DATA ASSIMILATION WITH AGENT-BASED MODELS USING MARKOV CHAIN SAMPLING

Daniel Tang
Leeds Institute for Data Analytics, University of Leeds, UK*
D.Tang@leeds.ac.uk

Nick Malleson
School of Geography, University of Leeds, UK

April 22, 2022

ABSTRACT

Every day, weather forecasting centres around the world make use of noisy, incomplete observations of the atmosphere to update their weather forecasts. This process is known as data assimilation, data fusion or state estimation and is best expressed as Bayesian inference: given a set of observations, some prior beliefs and a model of the target system, what is the probability distribution of some set of unobserved measures or latent variables at some time, possibly in the future?

While data assimilation has developed rapidly in some areas, relatively little progress has been made in performing data assimilation with agent-based models. This has hampered the use of agent-based models to make quantitative claims about real-world systems.

Here we present an algorithm that uses Markov-Chain-Monte-Carlo methods to generate samples of the parameters and trajectories of an agent-based model over a window of time given a set of possibly noisy, aggregated and incomplete observations of the system. This can be used as-is, or as part of a data assimilation cycle or sequential-MCMC algorithm.

Our algorithm is applicable to time-stepping, agent-based models whose agents have a finite set of states and a finite number of ways of acting on the world. As presented the algorithm is only practical for agents with a few bytes of internal state although we discuss ways of removing this restriction. We demonstrate the algorithm by performing data assimilation with an agent-based, spatial predator-prey model.

Keywords Data assimilation, Bayesian inference, Agent based model, Integer linear programming, predator prey model

1 Introduction

Agent-based models (ABMs) have been widely adopted as an intuitive way to model systems that consist of a heterogeneous collection of autonomous, interacting agents. ABMs are sometimes used to show that an unexpected collective behaviour can emerge from a known agent behaviour. For example Schelling (1971) famously showed that agents with only a very slight preference to live close to agents of their own race can quickly form a highly racially-segregated population. This use of ABMs requires no assimilation of real-world data but generates results that cannot be used to assert anything about the real world without first providing a convincing reason to believe that the model is a proper abstraction of the real-world. In this paper, we'll instead deal with an alternative use case where we have an ABM of some real-world target system and some observations of that same system, and our interest lies in some set of unobserved, real-world measures or latent variables. Our aim will be to combine the information in the observational data with the knowledge encoded in the model in order to make probabilistic assertions about the unobserved measures. This allows us to validate the model against out-of-sample observations as well as generate quantitative forecasts with uncertainty intervals.

*This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 757455)

The observations may contain aggregated or macro-scale measures and may be subject to noise during the measurement process. The unobserved measures may be taken at any time before, during and/or after the observed interval. The behaviour of the agents in the model will generally be stochastic, in order to account for our uncertainty in agent behaviour, and will be a function of some set of unknown parameters, θ, in order to account for our uncertainty over which behavioural model best represents the real-world target entities. Note that the parameters can, and generally should, contain values that control the size of the model stochasticity in order to account for "unknown unknowns". The target system may also interact with a world outside of the model, these interactions necessarily consist of agents being injected into the model at certain times and/or external influences on modelled agents' behaviour[2]. Our prior beliefs about these boundary conditions will also, in general, be probabilistic. The start state of the model can also be treated as a boundary condition consisting of the injection of agents into the model at time $t = 0$. Although the boundary conditions are semantically distinct from the parameters, their mathematical treatment will be identical so we assume θ specifies a set of boundary conditions. The ABM itself contains information and this is contained in a "forecast" distribution, $P(\tau|\theta)$, which is the probability that a set of agents would exhibit behaviours τ on a model execution given parameters/boundary conditions θ. We'll call τ a *model trajectory*.

Our interest will be in some aspect of the posterior distribution $P(\tau, \theta|\Omega)$, where Ω is our set of real-world observations. Any information about $P(\tau, \theta|\Omega)$ can be expressed without loss of generality as the expectation of some, possibly vector-valued, unobserved measure, Λ

$$\mathbb{E}_{P(\tau,\theta|\Omega)}(\Lambda(\tau, \theta)) = \int \Lambda(\tau, \theta)P(\tau, \theta|\Omega)d\tau d\theta. \tag{1}$$

Inserting Bayes' rule gives us

$$\mathbb{E}_{P(\tau,\theta|\Omega)}(\Lambda(\tau, \theta)) = \int \Lambda(\tau, \theta)\frac{P(\Omega|\tau)P(\tau|\theta)P(\theta)}{P(\Omega)}d\tau d\theta \tag{2}$$

where $P(\theta)$ is our prior belief about the parameters and boundary conditions (after accounting for any micro-calibration or other relevant direct observations we may have), $P(\tau|\theta)$ is the ABM's forecast and $P(\Omega|\tau)$ is the observation likelihood, which we assume is easy to calculate given a trajectory that covers the observed period. The prior probability of the observations, $P(\Omega)$, is just a normalising constant so is not usually explicitly calculated when sampling.

There are various approaches in the literature to solving this equation for ABMs. One strategy is to first transform the problem into a simpler form, such as a set of partial differential equations (Lloyd et al., 2016) or a graphical model (Liao & Barooah, 2010), for which there exists a well known method of solving the problem. Tang (2019) expresses the problem in terms of creation and annihilation operators then uses symbolic computation to evaluate the integral in (2). However, the most popular strategy in the literature is to sample from the posterior, $P(\tau, \theta|\Omega)$, and use the samples to approximate the integral using Monte Carlo integration. This is the technique we employ here.

1.1 Importance sampling from the prior

The simplest Monte Carlo approach would be to approximate $\mathbb{E}_{P(\tau,\theta|\Omega)}(\Lambda(\tau, \theta))$ using importance sampling as follows:

1. Take a sample from the prior parameters/boundary conditions $\theta_i \sim P(\theta)$.
2. Execute the model forward with the given θ_i to generate a sample trajectory $\tau_i \sim P(\tau|\theta_i)$.
3. Calculate a weight $w_i = P(\Omega|\tau_i)$ to create a weighted sample $\langle \tau_i, \theta_i, w_i \rangle$.
4. Repeat N times from step 1 to get a set of weighted samples $\{\langle \tau_1, \theta_1, w_1 \rangle \dots \langle \tau_N, \theta_N, w_N \rangle\}$.
5. Approximate $\mathbb{E}_{P(\tau,\theta|\Omega)}(\Lambda(\tau, \theta)) \approx \frac{1}{\sum_{j=1}^{N} w_j} \sum_{i=1}^{N} w_i \Lambda(\tau_i, \theta_i)$.

However, Chatterjee & Diaconis (2018) show that the information contained in the weighted samples reduces exponentially with the KL-divergence from the prior to the posterior, $D_{KL}\left(P(\tau, \theta|\Omega) \;||\; P(\tau|\theta)P(\theta)\right)$. This divergence can be thought of as a measure of the information contained in the observations Ω. For reasons we'll see later, in the case of ABMs, the observations usually contain enough information to make the set of weighted samples less informative than a single unweighted sample, even when N is astronomically high. So this algorithm will not work unless the observations contain very little information.

1.2 Particle filtering

In the literature, the most popular way of solving (2) for ABMs is to use *particle filtering*, also known as *sequential Monte Carlo*. This splits the time period of interest into smaller, contiguous time windows with (not necessarily equidistant) end times $\langle t_1 \dots t_N \rangle$ where $t_m < t_{m+1}$. The trajectory and observations are similarly split based on which window they occur in, $\tau = \langle \tau_1 \dots \tau_N \rangle$ and $\Omega = \langle \Omega_1 \dots \Omega_N \rangle$. The posterior distribution of the whole trajectory can

[2] we take the purist view that an ABM models everything as an agent

then be split into the product of contributions from each window given the distribution over the previous window, leading to a recursion relation

$$P\left(\tau_{1:t+1}, \theta | \Omega_{1:t+1}\right) = \frac{P(\Omega_{t+1}|\tau_{t+1}) P(\tau_{t+1}|\tau_{1:t}, \theta) P\left(\tau_{1:t}, \theta | \Omega_{1:t}\right)}{P(\Omega_{t+1}|\Omega_{1:t})} \tag{3}$$

where $\tau_{1:t} = \langle \tau_1 \dots \tau_t \rangle$ and $P(\tau_{t+1}|\tau_{1:t}, \theta)$ is the probability that an execution of the ABM that starts with trajectory $\tau_{1:t}$ would go on to produce τ_{t+1} given the parameters/boundary conditions θ. The final posterior $P(\tau_{1:N}, \theta|\Omega_{1:N})$ can now be built up in steps, starting with the first window and moving to the last, using the recursion as we go and approximating each $P(\tau_{1:t}, \theta|\Omega_{1:t})$ as a set of samples, or "particles". This makes the problem easier in two respects, firstly the dimension of the distributions we need to deal with at each recursion step are reduced since, if we consider each particle separately, we're only concerned with the trajectory in one window at a time, secondly we split the information in the observations into smaller chunks, so at each recursion the information in Ω_t is more likely to be small enough to make it practical to perform importance sampling from the prior.

If the unobserved measure of interest, Λ, depends only on the states of the agents, σ_N, at time t_N and/or the parameters/boundary conditions, θ, rather than on the full trajectory, $\tau_{1:N}$, then we can reduce the dimensionality even further by performing the recursion on model states rather than trajectories (where a model state tells us how many agents there are in each agent state). First arrange the windows so that each observation lies at the end of a window, then marginalise equation (3) over $\tau_{1:t+1}$ for a fixed σ_{t+1} to get

$$P\left(\sigma_{t+1}, \theta | \Omega_{1:t+1}\right) = \frac{P(\Omega_{t+1}|\sigma_{t+1})}{P(\Omega_{t+1}|\Omega_{1:t})} \int P(\sigma_{t+1}|\sigma_t, \theta) P\left(\sigma_t, \theta | \Omega_{1:t}\right) d\sigma_t \tag{4}$$

where we've assumed that the observation likelihoods depend only on the model state and made explicit in $P(\sigma_{t+1}|\sigma_t, \theta)$ that the probability of the state σ_{t+1} depends only on σ_t and θ [3]. Note that the definition of "model state" can always be re-defined in such a way as to make these assumptions true. In fact, we can see that equation (3) can be thought of as a special case of equation (4) where the model state, σ_t, is the trajectory, $\tau_{1:t}$, and the integral over σ_t is non-zero for only a single trajectory, since $P(\tau'_{1:t+1}|\tau_{1:t}, \theta) = \delta_{\tau_{1:t}}(\tau'_{1:t}) P(\tau'_{t+1}|\tau_{1:t}, \theta)$, where δ here is the multivariate delta distribution.

So, we can consider a recursion on *generalised trajectories*, X_t,

$$P\left(X_{t+1} | \Omega_{1:t+1}\right) = \frac{P(\Omega_{t+1}|X_{t+1})}{P(\Omega_{t+1}|\Omega_{1:t})} \int P(X_{t+1}|X_t) P\left(X_t | \Omega_{1:t}\right) dX_t \tag{5}$$

so that equations (3) and (4) would correspond to $X_t = \langle \tau_{1:t}, \theta \rangle$ and $X_t = \langle \sigma_t, \theta \rangle$ respectively.

A particle filter solves this recursion by taking a set of samples $X_t^{1:R}$ drawn from $P(X_t|\Omega_{1:t})$, and generating a set of samples $X_{t+1}^{1:R}$ drawn from $P(X_{t+1}|\Omega_{1:t+1})$. The simplest algorithm to do this is known as sequential importance resampling which consists of the following steps:

1. Generate a set of samples $X_0^{1:R}$ from the prior $P(X_0)$. Set $t = 0$.
2. For each sample X_t^i, generate a sample from the forecast $\hat{X}_{t+1}^i \sim P(X_{t+1}|X_t^i)$ by executing the ABM from the end state of X_t^i to give a set of forecast samples $\hat{X}_{t+1}^{1:R}$.
3. Calculate a weight for each forecast sample $\hat{X}_{t+1}^i$

$$w_i = \frac{P(\Omega_{t+1}|\hat{X}_{t+1}^i)}{\sum_{j=1}^R P(\Omega_{t+1}|\hat{X}_{t+1}^j)}.$$

4. Resample from the weighted samples by taking R samples from the approximation

$$P(X_{t+1}|\Omega_{1:t+1}) \approx \sum_i w_i \delta_{\hat{X}_{t+1}^i}(X_{t+1}) \tag{6}$$

   to give a new set of unweighted samples $X_{t+1}^{1:R}$. There are a few ways of performing this resampling (Douc & Cappé, 2005), the simplest being to draw an integer $i \in [1, R]$ with probability w_i, returning the sample $\hat{X}_{t+1}^i$ and repeating R times.
5. If $t < N$, increment t and repeat from step 2.

However, this algorithm suffers from the problem of *sample impoverishment* (Li et al., 2014) or *sample deprivation* which describes the situation when the resampling step leaves many particles in the same state, so the number of distinct samples in our sample set is much smaller than the total number of samples. This is because, as we've seen, after each

[3] technically, if θ contains time dependent boundary conditions, the model states need to be timestamped so that the model knows which boundary conditions to apply

importance sampling step the information in the observations causes the effective sample size to decrease and so the resampling step is likely to generate repeated samples. This loss of effective sampling size can accumulate and in practice as the number of windows increases, the number of particles required to maintain a desired accuracy increases exponentially. In the worst case, we encounter an observation that is impossible for all our samples and we end up with an effective sample size of zero. Malleson et al. (2020) show that sample impoverishment is an issue when applied to data assimilation with an ABM of crowd movement. Khan et al. (2003) encounter the same problem in a model of ant movement.

To get a better understanding of impoverishment, suppose we use the above algorithm to generate a set of samples $\left\{\left\langle \tau^1_{1:N}, \theta^1 \right\rangle \dots \left\langle \tau^S_{1:N}, \theta^S \right\rangle\right\}$ of the full trajectory $P(\tau_{1:N}, \theta|\Omega_{1:N})$, from which we generate marginalisations over each window $W_t = \left\{\left\langle \tau^1_t, \theta^1 \right\rangle \dots \left\langle \tau^S_t, \theta^S \right\rangle\right\}$. On the one hand, the repeated impoverishment at each step causes the number of distinct values in W_{N-L} to decrease exponentially as the lag, L, increases (i.e. as we look further back in time), meaning we lose information about the true distribution of $P(\tau_{N-L}, \theta|\Omega_{1:N})$. However, on the other hand, the model dynamics and observations make $P(\tau_N, \theta|\Omega_{1:N})$ increasingly independent of $P(\tau_{N-L}, \theta|\Omega_{1:N})$ as L increases. So if all we care about is the final window (which is often the case) impoverishment only becomes a problem when the first effect (the rate of loss of information) dominates the second (the rate of increase in independence from the past).

Unfortunately every window's trajectory depends on the parameters, θ, but these are sampled in the first window and never subsequently change, so after assimilating only a few windows all samples are likely to have collapsed to the same value of θ, giving us little information about the true distribution of the marginalised posterior $P(\theta|\Omega_{1:N})$. This is fine if the parameters are known at $t = 0$, but if there is prior uncertainty in θ and the observations don't reduce that uncertainty faster than the collapse in information contained in the samples, then impoverishment will be a problem (Liu & West, 2001; Andrieu, Doucet, Singh, & Tadic, 2004).

A simple way of dealing with sample impoverishment is to add noise to the particles at each step. This is known as roughening (Gordon, Salmond, & Smith, 1993; Li, Sun, Sattar, & Corchado, 2014) and is equivalent to replacing the delta function in equation (6) with some other function with wider support so we end up with a weighted kernel density estimation. Kieu et al. (2020) uses this technique in the context of an ABM of public transport. However, Liu & West (2001) shows that this leads to over-dispersion unless we "shrink" the samples towards their mean value. More generally, the consequence of roughening is that a finite number of samples from a roughened particle filter is no longer a draw from the true posterior, when averaged over all possible draws of the same size, so we need to convince ourselves that the error introduced by the roughening process is small enough for our requirements; not always an easy task for an ABM where distributions can be highly discontinuous.

Wang & Hu (2015) deals with sample impoverishment in the context of ABM by using the samples at time t to approximate the marginal probabilities of the state of each individual agent at the start of the next window, $P(a_j)$, then drawing new samples from the product of the marginals $P(a_1 \dots a_n) = \prod_{j=1}^{N} P(a_j)$, i.e. sampling the state of the j^{th} agent from the marginal $P(a_j)$. This improves sample diversity at the price of losing all correlations between agents.

In the context of ABMs, sample impoverishment can manifest in its most extreme form by generating particles that have zero weight and so contribute no information to the resampling step. This happens because a draw from the forecast of an ABM $P(X_{t+1}|X_t)$ often has an extremely high probability of having a likelihood,$P(\Omega_t|X_{t+1})$ of zero. For example, suppose we're tracking mobile phones by IMEI and receive intermittent signals from the phones identifying which cell they're in. Suppose for simplicity that if a phone is in a known cell at time t then it could be in any of 8 cells at time $t + 1$, each with equal prior probability. Suppose at time $t + 1$ we get signals from 10 phones. Even if we know the correct model state at time t, the probability that a sample from the forecast at time $t + 1$ has a non-zero likelihood is 8^{-10}, about one in a billion. So sequential importance resampling will fail in a single step in this case.

1.3 Kalman filtering

One potential solution to the problem of impoverishment is to assume that the distributions are Gaussian. This is the basis of the data assimilation algorithms that have had a great deal of success in geophysical models (Carrassi, Bocquet, Bertino, & Evensen, 2018; Talagrand, 1997; Kalnay, 2003; Lewis, Lakshmivarahan, & Dhall, 2006). If the forecast $P(X_{t+1}|\Omega_{1:t})$ and likelihood function $P(\Omega_{t+1}|X_{t+1})$ are multivariate Gaussians, the recursion in (5) can be solved analytically and the recursion can be evaluated even when the observations Ω_t are highly informative. This idea leads to the *Ensemble Kalman filter* (Evensen, 2003) and the *Unscented Kalman filter* (Wan et al., 2001), both of which have been applied to data assimilation in ABM (Ward, Evans, & Malleson, 2016; Clay, Kieu, Ward, Heppenstall, & Malleson, 2020).

However, the Gaussian assumption is not usually a good one for ABMs. For example, suppose there is a sports hall containing N people and we take the model state to be the $2N$ dimensional vector consisting of the (x, y) coordinates of each person. Even if the initial state is distributed according to a $2N$ dimensional Gaussian distribution, the either/or decisions of the people can quickly transform this into a multimodal distribution. Suppose two exits are opened on opposite sides of the hall and each person moves towards their nearest exit. Each person makes an either/or decision which exit to move towards based on their position, so we can think of the prior Gaussian as being partitioned into 2^N regions corresponding to all possible sets of decisions made by the people. In the forecast distribution, the probability

mass in each region will move towards a different model state (one of the 2^N states where all agents are at one or other of the exits), creating a highly non-Gaussian distribution with 2^N modes. In addition, the likelihood function is often highly non-Gaussian in this state space. For example, suppose there is a sensor at one of the exits that trips if any agents are close. Suppose for example that there are just two agents so the model state space is (x_1, y_1, x_2, y_2) and the origin of the agents' coordinate system is at the sensor. If the sensor trips, the likelihood function is zero everywhere except close to the perpendicular planes $(0, 0, x_2, y_2)$ and $(x_1, y_1, 0, 0)$. This clearly isn't well approximated by any Gaussian. This is a consequence of the fact that we don't know which agent caused the sensor to trip, a problem known as the *data association problem* (Lueck et al., 2019). This problem becomes superexponentially worse if we have multiple sensors. If there are N agents and M sensors the likelihood function is the union of $\frac{N!}{(N-M)!}$ hyperplanes.

An alternative representation of the model state space is the *occupation number* representation where each agent state becomes a dimension in the state space and the coordinate values give the number of agents in each state. In this representation Gaussians may be appropriate in some models if there are a large number of agents in every state. However, as the occupation numbers get smaller, an increasing proportion of a Gaussian approximation lies on states that contain negative occupation numbers, which is problematic as a negative occupation number has no natural interpretation. In very high dimensions, the vast majority of the probability mass will lie in a region with at least one negative occupation number. Likelihood functions in this space can also be highly non-Gaussian. For example, the sensor in (a discretised version of) the sports hall example above would have a likelihood function that is a step function on the dimension that corresponds to the state close to the door.

1.4 Metropolis-Hastings

In the following sections we present a solution to these problems by using the Metropolis-Hastings algorithm to construct a Markov process whose stationary state is $P(\tau_{1:N}, \theta|\Omega_{1:N})$. To our knowledge this is the first time this has been done in a way that is applicable to a wide range of ABMs and observations. For small N we can sample directly from this Markov process to get samples from the posterior. For larger N, or if we wish to perform *online* data assimilation where the observations form a continuous, effectively endless stream, we propose using the Markov transition function as part of a Sequential MCMC or MCMC Particle Filter algorithm (Finke, Doucet, & Johansen, 2020; Septier, Pang, Carmi, & Godsill, 2009) which solves the problem of impoverishment by combining MCMC steps with particle filtering. There are many specific ways of doing this, unfortunately the most appropriate algorithm to use depends on the nature of the ABM dynamics, the likelihood function and the unobserved measure, Λ, so there's no silver bullet. In section 6 we give some guidance by presenting some algorithms that are likely to be appropriate in the context of ABMs.

Since the nature of an ABM's parameters is highly model-specific, in the following we focus on the treatment of the trajectory but in a way that is fully integrated with a model-specific treatment of the parameters for which we give more general design advice. In the case of state estimation (where the parameters are already known) the parameter specific part can be left out completely.

2 Formulation of the problem

2.1 Definition of an ABM

Suppose we have a timestepping ABM that consists of agents with a finite number of possible internal states and a finite number of mutually exclusive ways of acting on their world. Given this, we formally define an ABM as:

- An ordered list of agent states $\mathcal{S} = \langle \sigma_0 ... \sigma_{S-1} \rangle$.
- An ordered list of agent actions $\mathcal{A} = \langle \alpha_0 ... \alpha_{A-1} \rangle$.
- An *agent timestep*, $\pi : \mathbb{Z} \times \mathbb{Z}^S \times \mathbb{Z} \to \mathbb{R}$, which defines the probability that an agent will act in a particular way such that $\pi(\psi, \Psi, a)$ gives the probability that an agent in state σ_ψ, surrounded by agents, Ψ, will perform action α_a (where Ψ is an S dimensional vector whose i^{th} element is the number of agents in state σ_i at the start of the timestep).
- An *action function*, $F : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}^S$, which defines the effect of an action on the world such that $F(\psi, a)$ is an S dimensional vector whose i^{th} element gives the number of agents in state σ_i that result from an agent in state σ_ψ performing act α_a (including the final state of the acting agent).

As a simple illustration, we define the "cat and mouse" ABM as follows:

Agent states An agent can be either a cat or a mouse and can be on one of two grid-squares, left or right, so

$$\mathcal{S} = \langle \text{left-cat}, \text{right-cat}, \text{left-mouse}, \text{right-mouse} \rangle .$$

Agent actions In any given timestep, an agent can either move to the other grid-square or stay still, so

$$\mathcal{A} = \langle \text{move}, \text{stay still} \rangle .$$

Agent timestep In a timestep, a cat will move or stay still with probability 0.5, a mouse will move if there are any cats on the same gridsquare or stay still otherwise. This behaviour can be expressed as the agent timestep:

$$\pi(\psi, \Psi, a) = \begin{cases} 0.5 & \text{if } \psi \in \{0, 1\} \\ 1 & \text{if } (\psi = 2, \Psi_1 = 0, a = 1) \text{ or } (\psi = 3, \Psi_2 = 0, a = 1) \\ 1 & \text{if } (\psi = 2, \Psi_1 > 0, a = 0) \text{ or } (\psi = 3, \Psi_2 > 0, a = 0) \\ 0 & \text{otherwise.} \end{cases}$$

.

Action function This gives the result of an agent's actions. For example, $F(1, 0)$ is the result of an agent in state $\psi = 1$ (*right cat*) performing the action $a = 0$ (*move)*, which is one cat in state $\psi = 0$ (*left cat*). So $F(1, 0) = \{1, 0, 0, 0\}$. This model is simple enough that we can write down F explicitly for all ψ and a:

$$\begin{aligned} F(0, 0) &= \{0, 1, 0, 0\} \\ F(1, 0) &= \{1, 0, 0, 0\} \\ F(2, 0) &= \{0, 0, 0, 1\} \\ F(3, 0) &= \{0, 0, 1, 0\} \\ F(0, 1) &= \{1, 0, 0, 0\} \\ F(1, 1) &= \{0, 1, 0, 0\} \\ F(2, 1) &= \{0, 0, 1, 0\} \\ F(3, 1) &= \{0, 0, 0, 1\}. \end{aligned}$$

2.2 A note on tensor notation

In the rest of this paper we'll make use of multidimensional arrays. These are just arrays of numbers, like vectors or matrices, but arranged in any number of dimensions. Each array has a *shape* which tells us how many dimensions it has and the number of elements in each dimension. For notational convenience, we'll distinguish between subscript dimensions and superscript dimensions[4] and will refer to the set of all arrays of a given shape with the symbol $\mathbb{R}$ adorned with the size of each subscript and superscript dimension. So, for example, $\mathbb{R}^N_{SA}$ describes the set of all 3-dimensional arrays that have one superscript dimension of size N and two subscript dimensions of sizes S and A.

An element of an array is referred to by specifying the sub- and superscript coordinates of the element. For example if $T \in \mathbb{R}^N_{SA}$, then $T^t_{\psi a}$ refers to the element of T that has coordinate t in the superscript dimension and coordinates (ψ, a) in the subscript dimensions. By convention, coordinates begin at 0.

We'll also borrow from tensor notation by using the Einstein summation convention, meaning that if the same index symbol is repeated in sub- and superscript positions in a term, then a summation over that dimension is implied. So, for example if $X \in \mathbb{R}_8$ and $Y \in \mathbb{R}^8$ then

$$X_\psi Y^\psi \equiv \sum_{\psi=0}^{7} X_\psi Y^\psi.$$

When the same symbol is repeated in the *same* position with no implied summation, this implies universal quantification. For example if $Y \in R_8$ then

$$X_\psi = Y_\psi$$

is equivalent to

$$\forall (0 \leq \psi < 8) : X_\psi = Y_\psi.$$

We refer to *slices* of an array by using the $*$ symbol in an index position. For example if $T \in \mathbb{R}^N_{SA}$ then $T^t_{\psi *}$ refers to the 1-dimensional array in $\mathbb{R}_A$ comprised of the elements $\left\langle T^t_{\psi 0} ... T^t_{\psi A-1} \right\rangle$.

The symbols $\mathbf{0}$ and $\mathbf{1}$ represent arrays whose elements are all 0 or 1 respectively. Their shape should be unambiguous from the context.

2.3 Trajectories

Let a model timestep be an array $E \in \mathbb{R}_{SA}$ whose elements $e_{\psi a}$ give the number of agents in state σ_ψ that perform act α_a in this timestep. For example, the timestep shown in Figure 1 for the cat and mouse example would be

$$E = \begin{array}{c} \\ \sigma_0 \\ \sigma_1 \\ \sigma_2 \\ \sigma_3 \end{array} \begin{array}{c} \begin{array}{cc} \alpha_0 & \alpha_1 \end{array} \\ \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \end{array}$$

[4] covariant and contravariant dimensions if you prefer

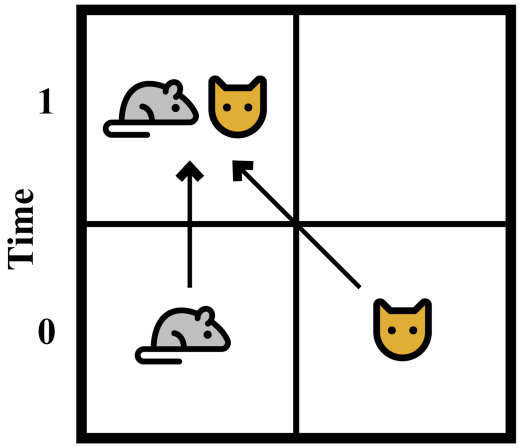

Figure 1: A simple cat and mouse model.

where one agent in state σ_1 (*right cat*) performs action α_0 (*move*) and one agent in state σ_2 (*left mouse*) performs action α_1 (*stay still*).

Let a model trajectory, T, be an array in $\mathbb{R}^N_{SA}$ that represents N timesteps of a model with S agent states and A actions, so that $T^t_{\psi a}$ denotes the $(\psi, a)^{th}$ element of the t^{th} timestep matrix.

An array must satisfy a number of constraints in order to be a valid trajectory of an ABM. Since the elements of a trajectory are counts of agents, they must be non-negative integers. We'll denote the set of all non-negative integer arrays

$$\mathbb{N}^N_{SA} = \left\{T \in \mathbb{R}^{\mathbb{N}}_{\mathbb{SA}} : T^t_{\psi a} \geq \mathbf{0}^t_{\psi a}, T^t_{\psi a} \in \mathbb{Z}\right\}. \tag{7}$$

A trajectory must also be *continuous* by which we mean that any agent that is the result of an action at timestep $t-1$ must be the cause of an action at timestep t, and any agents injected by the boundary conditions at timestep t must also act in timestep t[5]. We call this the *continuity constraint* and define the set of continuous arrays, with respect to an action function F:

$$\mathcal{C}^N_{SA}(F) = \left\{T \in \mathbb{R}^N_{SA} : I^0_\phi = T^0_{\phi b}\mathbf{1}^b, \forall\,(1 \leq t < N) : F^{\psi a}_\phi T^{t-1}_{\psi a} + I^t_\phi = T^t_{\phi b}\mathbf{1}^b\right\} \tag{8}$$

where $F \in \mathbb{R}^{SA}_S$ is the action array $F^{\psi a}_* = F(\psi, a)$ and I^t_ϕ is the number of boundary condition injections into state ϕ at time t. Note that this assumes that a timestep is performed by updating all agents in parallel, i.e. agents must act with no information about the actions of other agents in the same timestep. This is different from serial update, where agents are updated in a particular order within a timestep and have access to information about the actions of agents that come before them in the ordering. Note also that this assumes that agents in the same state are indistinguishable, i.e. if we swap two agents in the same state, the trajectory is unchanged.

So, we define the set of trajectories, $\mathcal{T}^N_{SA}(F)$, as set of arrays that satisfy (7) and (8).

$$\mathcal{T}^N_{SA}(F) = \mathbb{N}^N_{SA} \cap \mathcal{C}^N_{SA}(F). \tag{9}$$

2.4 The posterior of an ABM

Our goal is to generate samples from the Bayesian posterior over the trajectories, boundary conditions and model parameters, given a set of observations, Ω

$$P\left(T, \theta|\Omega\right) \propto P\left(\Omega|T\right) P(T|\theta)P(\theta).$$

The model forecast, $P(T|\theta)$, can be decomposed into the product of conditionals for each timestep and agent state

$$P\left(T|\theta\right) \propto P(\Psi^0_*|\theta) \prod_{t,\psi} P(T^t_{\psi *}|\Psi^t_*, \psi, \theta)$$

where $\Psi^t_* \in \mathbb{N}_S$ is the model state at the start of timestep t and $P(\Psi^0_*|\theta)$ is defined by the boundary conditions[6]. The term $P(T^t_{\psi *}|\Psi^t_*, \psi, \theta)$ is the probability that Ψ^t_ψ agents starting in state ψ will collectively perform the vector of actions

[5]This does not mean that agents cannot leave or enter the system, only that if they do then that change must be the result of an action or boundary condition.

[6]for notational clarity we assume that all injections are at time $t = 0$, but this can easily be extended to injections at any or all timesteps.

$T^t_{\psi *}$. The probability that a single agent performs action a is given by the agent timestep $\pi_\theta(\psi, \Psi^t, a)$ (where we've added the subscript θ to emphasise its possible dependence on the parameters). Since agent actions are simultaneous, the collective behaviour is given by the multinomial distribution

$$P\left(T^t_{\psi *} \mid \Psi^t_*, \psi, \theta\right) = \begin{cases} \Psi^t_\psi ! \prod_a \frac{\pi_\theta(\psi, \Psi^t_*, a)^{T^t_{\psi a}}}{T^t_{\psi a}!} & \text{if } T^t_{\psi a}\mathbf{1}^a = \Psi^t_\psi \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

If the trajectory is continuous then $\Psi^t_* = T^t_{*a}\mathbf{1}^a$ so

$$P(T|\theta) = \begin{cases} P(\Psi^0_* = T^0_{*c}\mathbf{1}^c|\theta) \prod_{t,\psi} \left(T^t_{\psi b}\mathbf{1}^b\right)! \prod_a \frac{\pi_\theta(\psi, T^t_{*d}\mathbf{1}^d, a)^{T^t_{\psi a}}}{T^t_{\psi a}!} & \text{if } T \in \mathcal{T}^N_{SA}(F) \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

The likelihood function, $P(\Omega|T)$, can also usually be decomposed. Without loss of generality, we take Ω to consist of some number of observations that are independent of each other given the trajectory, so that Ω is a set of pairs (ω, v) that consist of a stochastic observation operator ω and an observed value v (which may be a vector). We write $P(\omega(T) = v)$ to denote the probability of observation operator ω making observation v on trajectory T. So

$$P(\Omega|T) = \prod_{(\omega,v)\in\Omega} P(\omega(T) = v)$$

and the posterior can be written in terms of the agent timestep function as

$$P(T, \theta|\Omega) \propto \begin{cases} P(\theta)P(\Psi^0_* = T^0_{*c}\mathbf{1}^c|\theta) \prod_{(\omega,v)\in\Omega} P\left(\omega(T) = v\right) \prod_{t,\psi} \left(T^t_{\psi b}\mathbf{1}^b\right)! \prod_a \frac{\pi_\theta(\psi, T^t_{*d}\mathbf{1}^d, a)^{T^t_{\psi a}}}{T^t_{\psi a}!} & \text{if } T \in \mathcal{T}^N_{SA}(F) \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

3 Approximating the support of the posterior

As we saw in the introduction, sampling from equation (12) is difficult because it is often very hard to find proposal trajectories that have non-zero probability. Our strategy to solve this problem is to first derive an expression for a volume of trajectory/parameter space that bounds the support of the posterior, $\mathrm{supp}(P(T, \theta|\Omega))$ (i.e. the set of $\langle T, \theta\rangle$ pairs that have non-zero posterior probability). If we choose a bounding volume that it is easy to sample from and is a relatively tight approximation of $\mathrm{supp}(P(T|\Omega)))$ then there's a good chance that a sample drawn from this volume will have non-zero probability. The bounding volume doesn't need to be exact, it just needs to be tight enough to give us a good chance of finding a non-zero probability sample.

From equation (12)

$$\begin{aligned} \mathrm{supp}(P(T, \theta|\Omega)) = & \bigcap_{(\omega,v)\in\Omega,t,\psi,a} \mathcal{T}^N_{SA} \cap \mathrm{supp}(P(\theta)) \\ & \mathrm{supp}(P(\Psi^0_* = T^0_{*c}\mathbf{1}^c)|\theta) \cap \\ & \mathrm{supp}\left(P\left(\omega(T) = v\right)\right) \cap \\ & \left(\mathrm{supp}\left(\pi_\theta(\psi, T^t_{*b}\mathbf{1}^b, a)\right) \cup \left\{T : T^t_{\psi a} = 0\right\}\right) \end{aligned} \tag{13}$$

i.e. in order for $\langle T, \theta\rangle$ to have non-zero posterior probability, θ must have non-zero prior probability and T must be a trajectory of the ABM that has a possible start state, all observations must have non-zero likelihood and each non-zero element of T must denote an agent action with non-zero probability.

3.1 Convex mixed-integer polyhedra

We now approximate the support in equation (13) with a *convex, mixed-integer polyhedron*, $\mathcal{P}$, (see, for example, Conforti et al. (2010)) which we define to be a set of vectors, some of whose elements are integer and some real-valued, that satisfy a set of linear inequalities:

$$\mathcal{P} = \left\{X \in \mathbb{Z}^N \times \mathbb{R}^M : L \leq CX \leq U\right\}$$

for some matrix C and some vectors L and U .

Any linear inequality on T and θ can be expressed in this form by letting X consist of the elements of T and θ flattened into one dimension so that

$$X^i = W^{\psi a i}_t T^t_{\psi a} + G^i_j \theta^j$$

and W and G have inverses

$$\hat{W}^t_{\psi a i} = W^{\psi' a' i'}_{t'} \delta_{\psi'\psi}\delta_{a'a}\delta^{t't}\delta_{i'i}$$

and

$$\hat{G}_i^j = G_{j'}^{i'} \delta_{i'i} \delta^{j'j}$$

such that

$$T_{\psi a}^t = \hat{W}_{\psi ai}^t X^i$$

and

$$\theta^j = \hat{G}_i^j X^i$$

where δ^{ij} and δ_{ij} are 1 if $i = j$ and 0 otherwise. In this way we can switch unambiguously between X and $\langle T, \theta \rangle$.

We now consider how to express each term in equation (13) as a mixed integer polyhedron. From equation (9) we can see immediately that the set of all trajectories, $\mathcal{T}_{SA}^N$, is is already defined as a set of linear constraints on T. The support of the prior $P(\theta)$ can usually be easily bounded by a hyper-rectangle giving the range of each variable. The supports of the start state, $P(\Psi_*^0|\theta)$, the observations, $P(\omega(T) = v)$, and the agent actions, $\pi_\theta(\psi, T_{*b}^t \mathbf{1}^b, a)$, can often be easily expressed as linear constraints. However, if this is not the case, each of the probability distributions can be expressed as a computer program. In practice, these computer programs will be simple and it will be possible to use a technique known as *abstract interpretation* (Cousot & Cousot, 1977) using the domain of convex polyhedra (Cousot & Halbwachs, 1978; Becchi & Zaffanella, 2018; Fukuda, 2020) to efficiently calculate a convex polyhedron that bounds the support of the program. Software to perform abstract interpretation using convex polyhedra already exists (Henry, Monniaux, & Moy, 2012; Gurfinkel & Navas, 2021; Jeannet & Miné, 2009; Bagnara, Hill, & Zaffanella, 2008) and the technique has been used in applications such as compiler optimization (Sjödin et al., 2009) and verification of safety-critical systems (Halbwachs et al., 1997). The programs may contain calls to a random number generator `Random()` that returns a random floating-point number $0 \leq r < 1$. This can be represented in the polyhedral domain by introducing an real valued auxiliary variable, r, that satisfies $0 \leq r < 1$ for each call to `Random()`. These auxiliary variables can be removed immediately using the abstract interpretation software to project the polyhedron back into the space of X. This relies on converting the polyhedron to its set of vertices (Motzkin et al., 1953) and projecting the vertices, so the number of vertices must be of a tractable size. Alternatively, the auxiliary variables can be added to X and marginalised out when we take the samples.

If the number of agents is very much smaller than the number of agent states (which is often the case with agent-based models) then we may be willing to make the assumption that in any timestep there is at most one agent performing a given action from a given start state (i.e. $\forall \psi, a, t : T_t^{\psi a} \in \{0, 1\}$). Under this assumption, which we'll call the *Fermionic assumption*, the *Fermionic trajectories*, $\mathcal{F}_{SA}^N = \left\{ T \in \mathcal{T}_{SA}^N : \forall \psi, a, t : T_t^{\psi a} \in \{0, 1\} \right\}$, all lie on the corners of the unit hypercube. So the intersection of $\mathcal{F}_{SA}^N$ with any set is a convex polyhedron and $\mathrm{supp}(P(T|\Omega))$ can always be exactly represented as a mixed-integer polyhedron (finding that polyhedron isn't always tractable, though, in which case abstract interpretation software can be used to find a larger polyhedron that contains it).

So, if we let $\mathcal{P}(f)$ be a mixed integer polyhedron that contains $\mathrm{supp}(f)$ then from equation (13) we can approximate the support of the posterior as

$$\begin{aligned} \mathrm{supp}(P(T, \theta|\Omega)) \subseteq \mathcal{P}(P(W_t^{\psi ai} T_{\psi a}^t + G_j^i \theta^j | \Omega)) = & \bigcap_{(\omega, v) \in \Omega, t, \psi, a} \mathcal{T}_{SA}^N \cap \mathcal{P}(P(\theta)) \\ & \mathcal{P}(P(\Psi_*^0 = T_{*c}^0 \mathbf{1}^c | \theta)) \cap \\ & \mathcal{P}\left(P\left(\omega(T) = v\right)\right) \cap \\ & \left(\mathcal{P}\left(\pi_\theta(\psi, T_{*b}^t \mathbf{1}^b, a)\right) \cup \left\{T : T_{\psi a}^t = 0\right\}\right). \end{aligned} \tag{14}$$

The intersection of two mixed-integer polyhedra is easy to express as another mixed integer polyhedron by just concatenating the constraints

$$\begin{aligned} \left\{X \in \mathbb{Z}^N \times \mathbb{R}^M : L \leq CX \leq U\right\} \cap \left\{X \in \mathbb{Z}^N \times \mathbb{R}^M : L' \leq C'X \leq U'\right\} & \\ = \left\{X \in \mathbb{Z}^N \times \mathbb{R}^M : \begin{pmatrix} L \\ L' \end{pmatrix} \leq \begin{pmatrix} C \\ C' \end{pmatrix} X \leq \begin{pmatrix} U \\ U' \end{pmatrix}\right\} & \end{aligned} \tag{15}$$

so the only difficulty in calculating $\mathcal{P}(P(X|\Omega))$ is the union in the final term of equation (14). We transform this into an intersection by introducing an auxiliary variable z and using the following identity: if the hyper-rectangle $0 \leq X \leq H$ bounds $L \leq CX \leq U$ for some vector $H \in \mathbb{Z}^N \times \mathbb{R}^M$, then for any given i

$$\begin{aligned} \left\{X \in \mathbb{Z}^N \times \mathbb{R}^M : L \leq CX \leq U\right\} \cup \left\{X : X^i = 0, \mathbf{0} \leq X \leq H\right\} = \{\ & X \in \mathbb{Z}^N \times \mathbb{R}^M, z \in \{0, 1\} : \\ & CX + (\overline{B} - U)z \leq \overline{B}, \\ & \underline{B} \leq CX + (\underline{B} - L)z, \\ & 0 \leq H^i z - X^i, \\ & z - X^i \leq 0\ \} \end{aligned} \tag{16}$$

where $\overline{B}$ are upper bounds on the values of CX in the hyper-rectangle, defined as

$$\overline{B} = \frac{(C + |C|)\, H}{2}$$

where $|C|$ here denotes an element-wise absolute value of a matrix and $\underline{B}$ are lower bounds on the values of CX defined as

$$\underline{B} = \frac{(C - |C|)\, H}{2}.$$

To see why this identity holds, note first that the constraints make z into an indicator variable that is 0 if $X^i = 0$ or 1 otherwise. When $z = 1$ the first constraint on the right hand side of (16) is equal to $CX \leq U$ and the second is equal to $L \leq CX$ so their intersection is $L \leq CX \leq U$ as required, whereas when $z = 0$ the constraints are equal to $\underline{B} \leq CX \leq \overline{B}$. But $\underline{B}$ and $\overline{B}$ are lower and upper bounds on the values of CX in the hyper-rectangle so this is satisfied for all values $0 \leq X \leq H$ as required.

There are two things worth noting here. Firstly if we make the Fermionic assumption then $z = X^i = T^t_{\psi a}$ so the auxiliary indicator variable becomes unnecessary. Secondly, we must bound $\mathcal{P}(\pi_\theta(\psi, T^t_{*b}\mathbf{1}^b, a))$ by a hyper-rectangle with its lower corner at the origin. In practice, this is not a problem as we can just make a change of variable, imposing upper and lower bounds on each variable such that the probability of being outside those bounds is negligible, then shifting if the lower bound isn't on zero.

So, the support of the posterior can be approximated by a mixed-integer polyhedron, $\mathcal{P}(P(X|\Omega))$, by using equations (14), (15) and (16).

As a simple illustration, consider a two-timestep trajectory of the cat and mouse model described in section 2.1. Suppose we flip a fair coin to decide whether each agent state is occupied or empty at $t = 0$. Suppose also that we observe a cat in the left grid-square at time $t = 1$. Our aim is to construct a mixed-integer polyhedron, $\mathcal{P}(P(X|\Omega))$, that bounds the support of the posterior.

Working through (14) term by term, the $\mathcal{T}^2_{4\,2}$ term constrains to non-negative, continuous trajectories as defined in equation (9), which is already in linear form. The second term is the support of the prior over the parameters which we ignore here since there are no parameters. The next term is the support of the start state, since we're flipping a coin for each agent state at $t = 0$, each state occupation must be at most 1, which can be expressed as

$$\left\{T : T^0_{\psi 0} + T^0_{\psi 1} \leq \mathbf{1}_\psi\right\}.$$

The fourth term is the support of the observation. Since we observe a cat in the left grid-square at time $t = 1$ we add the constraint

$$T^1_{00} + T^1_{01} = 1.$$

The final term is the constraint due to agent interactions. The impossible interactions are a mouse staying put when there is a cat on the same gridsquare or moving when there are no cats, so we constrain against these

$$\begin{aligned}
\mathrm{supp}(\pi(2, T^t_{*a}\mathbf{1}^a, 0)) &= \left\{T : -T^t_{00} - T^t_{01} \leq -1\right\} \\
\mathrm{supp}(\pi(3, T^t_{*a}\mathbf{1}^a, 0)) &= \left\{T : -T^t_{10} - T^t_{11} \leq -1\right\} \\
\mathrm{supp}(\pi(2, T^t_{*a}\mathbf{1}^a, 1)) &= \left\{T : T^t_{00} + T^t_{01} \leq 0\right\} \\
\mathrm{supp}(\pi(3, T^t_{*a}\mathbf{1}^a, 1)) &= \left\{T : T^t_{10} + T^t_{11} \leq 0\right\}
\end{aligned} \tag{17}$$

for all t. If, for simplicity, we make the Fermionic assumption by adding the constraints

$$\mathbf{0}^t_{\psi a} \leq T^t_{\psi a} \leq \mathbf{1}^t_{\psi a}$$

then using the identity in (16) to take the union of each constraint in (17) with $\left\{T : T^t_{\psi a} = 0, \mathbf{0} \leq T \leq \mathbf{1}\right\}$ finally gives the four constraints

$$\begin{aligned}
-T^t_{00} - T^t_{01} + T^t_{20} &\leq 0 \\
-T^t_{10} - T^t_{11} + T^t_{30} &\leq 0 \\
T^t_{00} + T^t_{01} + 2T^t_{21} &\leq 2 \\
T^t_{10} + T^t_{11} + 2T^t_{31} &\leq 2
\end{aligned}$$

for each timestep $t = 0$ and $t = 1$.

Taken together, these constraints define a polyhedron that bounds the set of (Fermionic) trajectories for the cat and mouse ABM with the given start state distribution and observation.

4 Sampling from the posterior

Having shown how to calculate $\mathcal{P}(P(X|\Omega))$, we now show how to use this to sample from $P(X|\Omega)$.

Various methods of sampling from a mixed-integer polyhedron exist in the literature. Baumert et al. (2009) presents a very general algorithm for sampling from any weighted subset of integer points within a bounding hyper-rectangle. The algorithm relies on sampling points on a random walk. However, in our case the number of valid trajectories could be a vanishingly small proportion of the number of points in the smallest bounding hyper-rectangle, so a random walk would be unlikely to come across any valid trajectories. Universal hashing (Meel et al., 2016) provides a promising way of sampling uniformly from the integer points in a convex polyhedron, but we have found that this technique doesn't scale to the number of dimensions needed for our application.

The Metropolis-Hastings algorithm is another well known and widely used sampling algorithm and this is the approach we take here. To do this we need to define

- a set of Markov states, $\mathcal{M}$, along with a mapping, $E : \mathcal{M} \to \mathbb{Z}^N \times \mathbb{R}^M$, which maps Markov states to sample values, X,
- a stochastic proposal function $f : \mathcal{M} \to \mathcal{M}$ from which we can generate proposal transitions from one Markov state to another,
- a probability distribution $P : \mathcal{M} \to \mathbb{R}$ which gives the probability of each Markov state (this need not be normalised as the Metropolis Hastings algorithm only ever needs probability ratios).

In order to be of use in practice, the proposal function, f, must have the following properties:

- For any two Markov states there should exist a sequence of transitions which forms a path between those states and has a non-zero probability of being proposed and accepted.
- Given a current Markov state, there should be a computationally efficient procedure to generate a proposal and calculate its acceptance probability.

4.1 The set of Markov states

Given the polyhedron $\mathcal{P}(P(X|\Omega))$, we split the constraints into two sets: equalities (i.e. those whose lower and upper bound have the same value) and inequalities (i.e. those whose lower and upper bounds differ) so that

$$\mathcal{P}(P(X|\Omega)) = \left\{X \in \mathbb{Z}^N \times \mathbb{R}^M : (L \leq CX \leq U) \cap (DX = E)\right\}. \tag{18}$$

Our aim will be to use the equality constraints to reduce the dimension of the problem and so improve the efficiency of the algorithm.

Suppose there are N_e equality constraints and we partition the elements of X into 'basic' and 'non-basic' elements, X_B and X_N respectively, so that there are exactly N_e basic elements (note that since the continuity constraints (8) are equality constraints then there are always at least $(N-1)S$ basic elements).

If we let Q be a permutation matrix that separates the basic from non-basic elements so that

$$QX = \begin{pmatrix} X_B \\ X_N \end{pmatrix}$$

then we can write

$$DX = DQ^{-1}\begin{pmatrix} X_B \\ X_N \end{pmatrix} = (B \mid N)\begin{pmatrix} X_B \\ X_N \end{pmatrix} = E$$

so

$$BX_B + NX_N = E. \tag{19}$$

Now, since there are N_e basic variables and N_e equality constraints, B is square so if we choose the basic variables to ensure that B has an inverse then

$$X_B = B^{-1}(E - NX_N) \tag{20}$$

so letting

$$V = Q^{-1}\begin{pmatrix} B^{-1}E \\ \mathbf{0} \end{pmatrix},\ M = Q^{-1}\begin{pmatrix} -B^{-1}N \\ \mathbf{1} \end{pmatrix}$$

gives

$$X = V + MX_N. \tag{21}$$

In section 4.4 we'll show how to choose X_B to ensure that B has an inverse and the integer dimensions of X_B remain integer as long as the integer dimensions of X_N are integer. Given this, then the original polyhedron is equivalent to the reduced dimension polyhedron

$$\mathcal{P}' = \left\{X_N \in \mathbb{Z}^{N'} \times \mathbb{R}^{M'} : L - CV \leq CMX_N \leq U - CV\right\} \tag{22}$$

where N' and M' are the number of integer and real-valued elements remaining in X_N.

Since Q is just a permutation matrix, X_N and X_B can be bound within hyper-rectangles by transforming the $\mathcal{P}$-bounding hyper-rectangle, H, from section 3.1

$$QH = \begin{pmatrix} H_B \\ H_N \end{pmatrix}.$$

So, X_N is bound by $0 \leq X_N \leq H_N$. Given this, we define the set of Markov states to be

$$\mathcal{M} = \{X_N : 0 \leq X_N \leq H_N, \}.$$

and equation (21) maps each Markov state, $X_N \in \mathcal{M}$, to a unique sample, X. This set includes values outside of the polyhedron (22), we'll show how we deal with that later.

4.2 The proposal function

In order to propose a new Markov state given a current state, X_N, first choose an element of X_N and a direction in which to perturb that element. This defines a *rounded perturbed state*, X'_N, which is equal to X_N but with one of its elements rounded to the nearest integer and perturbed by ± 1. Only rounded perturbed states that remain inside the hyper-rectangle H_N are considered. Real valued variables should be scaled so that a perturbation of ± 1 usually changes $P(X|\Omega)$ by a small amount. The boundaries, H_N, should be chosen to lie on an integer value plus 0.5 for real-valued variables. So, we can define the set of all valid perturbations

$$\mathcal{D}(X_N) = \{\langle i, \delta\rangle : X'_N = \lfloor X_N \rceil_i + \delta e_i, 0 \leq X'_N \leq H_N, \delta \in \{+1, -1\}, i \in \mathcal{V}\}$$

where e_i is the unit vector with a 1 in the i^{th} element, $\mathcal{V}$ is the set of indices of X_N and we use the notation $\lfloor . \rceil_i$ to denote the rounding of the i^{th} element of a vector to the nearest integer.

Assume for now that we have an approximation of the target distribution $\tilde{P}(X_N) \approx P(X_N|\Omega)$. Given a current state, X_N, the probability of choosing perturbation $\langle i, \delta\rangle \in \mathcal{D}(X_N)$ is given by

$$P(X_N, i, \delta) = \frac{\min\left(1, \frac{\tilde{P}(\lfloor X_N \rceil_i + \delta e_i)}{\tilde{P}(\lfloor X_N \rceil_i)}\right)}{S(X_N)} \tag{23}$$

where

$$S(X_N) = \sum_{\langle i,\delta\rangle \in \mathcal{D}(X_N)} \min\left(1, \frac{\tilde{P}(\lfloor X_N \rceil_i + \delta e_i)}{\tilde{P}(\lfloor X_N \rceil_i)}\right) \tag{24}$$

Given a perturbation $\langle i, \delta\rangle$, the final proposed state is given by

$$X'_N = \lfloor X_N \rceil_i + (\delta + \epsilon) e_i$$

where ϵ is a *fractional perturbation* drawn with uniform probability from $[-0.5 : 0.5]$ if the i^{th} element is real-valued or $\epsilon = 0$ if it is integer-valued.

4.2.1 Approximating the posterior

The proposal function requires an approximation of the target distribution, $P(X|\Omega)$. For this, we use a function of the form

$$\tilde{P}(X) = \prod_i \tilde{P}_i(Z_i X) \tag{25}$$

where Z_i are row vectors and $\tilde{P}_i$ are univariate factors of any form. We call this a *linearly factorized distribution*, where the linearity lies in the Z_i linear reductions while the P_i can be non-linear.

The first thing to note is that we can express $\mathcal{P}(P(X|\Omega))$ as a linearly factorized distribution in the following way: first define the infeasibility function $\iota(l, x, u)$ to be equal to 0 if $l < x < u$ or equal to the distance to the nearest bound otherwise

$$\iota(l, x, u) = \begin{cases} x - u & \text{if } x > u \\ l - x & \text{if } x < l \\ 0 & \text{otherwise.} \end{cases}$$

Now, given a set of constraints $L \leq CX \leq U$, if we define a factor for each constraint

$$\tilde{P}_i(C_i X) = e^{\frac{-\iota(L_i, C_i X, U_i)}{\tau}} \tag{26}$$

so that

$$\prod_i \tilde{P}_i(C_i X) = e^{\frac{-\sum_i \iota(L_i, C_i X, U_i)}{\tau}} \tag{27}$$

is a linearly factorized distribution that is 1 inside the polyhedron and that decays exponentially outside at a rate set by τ, so as $\tau \to 0$ equation (27) tends to an indicator function for the set of points in $\mathcal{P}(P(X|\Omega))$.

A linearly factorized distribution in X can easily be transformed into a linearly factorized distribution in X_N by using the linear transform in equation (21). Transformation to X_N ensures that the resulting distribution satisfies the equality constraints $D(V + MX_N) = E$, while the factors in equation (27) can be used to ensure that $0 \leq X_B \leq H_B$ and that the constraints in (22) are satisfied.

Turning our attention now to the distribution inside the polyhedron, if we look back at equation (12) we can see that the posterior of an ABM is already highly factorized and many of the factors are already linearly factorised.

However, when considering the efficiency of the final algorithm, there is a trade off between the closeness of $\tilde{P}(X_N)$ to $P(X_N|\Omega)$ and the number of factors involved in its calculation. In our experiments, we have found we can attain very good acceptance probabilities with relatively simple renderings of $\tilde{P}(X_N)$. The start state, likelihood function and prior over θ are often already uncorrelated functions of the elements of θ and the state occupation numbers, so are already in the correct form. This leaves only the multinomial term in equation (12) which we approximate with the linearly factorized

$$\prod_{t,\psi} (T^t_{\psi b}\mathbf{1}^b)! \prod_a \frac{\tilde{\pi}^t_{\psi a}{}^{T^t_{\psi a}}}{T^t_{\psi a}!} \tag{28}$$

where

$$\log(\tilde{\pi}^t_{\psi a}) = \frac{\int T^t_{\psi a} \log\left(\pi_\theta(\psi, \Psi^t, a)\right) P(T,\theta) dT d\theta}{\int T^t_{\psi a} P(T,\theta) dT d\theta}.$$

This integral can be calculated by drawing samples from the prior $P(T,\theta)$ and using Monte Carlo integration. This approximation marginalises over θ, if π_θ has a particularly strong dependence on θ then it may be worth capturing some of that dependence by, for example, replacing the $T^t_{\psi a}$ term in (28) with $T^t_{\psi a} + R\theta$ for some row vector R, normalising and minimising the expectation of the difference between the log probabilities of the approximation and the true multinomial.

So, the final approximation, $\tilde{P}(X_N)$, is given by the product of the factors of the bounding convex polyhedron (27), the multinomial approximation (28) and the approximations of the observations, the start state (boundary conditions) and $P(\theta)$.

Since some of the Markov states are infeasible, calculating $\tilde{P}$ for these states may require the evaluation of a factor on a value that is outside its domain (for example, equation (28) is undefined if $T^t_{\psi a}$ is negative). In this case we choose the value at the nearest point that is within the domain.

4.3 The probability of a Markov state

If a Markov state satisfies all the constraints in (22) then its probability is defined to be equal to the true posterior given by (12)

$$P(X_N) = P(X|\Omega) = P(T,\theta|\Omega)$$

where

$$X = V + MX_N = W^{\psi ai}_t T^t_{\psi a} + G^i_j \theta^j.$$

If the constraints are not all satisfied, then the probability is defined to be the product of the factors in (12) and the infeasibility factors in (27). If any factor in (12) becomes zero as a consequence of the infeasibility, then it is replaced by its linearly factorized approximation (which must be non-zero otherwise the state wouldn't have been proposed).

This means that, as long as τ is finite, the Markov chain will pass through some infeasible states. However, if we just ignore these and only take samples from feasible states then we end up with samples from the true posterior. Allowing the Markov chain to pass through infeasible states has the advantage of ensuring that there exists a path between any two feasible states and improves mixing. When used in the context of a particle filter, it also solves the problem of unexpected observations that can't be generated by any particle, as we can allow the particles to become temporarily infeasible. The price we pay for this is the computational cost of moving through infeasible states that don't generate useful samples.

It's worth noting that equation (27) is a Boltzmann distribution which describes a thermodynamic system, with an "energy" equal to the infeasibility of X, at equilibrium with a heat bath of "temperature" τ. It has been shown that simulations of thermodynamic systems of this type are able to solve large linear integer optimisation problems like the one we're dealing with here (Kirkpatrick et al., 1983). Here, we won't be changing the temperature during the sampling process, but we do need to choose a value for τ. Higher temperatures will increase mixing in the Markov chain, but will also increase the proportion of time spent in infeasible states so we need to find a temperature that is high enough to ensure good mixing of the chain and low enough to ensure that a reasonable proportion of samples are feasible. We have found that these two competing effects roughly cancel each other out when measuring the overall efficiency of the

algorithm in effective samples per second, so the value of τ is not critical within a certain range. We have found that a good rule of thumb is to set the temperature so that 50-80% of the samples in a chain are infeasible.

From the Metropolis-Hastings algorithm, the probability of accepting a proposal $X'_N = \lfloor X_N \rceil_i + (\delta + \epsilon)e_i$ is given by

$$\alpha = \min\left(1, \frac{P(X'_N)P(X'_N, i, -\delta)}{P(X_N)P(X_N, i, \delta)}\right) = \min\left(1, \frac{P(X'_N)}{P(X_N)}\frac{\tilde{P}(\lfloor X_N \rceil_i)}{\tilde{P}(\lfloor X'_N \rceil_i)}\frac{S(X_N)}{S(X'_N)}\right).$$

Since $\tilde{P}(X_N)$ approximates $P(X_N)$, $\frac{P(X'_N)}{P(X_N)}\frac{\tilde{P}(\lfloor X_N \rceil_i)}{\tilde{P}(\lfloor X'_N \rceil_i)}$ should be close to 1. We'll see in the next section that if we choose the basis carefully then $S(X_N)$ will not change very much between transitions so the ratio $\frac{S(X_N)}{S(X'_N)}$ should also be close to 1 meaning that a good proportion of proposals should be accepted. In our experiments around 85% of proposals were accepted.

4.4 Choosing a basis

Our definition of a Markov state depends on a partition of X into basic and non-basic variables such that B in equation (20) has an inverse, B^{-1}, and if $X_N \in \mathbb{Z}^{N'} \times \mathbb{R}^{M'}$ then $X \in \mathbb{Z}^N \times \mathbb{R}^M$ (i.e. if X_N has all integers in the correct elements then so does X). In general there exist many partitions that satisfy these requirements so it remains to define a method of choosing one. The choice of basis affects the efficiency of the Markov chain via:

- the computational cost of updating the proposal probabilities $P(X_N, i, \delta)$ and the sum $S(X_N)$ after each transition.
- the effect of the ratio of sums $S(X_N)/S(X'_N)$ on the acceptance probability
- the mixing rate of the Markov chain, since the choice of basis affects the available transitions in X space.

From (23) and (24) we see that in order to calculate $P(X_N, i, \delta)$ and $S(X_N)$ we need to calculate $\frac{\tilde{P}(\lfloor X_N \rceil_i + \delta e_i)}{\tilde{P}(\lfloor X_N \rceil_i)}$ for all $\langle i, \delta \rangle \in \mathcal{D}(X_N)$. Since $\tilde{P}$ is linearly factorised

$$\frac{\tilde{P}(\lfloor X_N \rceil_i + \delta e_i)}{\tilde{P}(\lfloor X_N \rceil_i)} = \frac{\prod_j \tilde{P}_j(Z_j(\lfloor X_N \rceil_i + \delta e_i))}{\prod_j \tilde{P}_j(Z_j \lfloor X_N \rceil_i)} = \prod_{j: Z_j e_i \neq 0} \frac{\tilde{P}_j(Z_j \lfloor X_N \rceil_i + \delta Z_j e_i)}{\tilde{P}_j(Z_j \lfloor X_N \rceil_i)}.$$

Let

$$P'(X_N, i, \delta) = \min\left(1, \prod_{j: Z_j e_i \neq 0} \frac{\tilde{P}_j(Z_j \lfloor X_N \rceil_i + \delta Z_j e_i))}{\tilde{P}_j(Z_j \lfloor X_N \rceil_i)}\right)$$

be the un-normalised proposal probability. If we accept a perturbation $\langle k, \delta' \rangle$, $P'(X_N, i, \delta)$ needs to be updated to $P'(\lfloor X_N \rceil_k + (\delta' + \epsilon)e_k, i, \delta)$. However, this update only affects factors $\tilde{P}_j$ such that $\{j : Z_j e_k \neq 0\}$, so $P'(X_N, i, \delta)$ only changes if $\{j : Z_j e_i \neq 0\}$ intersects with $\{j : Z_j e_k \neq 0\}$. So, if we let Z be the matrix formed from the row vectors Z_i and choose a basis such that Z is sparse, $P'(X_N, i, \delta)$ can be stored for each value of $\langle i, \delta \rangle \in \mathcal{D}(X_N)$, and only values for a few values of i will need to be recalculated on transition. Tang (2022) shows that a binary sum-tree data structure can be used to efficiently update and draw from the un-normalised probability distribution defined by $P'(X_N, i, \delta)$ while also keeping track of $S(X_N)$.

The effect of $\frac{S(X_N)}{S(X'_N)}$ on the acceptance probability is also improved if Z is sparse because only a small percentage of the terms in the sum change between $S(X_N)$ and $S(X'_N)$ so we should expect their ratio to be close to 1.

The Markov chain will also mix better if the columns of Z are sparse. Mihelich et al. (2018) shows that the Kolmogorov-Sinai entropy of the Markov process is a good proxy for its mixing speed. In the context of our algorithm the entropy is higher if the neighbours of a Markov state have similar probability. If Z is sparse (and short) then adjacent Markov states are more likely to have similar probabilities, all else being equal.

So, our aim will be to find a basis that keeps Z sparse, while maintaining the correct integer signature of X. To do this we use the following algorithm: Start with a linearly factorized distribution $\tilde{P}$ on X, rather than X_N

$$\tilde{P}(X) = \prod_i \tilde{P}_i(C_i X).$$

and a set of equality constraints that need to be satisfied for all Markov states X_N

$$DX = E$$

so we can form a mixed-integer polyhedron

$$\mathcal{P} = \left\{X \in \mathbb{Z}^N \times \mathbb{R}^M : L \leq CX \leq U \cap DX = E\right\}$$

C is usually sparse to start with, so our aim is to maintain that sparsity while using the equality constraints to reduce the dimension. If we let

$$W = \begin{pmatrix} D \\ C \end{pmatrix}$$

and

$$F = \begin{pmatrix} E \\ \mathbf{0} \end{pmatrix}$$

then

$$\mathcal{P} = \left\{ X \in \mathbb{Z}^N \times \mathbb{R}^M : \begin{pmatrix} \mathbf{0} \\ L \end{pmatrix} \leq WX - F \leq \begin{pmatrix} \mathbf{0} \\ U \end{pmatrix} \right\} \tag{29}$$

Our aim is to mark variables from X as basic one at a time. Once we're done, the remaining variables will make up the non-basic variables, X_N.

Given a set of constraints in the form (29), we mark the j^{th} element of X as basic and the i^{th} equality constraint as "reduced" by choosing an element W^i_j to be a *pivot point* and performing Gaussian elimination on the j^{th} column of W so that W^i_j is the only non-zero element in the column W_j. This can be done by pre-multiplying by G

$$\begin{pmatrix} \mathbf{0} \\ L \end{pmatrix} \leq GWX - GF \leq \begin{pmatrix} \mathbf{0} \\ U \end{pmatrix}$$

where G is the matrix

$$G = \begin{pmatrix} 1 & & & -\frac{W^0_j}{W^i_j} & & \\ & \ddots & & \vdots & & \\ & & 1 & -\frac{W^{i-1}_j}{W^i_j} & & \\ & & & \frac{1}{W^i_j} & & \\ & & & -\frac{W^{i+1}_j}{W^i_j} & 1 & \\ & & & \vdots & & \ddots \\ & & & -\frac{W^n_j}{W^i_j} & & 1 \end{pmatrix}$$

Letting $\hat{W} = GW$ and $\hat{F} = GF$ recovers the canonical form

$$\begin{pmatrix} \mathbf{0} \\ L \end{pmatrix} \leq \hat{W}X - \hat{F} \leq \begin{pmatrix} \mathbf{0} \\ U \end{pmatrix} \tag{30}$$

but now with the j^{th} element of X marked as basic and the i^{th} row of $\hat{W}$ marked as "reduced". We now repeat the process until no more equality constraints can be reduced.

The use of Gaussian elimination here ensures that B^{-1} exists. In order to ensure that X_B has the correct integer structure, given that X_N does, we add the requirement that we can only mark integer elements of X as basic on a pivot point W^i_j such that the row vector W^i is zero on all non-integer columns and W^i_j divides W^i_k for all k (so for rows that have a non-zero element on a real-valued column, we always eliminate on a real-valued column). This requirement ensures that a reduced equality constraint has the form $X^i_B = F^i - W^i X_N$ where W^i is a row vector of integer coefficients on integer elements of X_N. If this constraint has any integer solution then F^i must also be an integer. So, if X_N has the correct integer structure, then X^i_B is also an integer.

This defines a set of valid pivot points at each elimination step. In order to decide which point to choose, we note that when we pivot on W^i_j, GW differs from W in at most

$$\mu = (|W_j|_0 - 1)(\left|W^i\right|_0 - 1)$$

elements, where $|\cdot|_0$ is the L0-norm (i.e. the number of non-zero elements in a vector or matrix). So, this gives an upper bound on $|WG|_0 - |G|_0$, the change in L0-norm (or reduction in sparsity) of W when we pivot on W^i_j. It has been found that choosing W^i_j to be the element that minimises μ (known as the Markowitz criterion) is a computationally efficient way of maintaining the sparsity of a matrix while performing Gaussian elimination (Markowitz, 1957; Suhl & Suhl, 1990; Maros, 2002). So, in order to find a sparse basis, we calculate the set of valid pivot points, choose the point that minimises μ, perform the pivot and repeat until no more valid pivot points remain. Suhl & Suhl (1990) presents a computationally efficient algorithm for doing this.

Aget type	Adjacent agents	Behaviour	Probability
Prey	No predators	die	0.100
		give birth	0.156
		move	0.744
Prey	Predators > 0	die	0.400
		give birth	0.156
		move	0.444
Predator	No prey	die	0.100
		give birth	0.000
		move	0.900
Predator	Prey > 0	die	0.100
		give birth	0.300
		move	0.600

Table 1: The probabilities of each behaviour in the predator-prey model

5 Spatial predator-prey demonstration

We demonstrate the above techniques on a 32x32 grid of "predator" and "prey" agents. All agents can move to an adjacent grid-square (i.e. up, down, left or right), give birth to another agent on the an adjacent grid-square or die. Predators only give birth when there is at least one prey on an adjacent grid-square (i.e. if there is a source of food close by). Prey may be eaten by predators so their probability of dying is much greater when predators are present on adjacent grid-squares. If any agent moves off the edge of the grid, it reappears at the opposite edge (i.e. the grid is topologically a torus). The probability of each agent's behaviour is shown in table 1. These values were chosen to minimise the probability that either species becomes extinct.

In order to generate observation data to assimilate, we used the following procedure:

1. Generate a start state by drawing the number of predators and prey in each gridsquare at $t = 0$ from a Bernoulli distribution with probability 0.05. i.e. for $\Psi_*^0 \in \{0, 1\}^S$
$$P(\Psi_*^0) = \prod_\psi 0.05^{\Psi_\psi^0} 0.95^{1-\Psi_\psi^0} \tag{31}$$
2. Simulate the ABM for 16 timesteps from the start state. If the resulting trajectory isn't Fermionic, repeat from step 1. The resulting trajectory, T_{real}, is considered to be the "real-world" trajectory from which we take observations.
3. For each timestep and gridsquare of T_{real}, take two draws from a Bernoulli distribution with probability 0.05. If the first draw is 1, then observe the number of predators in that gridsquare at that time. If the second draw is 1 then observe the number of Prey. The observations were noiseless so the observed count was the true count.

Given the (Fermionic) model, the start state $P(\Psi_*^0)$ and the observations, an approximation of the posterior was generated as described in section 4.2.1 and used to generate a sparse basis as described in section 4.4. The resulting basis had 22.4 non-zero elements per million.

Four initial feasible solutions were generated in the following way:

1. Draw a start state from $P(\Psi_*^0)$.
2. Simulate for 16 timesteps to get a (non-Fermionic) prior trajectory.
3. Starting with the prior trajectory, generate proposals as described in section 4.2 and accept with probability 1.0 until a feasible state is reached.

Starting with these four initial solutions, four separate Markov chains were generated using the Matropolis-Hastings algorithm as described in section 4. 1,000,000 samples were then taken from each chain and discarded in order to burn them in. Finally, 10,000,000 samples were taken from each chain and split into first and last halves to give 8 sample sequences of 5,000,000 samples each. For all samples, the temperature, τ, was set to 0.1.

A feasible sample took an average of 42μs to generate on a single core of a 1.6GHz Intel i5-8250U. 85% of the proposals were accepted and 70% of samples were infeasible.

In order to assess the convergence of the chains, a set of summary statistics were calculated for each sample of each sequence. The summary statistics consisted of the number of agents within each shaded square shown in figure 2

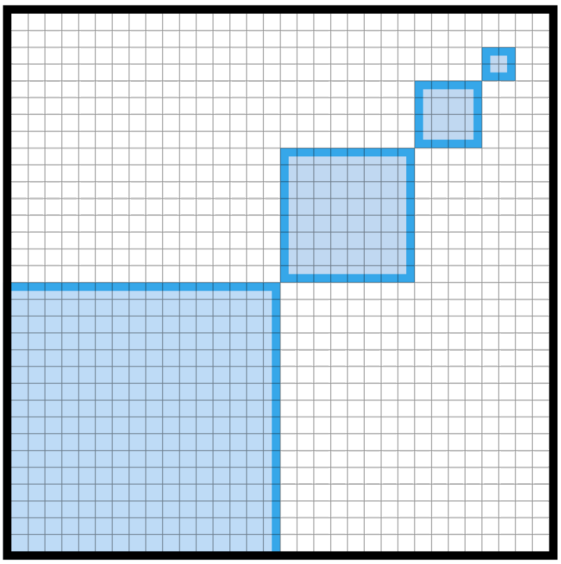

Figure 2: Summary statistics consist of the total number of agents in each of the four shaded regions.

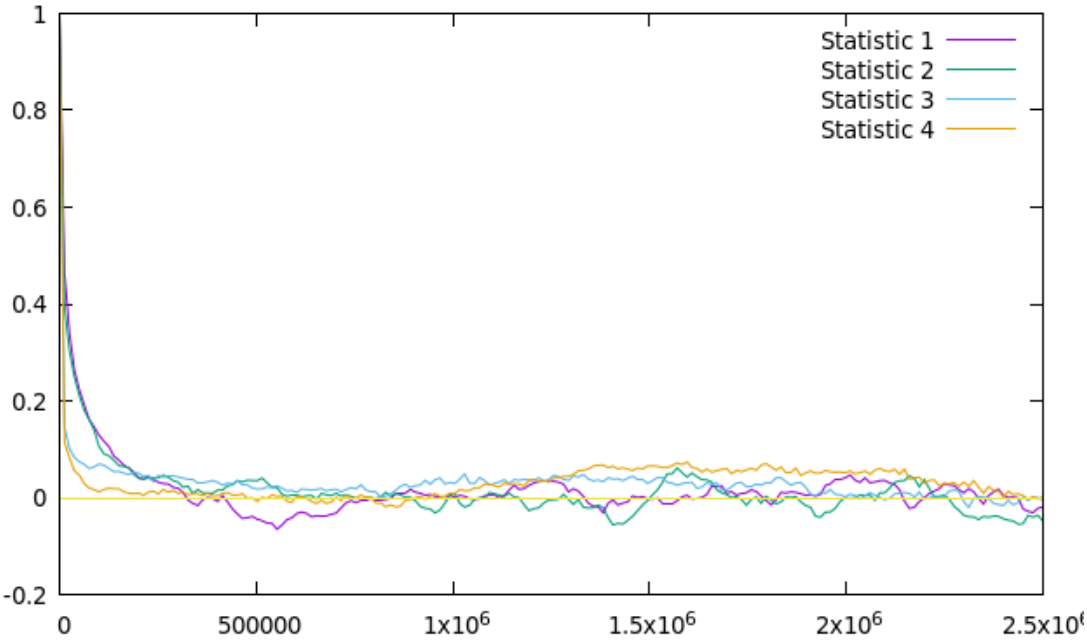

Figure 3: Mean autocorrelation for each of the summary statistics, averaged over all chains, as a function of the lag in number of samples.

measured at the end of the last timestep. We'll refer to these as statistic $1 \ldots 4$ going from largest to smallest area. This arrangement of regions was chosen in order to capture the convergence at different spatial scales. From the summary statistics, we calculated the the Gelman-Rubin diagnostic (Gelman & Rubin, 1992). This gives a measure of the uncertainty in the standard deviation of the statistic due to the finite length of the sample sequences. If this number is close to 1, then we have some justification in believing that we have taken enough samples. A value of less than 1.1 is often used as a criterion for convergence. We also approximated the autocorrelation of each statistic at various time lags for each sample sequence. A decline of autocorrelation to zero long before the lag reaches the total number of samples shows good convergence. Finally, we approximated the number of effective samples in each sequence (i.e. the number of samples from a perfect IID sampler that would give the same uncertainty in the mean of the statistic). Details on how these measures were calculated are given in the appendix and our results are shown in figure 3 and table 2.

Finally, figure 4 shows the mean number of agents at the end of the simulation, averaged over all samples, along with the end state of the "real" trajectory T_{real}. This is a visual way of showing that the observations have given rise to a posterior that gives some information about the true positions of the agents.

Measure	Statistic 1	Statistic 2	Statistic 3	Statistic 4
Gelman-Rubin	1.00727	1.01167	1.00451	1.00153
Effective samples	420	417	489	1019

Table 2: The Gelman Rubin diagnostic and number of effective samples per sample-sequence for each statistic

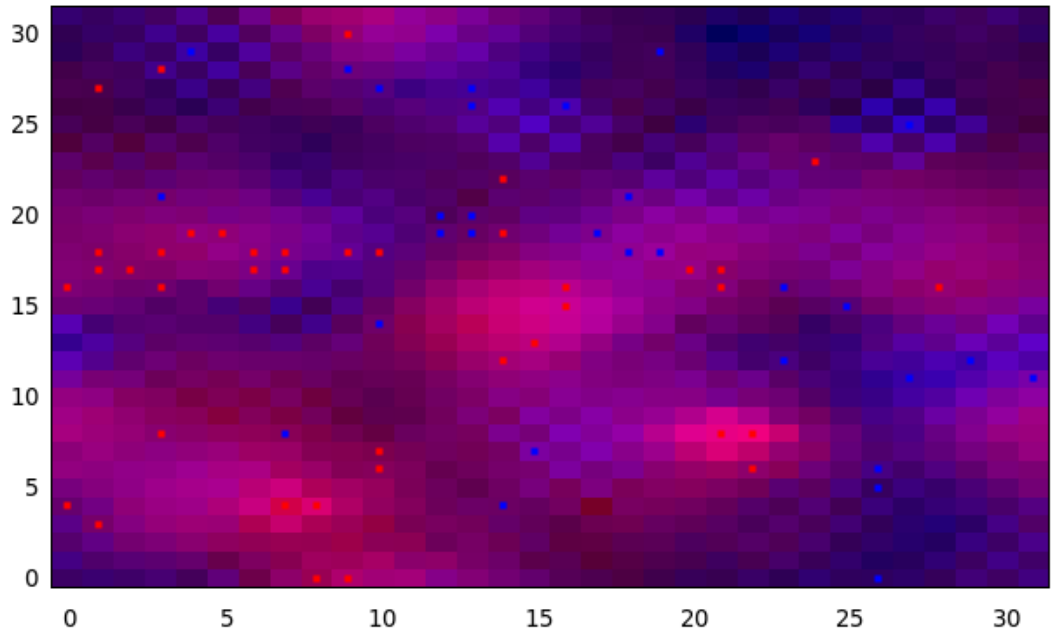

Figure 4: A representation of the positions of the agents at time $t = 16$. The dots represent the positions of agents in the "real" trajectory, T_{real}, from which the observations were generated. Blue dots represent predators, red dots represent prey. The background colour of each square has blue/red intensity proportional to the log of the mean number of predators/prey in that square averaged over all samples.

The code used to generate these results can be found at `https://github.com/danftang/AgentBasedMCMC`.

6 Discussion

6.1 Integration into sequential MCMC

The predator-prey example demonstrated assimilation over a relatively short simulation period. In practice it's likely that assimilation would be required over a longer time period, or perhaps the observational data arrives in a continuous stream. As discussed earlier, the way to deal with this is to split the longer time period into windows as in equation (5) and treat each window separately. We can make use of the MCMC sampler to implement this broad idea in many different ways.

One way would be the following: given N samples from the posterior $X_t^{1:N} \sim P(X_t|\Omega_{1:t})$, approximate the posterior from some family of distributions, $P_\Theta(X_t)$, by finding the Θ that minimises the KL-divergence from $P_\Theta(X_t)$ to $P(X_t|\Omega_{1:t})$. The KL-divergence can itself be approximated from the samples $X_t^{1:N}$ with

$$D_{KL}\left(P(X_t|\Omega_{1:t}) \mid P_\Theta(X_t)\right) \approx \sum_{s=1}^{N} \frac{1}{N} \log\left(\frac{1}{N P_\Theta(X_t^s)}\right)$$

which is minimised when

$$\Theta \approx \min_\Theta \left(-\sum_{s=1}^{N} \log\left(P_\Theta\left(X_t^s\right)\right)\right).$$

The approximation, $P_\Theta(X_t)$, can then be inserted into the recursion equation (5) to generate samples from the next window $X_{t+1}^{1:N} \sim P(X_{t+1}|\Omega_{1:t+1})$, from where we can repeat the cycle indefinitely. The exact form of P_Θ is model dependent. The simplest form would be a product of univariate Poisson distributions, one for each of the integer variables, and Gaussians for the real-valued variables. If we wish to capture correlations between variables, a multivariate Poisson distribution could be used (see for example Muñoz-Pichardo et al. (2021), where P_Θ is a product of univariate

Poisson distributions whose rate parameters are given by $\Lambda = BX_R + AX_I$, where X_R are the real-valued variables and X_I are the integer valued variables, B and A are matrices, and A is triangular).

An alternative approach is to use the *Resample-Move* algorithm presented in Gilks & Berzuini (2001). This is a particle filter based on the sequential importance resampling algorithm described earlier. The innovation comes in the form of an additional MCMC *move* step after each resample step. At each move step each particle, $X_t^i = \left\langle \tau_{1:t}^i, \theta^i \right\rangle$, is "moved" by initialising the Markov chain with X_t^i and drawing a single new sample $X_t'^i \sim P(X_t|\Omega_{1:t})$ by iterating one or more times. This solves the problem of particle impoverishment because even if multiple samples collapse to the same state during the resample step, the subsequent move step is likely to move them apart again. The algorithm remains exact because the samples coming out of the m^{th} resample step are already distributed according to the target distribution, $P(X_t|\Omega_{1:t})$, but this is also the stationary distribution of the Markov transition so the moved samples are also distributed according to the target distribution.

The Resample-Move algorithm still uses an importance sampling step with a draw from the prior which, as we have seen, is often a problem for ABMs because drawing from the prior is likely to result in a sample with zero weight. If this is the case, we can replace the draw from the prior with an draw from $P(X_{t+1}|\Omega_{1:t+1}, X_t^i)$ using our MCMC algorithm for each particle X_t^i from the previous timestep. The correct weight of the particle is then proportional to $P(\Omega_{t+1}|X_t^i)$ (Doucet et al., 2009). Unfortunately we don't immediately know this value. If we're doing an assimilation step every timestep, or if the ABM is simple enough, it may be possible to calculate this value analytically, however, often this is not possible. Han & Carlin (2001); Newton & Raftery (1994); Štefankovič, Vempala, & Vigoda (2009) give a number of ways of approximating these values, at least up to a multiplicative constant (which is all we need). The calculation of $P(\Omega_{t+1}|X_t^i)$ can be avoided if, during the move step of the Resample-Move algorithm, we're willing to iterate the MCMC moves until convergence over the whole trajectory. In this case, the particles regain equal weighting through the MCMC moves. Since all particles end up with equal weight, there is no need for a resampling step so we never get impoverishment. The disadvantage is that we need to iterate to convergence on each particle, so it is more computationally expensive. This scheme can be made more computationally efficient if we're content with samples from $P(X_{t-L}|\Omega_{1:t})$ for some time lag L, that is we're getting samples from the smoothed values at time $t - L$ which account for observations up to time t. In this case we take samples from $P(\tau_{1:t}, \theta|\Omega_{1:t})$ and marginalise to get samples from $P(X_{t-L}|\Omega_{1:t})$. We now only need to iterate the MCMC moves until the marginalised values converge, but since a new observation $\Omega_{1:t+1}$ has much smaller effect on X_{t-L} than on X_t we expect to reach convergence of the marginal in fewer iterations for a fixed measure of convergence.

As the number of assimilated windows increases, there may be a lag L where we are happy to assume that the new observations Ω_{t+1} do not change our beliefs about the model state before the end of the $(t - L)^{th}$ window, i.e. $P(\tau_{1:t-L}|\Omega_{1:t}, \theta) \approx P(\tau_{1:t-L}|\Omega_{1:t+1}, \theta)$. In this case, if we're doing state estimation rather than parameter estimation (i.e. θ is fixed) it is sufficient to perturb the trajectories of only a finite number of windows into the past when performing the MCMC moves, so we perturb $\tau_{t-L+1:t+1}$ while holding $\tau_{1:t-L}$ fixed. This means that the dimension of the MCMC problem does not increase as we assimilate more windows. The early parts of the trajectory are then subject to possible particle impoverishment again if we're using an algorithm with a resampling step, but this time it doesn't matter because it is, by assumption, not affecting the distribution of the current state.

6.2 Limitations and further work

This algorithm presented here is intended as the proof-of-concept of a novel approach to MCMC sampling from an ABM rather than a production-ready tool for use by practitioners. At present, the bounding of the supports of the agent timestep and observation likelihood functions by a convex polyhedron must either be done by hand or by third party abstract interpretation tools. This process could be automated, taking as input a computer program and outputting a set of linear constraints for the support. Similarly, the temperature setting, τ, at present needs to be set by trial-and-error, whereas an automated warm-up or adaptive scheme would improve the algorithm's efficiency and usability.

As presented, the algorithm is only applicable if the internal state of an agent can be represented in a few bytes. This is because it depends on explicitly representing the probability of each Markov transition at each iteration. But the number of possible transitions is of the order of the number of non-basic variables, and this expands exponentially with the size of the internal state of an agent. So, if agents have a large amount of internal state it will become impractical to represent this distribution explicitly. This problem can be overcome by considering only a subset of non-basic variables for update at each iteration, in a similar way to *partial pricing* in the simplex algorithm (Maros, 2002). For example, given the current Markov state, only consider updating the non-basic variables that correspond to alternative actions of some agent in the current trajectory at some timestep. In this situation we also can't explicitly store the coefficients of the linear constraints, C, so in order to calculate the effect of a perturbation on the basic variables we'd calculate this on the fly from equations (7) and (8), the agent timestep function and the observation operators. Finding a good basis of non-basic variables could also be done on the fly for some subset of variables, or alternatively equality constraints could be left as-is as inequalities with equal lower and upper bounds. More work needs to be done exploring the relative efficiency of these options.

7 Conclusion

We have described and demonstrated an algorithm to sample $\langle$model-trajectory, parameter$\rangle$ pairs from the posterior distribution of an agent-based model given a set of observations. We derived an analytical expression for the posterior and showed how to generate a convex polyhedron that bounds the support of this distribution. We then showed how this support can be used to construct an approximation of the posterior in the form of a linearly factorized distribution, and how this can be embedded in a Metropolis-Hastings algorithm to generate samples from the posterior. This allowed us to perform Bayesian inference with a spatial predator/prey model and so demonstrate data assimilation on an ABM.

The current lack of tools to perform data assimilation with agent-based models has restricted their use in data analytics. The development of more powerful techniques for performing Bayesian inference with ABM could transform the usefulness and applicability of these models and the algorithm described here will hopefully be a first step towards this transformation.

Appendix A Calculation of convergence statistics

Given m sample sequences, each containing n samples, let x_{ij} to refer to the i^{th} sample of the j^{th} sequence and let

$$\bar{x}_j = \frac{1}{n}\sum_{i=1}^{n} x_{ij}$$

be the mean of the j^{th} sequence and

$$\bar{x} = \frac{1}{m}\sum_{j=1}^{m} \bar{x}_j$$

be the mean over all sequences.

Let W be the within-sequence variance

$$W = \frac{1}{m}\sum_{j=1}^{m} \frac{1}{n-1}\sum_{i=1}^{n}(x_{ij} - \bar{x}_j)^2$$

and B be the between-sequence variance

$$B = \frac{n}{m-1}\sum_{j=1}^{m}(\bar{x}_j - \bar{x})^2$$

Following Gelman et al. (2013), an overapproximation of the true variance of the statistic can be calculated as

$$\widehat{\text{var}}^{+} = \frac{n-1}{n}W + \frac{1}{n}B$$

and the Gelman-Rubin statistic can be defined as

$$\hat{R} = \sqrt{\frac{\widehat{\text{var}}^{+}}{W}}$$

Also following Gelman et al. (2013), we approximate the autocorrelation as

$$\hat{\rho}_t = 1 - \frac{V_t}{2\widehat{\text{var}}^{+}}$$

where

$$V_t = \frac{1}{n-t}\sum_{i=1}^{n-t}(x_i - x_{i+t})^2.$$

$\hat{\rho}_t$ can be used to calculate the effective number of samples using

$$\hat{n}_e = \frac{mn}{1 + 2\sum_t \hat{\rho}_t}$$

where the sum runs from $t = 0$ until the first t such that $\hat{\rho}_t \leq 0$.

References

Andrieu, C., Doucet, A., Singh, S. S., & Tadic, V. B. (2004). Particle methods for change detection, system identification, and control. *Proceedings of the IEEE*, *92*(3), 423–438.

Bagnara, R., Hill, P. M., & Zaffanella, E. (2008). The parma polyhedra library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, *72*(1-2), 3–21.

Baumert, S., Ghate, A., Kiatsupaibul, S., Shen, Y., Smith, R. L., & Zabinsky, Z. B. (2009). Discrete hit-and-run for sampling points from arbitrary distributions over subsets of integer hyperrectangles. *Operations Research*, *57*(3), 727–739.

Becchi, A., & Zaffanella, E. (2018). An efficient abstract domain for not necessarily closed polyhedra. In *International static analysis symposium* (pp. 146–165).

Carrassi, A., Bocquet, M., Bertino, L., & Evensen, G. (2018). Data assimilation in the geosciences: An overview of methods, issues, and perspectives. *Wiley Interdisciplinary Reviews: Climate Change*, *9*(5), e535.

Chatterjee, S., & Diaconis, P. (2018). The sample size required in importance sampling. *The Annals of Applied Probability*, *28*(2), 1099–1135.

Clay, R., Kieu, L.-M., Ward, J. A., Heppenstall, A., & Malleson, N. (2020). Towards Real-Time Crowd Simulation Under Uncertainty Using an Agent-Based Model and an Unscented Kalman Filter. In Y. Demazeau, T. Holvoet, J. M. Corchado, & S. Costantini (Eds.), *Advances in Practical Applications of Agents, Multi-Agent Systems, and Trustworthiness. The PAAMS Collection* (Vol. 12092, pp. 68–79). Cham: Springer International Publishing. doi: 10.1007/978-3-030-49778-1_6

Conforti, M., Cornuéjols, G., & Zambelli, G. (2010). Polyhedral approaches to mixed integer linear programming. In *50 years of integer programming 1958-2008* (pp. 343–385). Springer.

Cousot, P., & Cousot, R. (1977). Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th acm sigact-sigplan symposium on principles of programming languages* (pp. 238–252).

Cousot, P., & Halbwachs, N. (1978). Automatic discovery of linear restraints among variables of a program. In *Proceedings of the 5th acm sigact-sigplan symposium on principles of programming languages* (pp. 84–96).

Douc, R., & Cappé, O. (2005). Comparison of resampling schemes for particle filtering. In *Ispa 2005. proceedings of the 4th international symposium on image and signal processing and analysis, 2005.* (pp. 64–69).

Doucet, A., Johansen, A. M., et al. (2009). A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of nonlinear filtering*, *12*(656-704), 3.

Evensen, G. (2003). The ensemble kalman filter: Theoretical formulation and practical implementation. *Ocean dynamics*, *53*(4), 343–367.

Finke, A., Doucet, A., & Johansen, A. M. (2020). Limit theorems for sequential mcmc methods. *Advances in Applied Probability*, *52*(2), 377–403.

Fukuda, K. (2020). Polyhedral computation.
doi: https://doi.org/10.3929/ethz-b-000426218

Gelman, A., Carlin, J. B., Stern, H. S., & Rubin, D. B. (2013). *Bayesian data analysis* (3rd ed.). Retrieved 06/01/22, from `http://www.stat.columbia.edu/~gelman/book/`

Gelman, A., & Rubin, D. B. (1992). Inference from iterative simulation using multiple sequences. *Statistical science*, *7*(4), 457–472.

Gilks, W. R., & Berzuini, C. (2001). Following a moving target—monte carlo inference for dynamic bayesian models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *63*(1), 127–146.

Gordon, N. J., Salmond, D. J., & Smith, A. F. (1993). Novel approach to nonlinear/non-gaussian bayesian state estimation. In *Iee proceedings f-radar and signal processing* (Vol. 140, pp. 107–113).

Gurfinkel, A., & Navas, J. A. (2021). Abstract interpretation of LLVM with a region-based memory model. In *Software verification - 13th international conference, VSTTE 2021, and 14th international workshop, NSV 2021, october 18-19, 2021, revised selected papers.*

Halbwachs, N., Proy, Y.-E., & Roumanoff, P. (1997). Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, *11*(2), 157–185.

Han, C., & Carlin, B. P. (2001). Markov chain monte carlo methods for computing bayes factors: A comparative review. *Journal of the American Statistical Association*, *96*(455), 1122–1132.

Henry, J., Monniaux, D., & Moy, M. (2012). Pagai: A path sensitive static analyser. *Electronic Notes in Theoretical Computer Science*, *289*, 15–25.

Jeannet, B., & Miné, A. (2009). Apron: A library of numerical abstract domains for static analysis. In *International conference on computer aided verification* (pp. 661–667).

Kalnay, E. (2003). *Atmospheric Modeling, Data Assimilation and Predictability*. Cambridge University Press.

Khan, Z., Balch, T., & Dellaert, F. (2003). Efficient particle filter-based tracking of multiple interacting targets using an mrf-based motion model. In *Proceedings 2003 ieee/rsj international conference on intelligent robots and systems (iros 2003)(cat. no. 03ch37453)* (Vol. 1, pp. 254–259).

Kieu, L.-M., Malleson, N., & Heppenstall, A. (2020). Dealing with uncertainty in agent-based models for short-term predictions. *Royal Society Open Science*, *7*(1), 191074. doi: 10.1098/rsos.191074

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *science*, *220*(4598), 671–680.

Lewis, J. M., Lakshmivarahan, S., & Dhall, S. (2006). *Dynamic Data Assimilation: A Least Squares Approach*. Cambridge: Cambridge University Press.

Li, T., Sun, S., Sattar, T. P., & Corchado, J. M. (2014). Fight sample degeneracy and impoverishment in particle filters: A review of intelligent approaches. *Expert Systems with applications*, *41*(8), 3944–3954.

Liao, C., & Barooah, P. (2010). An integrated approach to occupancy modeling and estimation in commercial buildings. In *Proceedings of the 2010 american control conference* (pp. 3130–3135).

Liu, J., & West, M. (2001). Combined parameter and state estimation in simulation-based filtering. In *Sequential monte carlo methods in practice* (pp. 197–223). Springer.

Lloyd, D. J. B., Santitissadeekorn, N., & Short, M. B. (2016). Exploring data assimilation and forecasting issues for an urban crime model. *European Journal of Applied Mathematics*, *27*(Special Issue 03), 451–478. doi: 10.1017/S0956792515000625

Lueck, J., Rife, J. H., Swarup, S., & Uddin, N. (2019). Who Goes There? Using an Agent-based Simulation for Tracking Population Movement. In *Winter Simulation Conference, Dec 8 - 11, 2019.* National Harbor, MD, USA.

Malleson, N., Minors, K., Kieu, L.-M., Ward, J. A., West, A., & Heppenstall, A. (2020). Simulating crowds in real time with agent-based modelling and a particle filter. *Journal of Artificial Societies and Social Simulation*, *23*(3), 3. doi: 10.18564/jasss.4266

Markowitz, H. M. (1957). The elimination form of the inverse and its application to linear programming. *Management Science*, *3*(3), 255–269.

Maros, I. (2002). *Computational techniques of the simplex method* (Vol. 61). Springer Science & Business Media.

Meel, K. S., Vardi, M. Y., Chakraborty, S., Fremont, D. J., Seshia, S. A., Fried, D., ... Malik, S. (2016). Constrained sampling and counting: Universal hashing meets sat solving. In *Workshops at the thirtieth aaai conference on artificial intelligence.*

Mihelich, M., Dubrulle, B., Paillard, D., Kral, Q., & Faranda, D. (2018). Maximum kolmogorov-sinai entropy versus minimum mixing time in markov chains. *Journal of Statistical Physics*, *170*(1), 62–68.

Motzkin, T. S., Raiffa, H., Thompson, G. L., & Thrall, R. M. (1953). The double description method. *Contributions to the Theory of Games*, *2*(28), 51–73.

Muñoz-Pichardo, J., Pino-Mejías, R., García-Heras, J., Ruiz-Muñoz, F., & Luz González-Regalado, M. (2021). A multivariate poisson regression model for count data. *Journal of Applied Statistics*, *48*(13-15), 2525–2541.

Newton, M. A., & Raftery, A. E. (1994). Approximate bayesian inference with the weighted likelihood bootstrap. *Journal of the Royal Statistical Society: Series B (Methodological)*, *56*(1), 3–26.

Schelling, T. C. (1971). Dynamic models of segregation. *Journal of mathematical sociology*, *1*(2), 143–186.

Septier, F., Pang, S. K., Carmi, A., & Godsill, S. (2009). On mcmc-based particle methods for bayesian filtering: Application to multitarget tracking. In *3rd ieee international workshop on computational advances in multi-sensor adaptive processing (camsap)* (pp. 360–363).

Sjödin, J., Pop, S., Jagasia, H., Grosser, T., Pop, A., et al. (2009). Design of graphite and the polyhedral compilation package. In *Gcc developers' summit* (p. 113).

Štefankovič, D., Vempala, S., & Vigoda, E. (2009). Adaptive simulated annealing: A near-optimal connection between sampling and counting. *Journal of the ACM (JACM)*, *56*(3), 1–36.

Suhl, U. H., & Suhl, L. M. (1990). Computing sparse lu factorizations for large-scale linear programming bases. *ORSA Journal on Computing*, *2*(4), 325–335.

Talagrand, O. (1997). Assimilation of Observations, an Introduction. *Journal of the Meteorological Society of Japan. Ser. II*, *75*(1B), 191–209.

Tang, D. (2019). Data assimilation in agent-based models using creation and annihilation operators. *arXiv preprint arXiv:1910.09442*. Retrieved from `https://arxiv.org/abs/1910.09442`

Tang, D. (2022). *Mutable categorical distribution.* GitHub. Retrieved 06/01/22, from `https://github.com/danftang/MutableCategoricalDistribution`

Wan, E. A., Van Der Merwe, R., & Haykin, S. (2001). The unscented kalman filter. *Kalman filtering and neural networks*, *5*(2007), 221–280.

Wang, M., & Hu, X. (2015). Data assimilation in agent based simulation of smart environments using particle filters. *Simulation Modelling Practice and Theory*, *56*, 36–54. doi: 10.1016/j.simpat.2015.05.001

Ward, J. A., Evans, A. J., & Malleson, N. S. (2016). Dynamic calibration of agent-based models using data assimilation. *Royal Society Open Science*, *3*(4). doi: 10.1098/rsos.150703