitable threshold and resample if necessary.

### 5.3 Performance

QIWS performance is far superior to IPF. One reason for this is that QIWS is not iterative, but the following points should be noted:

- the comparison is somewhat unfair since our implementation is compiled C and C++ code and Ipfp’s implementation is interpreted R code.
- performance of population synthesis may not be a significant factor in most workflows, and thus speed may be of marginal advantage. However in large-scale ensemble simulation applications, the performance of QIWS may be an advantage.

### 5.4 Application / Range of Use

IPF can be used for microsimulation in cases where some marginal data is not fully specified. For example, some categorical constraint data may only be available for a subset of the whole population. By using this data in the initial guess for the iteration, rather than as a marginal constraint, the incomplete data is scaled to, and smoothed over, the overall population. An example of this type of microsimulation “CakeMap” is given in Lovelace and Dumont (2016).

QIWS requires that all marginals are fully-specified. It needs this data to construct a multivariate discrete distribution to sample from. It is thus not applicable to problems such as “CakeMap”.

For agent-based modelling (ABM) applications, QIWS has the advantage that it always generates integer populations. Thus integerisation, which may alter the population in such a way that the marginals are no longer met, is not required.

QIWS may potentially be of use in the process of anonymising populations. It is outside the scope of this work, but since full marginal data will likely be available, its use in this context may be worth investigating.

## 6 Conclusion

This work forms part of a larger project in which a city-scale population microsynthesis is used as the basis for an Agent-Based Model of commute patterns, using census data combined with a crowdsourced dataset. The QIWS algorithm and the software implementation in an R package was developed for this purpose, eliminating the need for integerising the synthesised population. This microsimulation involved categorical variables (i.e. marginals) representing home and work locations (at MSOA resolution), mode of transport, gender, age group, and economic activity type, with the crowdsourced data overlaid on the larger census population.

The QIWS algorithm was thus designed with this type of synthesis in mind, where all marginal data is fully specified, and in this context it fulfils the requirements.

If there are specific requirements for a degenerate population, statistical data is provided about the generated population that will guide the user in determining the suitability of the generated population.

In situations where integral populations are desirable QIWS presents an advantage over established techniques by eliminating the need for postprocessing the population - for instance by integerisation.

In applications where performance is a key factor, we have shown that our implementation of QIWS comfortably outperforms a popular R implementation of IPF (Barthelemy and Suesse 2016). For large-scale ensemble modelling, this could be advantageous.

We have established that QIWS can be used for microsynthesis given aggregate data. Although outside the scope of this work, we believe that the algorithm could equally be applied to an anonymisation process whereby individual-level data is first aggregated and then synthesised, for example (Nowok, Raab, and Dibben 2016).

We are aware that QIWS is more limited in scope than IPF in that it is unable to deal with cases where some marginal data is not explicitly specified, instead being expressed in the seed population used to initialise the IPF iteration. It may be that with further development of the algorithm some of these limitations could be overcome.

As such we envisage QIWS as a technique that complements rather than supplants established methods, and publicise the technique as such.

## 7 Acknowledgements

The authors acknowledge Innovate UK's Catch! project for providing the funding for this work.

## References

- Barthelemy, Johan, and Thomas Suesse. 2016. "CRAN - Package Mipfp." <https://cran.r-project.org/web/packages/mipfp/index.html>.
- Bratley, Paul, and Bennett L. Fox. 1988. "Algorithm 659: Implementing Sobol's Quasirandom Sequence Generator." *ACM Transactions on Mathematical Software (TOMS)* 14 (1): 88–100. <http://dl.acm.org/citation.cfm?id=214372>.
- Burkardt, John. 2008. "ASA032 - the Incomplete Gamma Function." [https://people.sc.fsu.edu/~jburkardt/cpp\\_src/asa032/asa032.html](https://people.sc.fsu.edu/~jburkardt/cpp_src/asa032/asa032.html).
- Joe, Stephen, and Frances Y. Kuo. 2003. "Remark on Algorithm 659: Implementing Sobol's Quasirandom Sequence Generator." *ACM Transactions on Mathematical Software (TOMS)* 29 (1): 49–57. <http://dl.acm.org/citation.cfm?id=641879>.
- Johnson, Steven. 2017. "Stevengj/Nlopt: Library for Nonlinear Optimization, Wrapping Many Algorithms for Global and Local, Constrained or Unconstrained, Optimization." Accessed February 20TZ. <https://github.com/stevengj/nlopt>.
- Jones, Phil Mike, Robin Lovelace, and Morgane Dumont. 2016. *rakeR: Easy Spatial Microsimulation (Raking) in R*. <https://CRAN.R-project.org/package=rakeR>.
- Lovelace, Robin, and Dimitris Ballas. 2013. "'Truncate, Replicate, Sample': A Method for Creating Integer Weights for Spatial Microsimulation." *Computers, Environment and Urban Systems* 41 (September): 1–11. <http://linkinghub.elsevier.com/retrieve/pii/S0198971513000240>.
- Lovelace, Robin, and Morgane Dumont. 2016. *Spatial Microsimulation with R*. CRC Press. <http://robinlovelace.net/spatial-microsim-book/>.
- Lovelace, Robin, Dimitris Ballas, Mark M.H. Birkin, Eveline van Leeuwen, Dimitris Ballas, Eveline van Leeuwen, and Mark M.H. Birkin. 2015. "Evaluating the Performance of Iterative Proportional Fitting for Spatial Microsimulation: New Tests for an Established Technique." *Journal of Artificial Societies and Social Simulation* 18 (2): 21. <http://jasss.soc.surrey.ac.uk/18/2/21.html> <https://www.dropbox.com/s/szexnh80exjv3ov/ipfinr-jasss.pdf?dl=0>.
- Matsumoto, Makoto, and Takuji Nishimura. 1998. "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator." *ACM Transactions on Modeling and Computer Simulation* 8 (1): 3–30. doi:10.1145/272991.272995.
- Nowok, Beata, Gillian M. Raab, and Chris Dibben. 2016. "Synthpop: Bespoke Creation of Synthetic Data in R." *Journal of Statistical Software* 74 (11). doi:10.18637/jss.v074.i11.
- Orcutt, Guy H. 1957. "A New Type of Socio-Economic System" *The Review of Economics and Statistics* 39

(2). doi:.

Press, William, Saul Teukolsky, William Vetterling, and Brian Flannery. 2017. "Numerical Recipes in C." Accessed February 20TZ. <http://www.nrbook.com/a/bookcpdf.php>.

Zaloznik, Maja. 2011. "Iterative Proportional Fitting-Theoretical Synthesis and Practical Limitations."