SAMPLING FROM THE BAYESIAN POSTER OF AN AGENT-BASED MODEL GIVEN PARTIAL OBSERVATIONS

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

January 22, 2021

ABSTRACT

abstract

***K*eywords** Data assimilation, Bayesian inference, Agent based model, Integer linear programming, predator prey model

1 Introduction

Suppose we have a timestepping ABM where agents have a finite number of possible internal states and a finite number of ways of acting on their world. Given this, we can define an ABM as:

- A domain of agent actions $\mathcal{A} = \{a_0...a_n\}$
- A domain of agent states $\mathcal{S} = \{\sigma_0...\sigma_m\}$
- An agent timestep function, $a = \pi(\psi, \Psi)$, which is a computer program whose input is the agent's state $\psi \in \mathcal{S}$ and a (sparse) vector, Ψ, whose elements ψ_i are the number of agents in state σ_i in the model, and whose output is a chosen action $a \in \mathcal{A}$. The program may make calls to a random number generator, so the function is stochastic and we write $P(\pi(\psi, \Psi) = a)$ to be the probability that the program will return action a, given inputs ψ and Ψ.
- An agent transition tensor $F_\phi^{\psi a}$ whose elements give the number of agents in state ϕ that result from an agent in state ψ performing act a (including the acting agent).

Let a model timestep consist of a matrix E whose elements $e_{\psi a}$ are the number of agents in state ψ that perform act a in this timestep. Similarly, let a model trajectory be a tensor $T^t_{\psi a}$ whose elements are the number of agents in state ψ that perform act a in timestep t.

A trajectory must satisfy a number of constraints in order to be a possible trajectory of an ABM. Since the elements of a trajectory are counts, they must all be non-negative integers. So we have the *non-negative integer constraint* that

$$\forall t, \psi, a : T^t_{\psi a} \in \mathbb{Z}_{\geq 0} \tag{1}$$

A trajectory, $T^t_{\psi a}$, must also be *continuous* which means that each agent that results from the actions of an agent in timestep t must appear in the actions of timestep $t + 1$. This can be expressed in terms of the *continuity constraints*:

$$\forall t \in 1...n : \forall \phi : \sum_{\psi, a} F_\phi^{\psi a} T^{t-1}_{\psi a} - \sum_a T^t_{\phi a} = 0 \tag{2}$$

If a trajectory satisfies constraints 1 and 2 we say that it is *valid*.

[*]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)

Suppose we have a prior belief, $P_0(\Psi)$, over the state of the ABM at time $t = 0$. The prior probability of a trajectory is then given by

$$P(T^t_{\psi a}) = \begin{cases} P_0(\sum_a T^0_{\psi a}) \prod_{t,\psi,a} P(\pi(\psi, \sum_b T^t_{\phi b}) = a)^{T^t_{\psi a}} & \text{if } T^t_{\psi a} \text{ is valid} \\ 0 & \text{otherwise} \end{cases}$$

Suppose we also have a set of noisy, aggregate observations, Ω and an easily computed likelihood function $P(\Omega|T^t_{\psi a})$. Our aim is to generate a number of samples from the posterior distribution

$$P(T^t_{\psi a}|\Omega) \propto P\left(\Omega|T^t_{\psi a}\right) P(T^t_{\psi a})$$

We assume that each observation is independent of the others given the trajectory, so that

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

Also, for notational convenience, we express our prior belief in terms of a set of observations, Ω_0, at $t = 0$, so that $P_0(\sum_a T^0_{\psi a}) \propto \prod_{(\omega,v)\in\Omega_0} P(\omega = v|T^t_{\psi a})$ (the same can be done for any boundary conditions that aren't at $t = 0$). The posterior can now be written as

$$P(T^t_{\psi a}|\Omega) \propto \begin{cases} \prod_{(\omega,v)\in\Omega} P\left(\omega = v\middle|T^t_{\psi a}\right) \prod_{t,\psi,a} P(\pi(\psi, \sum_b T^t_{\phi b}) = a)^{T^t_{\psi a}} & \text{if } T^t_{\psi a} \text{ is valid} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

This posterior has zero probability for the vast majority of trajectories because most trajectories are invalid, contain an impossible action or are refuted by the observations. This makes sampling from this distribution vary difficult in practice. For example, even though we can generate samples from the prior by simply performing a forward execution of the model, it is often very unlikely that this sample will fit the observations, so techniques such as rejection sampling are not practical.

Our strategy in this paper will be to use Markov Chain Monte-Carlo sampling. However, this requires a proposal function which generates a new sample from the current one. This is often done by randomly perturbing the current sample. However, if we randomly perturb an ABM trajectory it's very unlikely that we'll end up with a trajectory that is valid, contains only possible actions and satisfies the observations. In practice, then, a random perturbation would almost always be rejected and we'd probably end up stuck on the initial sample until we grew old.

2 Approximating the support of the posterior

In order to solve this problem, we'll make the states of our Markov chain an approximation of the support of the posterior, $P(T^t_{\psi a}|\Omega)$, (i.e. the set of trajectories that have non-zero probability) and restrict our proposal function to perturbations that are in this approximation.

If we let

$$P^t_{\pi\xi c}(T^s_{\psi a}) = P(\pi(\xi, \sum_b T^t_{\phi b}) = c)^{T^t_{\psi a}}$$

Then from equation 3

$$\text{supp}(P(\,.\,|\Omega)) = \bigcap_{(\omega,v)\in\Omega} \text{supp}(P(\omega = v \mid .)) \cap \bigcap_{t,\xi,c} \text{supp}(P^t_{\pi\xi c}) \tag{4}$$

i.e. in order for the posterior to be non-zero, all the observation likelihoods must be non-zero and the probability of each agent's action at every timestep must be non-zero.

In order to approximate this support, we first express each likelihood function $P(\omega = v|T^t_{\psi a})$ as a computer program $\omega(T^t_{\psi a})$ whose input is a trajectory and whose output is the value of the observation. This program may make calls to a random number generator, so the function is stochastic and we write $P(\omega(T^t_{\psi a}) = v)$ to denote the probability that $\omega(T^t_{\psi a})$ returns a value v. So equation 4 now consists of the intersection of terms of the form $\text{supp}(P(\rho(.) = y))$ where ρ is a computer program whose input is a trajectory and y is a given output. So, we need a way to find, or approximate, the set of inputs of a computer program that would produce a given output, and to take the intersection of these sets. Our strategy here is to calculate a set of linear inequalities, $\mathcal{P}$, on trajectories that are satisfied for all x such that $\rho(x)$ might return y. Note that the converse is not necessarily true; if x satisfies $\mathcal{P}$ then it might not be possible for $\rho(x)$ to return y. So $\mathcal{P}$ can be thought of as an approximation of $\text{supp}(P(\rho(.) = y))$ in that it may include some extra members. However, for our purposes, a certain amount of approximation is tolerable and won't result in any approximation in the Markov Chain. If we happen to choose a proposal from $\mathcal{P}$ that isn't in $\text{supp}(P(\rho(.) = y))$ then it will just be rejected. This is fine as long as we generate acceptable proposals at a reasonable rate.

A computer program can be transformed into a set of linear inequalities using a technique known as *abstract interpretation*(Cousot & Cousot, 1977). Tools to do this already exist (e.g. PAGAI(Henry, Monniaux, & Moy, 2012)) and are used widely in applications such as the verification of safety critical systems(Blanchet et al., 2003). For our application we choose the abstract domain of convex polyhedra(Cousot & Halbwachs, 1978)(Becchi & Zaffanella, 2018). Calls to a random number generator are dealt with, without loss of generality, by providing a single function `rand()` that returns a floating point number between 0 and 1 with uniform probability. In the abstract domain, this is equivalent to the generation of a new variable, r, that satisfies $0 \leq r < 1$.

Once we have the linear approximations of the support of each program, the intersection of these supports is just the set of trajectories that satisfy all of the inequalities. So, if we let $\mathcal{P}(\rho(.) = v)$ be the set of inequalities which approximate $\text{supp}(P(\rho(.) = v))$ then we have the *support inequalities*

$$\text{supp}(P(P(.|\Omega))) \subset \bigcap_{(\omega,v)\in\Omega} \mathcal{P}(\omega(.) = v) \cap \bigcap_{t,\xi,c} \mathcal{P}(\pi'(\xi, t, .) = c) \tag{5}$$

where

$$\pi'(\xi, s, T^t_{\psi a}) = \pi(\xi, \sum_b T^s_{\phi b})$$

In order to ensure the trajectory is valid, we also intersect with the constraints in 2, which are also linear. We can express the non-negative part of equation 1 as the linear constraints

$$\forall t, \psi, a : T^t_{\psi a} \geq 0. \tag{6}$$

The integer part we will consider in the next section.

3 From convex polyhedron to Markov process

Once we've generated a set of inequalities that contain the support of the posterior, we next need a proposal function that will allow us to transition between trajectories in a Markov chain, while ensuring the inequalities remain satisfied.

3.1 A proposal function for a Fermionic ABM

We define a Fermionic ABM to be one where no two agents can share the same state at the same time. i.e. in addition to the validity constraints of equation 1 and 2 the trajectory of a Fermionic ABM also satisfies the *Fermionic constraint*

$$\forall t, \psi : \sum_a T^t_{\psi a} \leq 1 \tag{7}$$

in which case we call it a *Fermionic trajectory*.

Theorem 1. *All valid Fermionic trajectories are on the vertices of the convex polyhedron described by equations 2, 6, 7 and 5.*

Proof. This can be seen immediately since equations 6 and 7 describe a unit hupercube, so all integer solutions are on vertices. □

Theorem 2. *For any pair of valid trajectories, A and B, there exists a path along edges of the polyhedron from A to B that only traverses through integer solutions.*

Proof. [Take the difference A-B and decompose into integer null vectors (loops)]

□

Since all Fermionic trajectories are on vertices, one possibility is to make a Markov process whose states are the vertices of the polyhedron and whose transitions are the edges. This would be computationally convenient as navigation along edges of a polyhedron can be achieved efficiently by using the pivoting operation of the Simplex algorithm(Dantzig, Orden, Wolfe, et al., 1955)(Thie & Keough, 2011). Given a system of m linear equality constraints on n variables (where any inequalities are turned into equalities by the addition of slack variables), we choose m *basic variables* which may be non-zero and constrain all *non-basic variables* to zero. This induces a unique solution which can be shown to be a vertex of the original polyhedron. A *pivot* involves swapping one of the basic variables for one of the non-basic variables so the set of basic variables loses one of its members and gains a new member from the set of non-basic variables. This induces a new solution, and new vertex which is adjacent to the original vertex in the sense that there is an edge connecting the two.

However, the use of pivoting is complicated somewhat when the vertices of the polyhedron are degenerate. A degenerate vertex is one that can be generated by more than one set of basic variables. This can happen when any of the basic variables are zero. This can be seen by noting that pivoting-out a basic variable has the effect of constraining its value to zero, but if it's already zero the constraint is already satisfied and the solution does not change. If there are many zero-valued basic variables there may be a large number of pivots that do not change the solution. In full generality, a degenerate vertex is associated with a "degeneracy graph"(Zörnig, 1993) whose vertices are the sets of basic variables that generate a solution at that vertex and whose edges are the pivots.

We'd like to somehow embed these degeneracy graphs into our Markov process in a computationally efficient way. Mapping each node on the degeneracy graph to a state in the Markov process is made difficult because we would also need to assign a probability to each node such that the sum of all node probabilities is the probability of the vertex. However, calculating the number of nodes in a degeneracy graph is computationally intensive(Zörnig, 1993) so it's not clear how we should assign probabilities. This problem would not arise if we could have a single Markov state represent the whole degeneracy graph, however, we would then have some way of choosing an edge of the vertex with a known probability, which is also potentially computationally costly(Gal & Geue, 1992)(Yamada, Yoruzuya, & Kataoka, 1994) as it may require large changes in the state, which may be quite large.

We propose a computationally efficient solution based on the observation in (Yamada et al., 1994) that the neighbours of a vertex can be described as the extreme points of the polyhedron

$$\begin{aligned} A_1 y &\leq 0, \\ \sum_i y_i &= 1, \\ y &\geq 0 \end{aligned}$$

where A_1 is the matrix formed from the rows of the simplex tableau that are equal to zero in the b column and contain at least one positive coefficient. We'll call this the *neighbourhood* polyhedron.

To turn this into a Markov process, begin with a polyhedron with m constraints on n variables. Consider now a vertex of that polyhedron with σ zero-valued basic variables. Suppose out of those σ basic variables, m_1 of them are on rows in the tableaux that have at least one positive coefficient (i.e. A_1 has m_1 rows)[2]. Assign to that vertex a Markov state for each of the $\binom{n-m+\sigma}{m_1}$ sets of m_1 basic variables chosen from the $n-m+\sigma$ zero-valued variables; call this the *neighbourhood state*. Note that it is clear that the neighbourhood polyhedron cannot have more vertices than there are neighbourhood states, but it may have fewer. Let $P(v)$ be the desired probability of the vertex and assign each Markov state a probability of

$$P(s) = \frac{P(v)}{\binom{n-m+\sigma}{m_1}}.$$

This assures that the sum of the probabilities of all possible neighbourhood states assigned to a vertex is the desired probability of the vertex.

In order to generate proposal transitions for each state, first decide with fixed probability, α, whether to make a transition to another state in the same vertex or a neighbouring vertex.

If we decide to transition to another state inside the same vertex then simply choose, with uniform probability, a member of the current neighbourhood state and swap it for a non-basic variable, again chosen uniformly. Since the proposed state has the same probability by definition, and the probability of making the reverse transition is the same as making the forward transition, then the proposal is always accepted.

If we choose to transition to a neighbouring vertex, first map the neighbourhood state, μ, to a valid set of basic variables B for the neighbourhood polyhedron using the following algorithm:

```
$P \leftarrow$ simplex tableau for the neighbourhood polyhedron
$Q \leftarrow \emptyset$
for $i = 0$ to $m$ do
    $q \leftarrow m_i$
    while the $q^{th}$ column of $P$ is zero on all rows $k \notin Q$ do
        $q \leftarrow (q+1) \mod (n-m+\sigma)$
    end while
    pivot $P$ on any non-zero row, $k \notin Q$, of the $q^{th}$ column
    add $q$ to $Q$
end for
```

Note that this terminates as long as A_1 is of full rank, which is guaranteed as long as our original set of constraints has full rank.

[need to prove that this is a unique mapping for all start states]

Once we have a tableau of the neighbourhood polyhedron in it's mapped pivot state, there will be exactly one row which hasn't been pivoted by the above algorithm (since there will be $m_1 + 1$ constraints, the extra one being the $\sum_i y_i = 1$

[2][need to prove this is constant for all nodes in the degeneracy graph]

constraint). Each possible pivot point on this row identifies a vertex of the neighbourhood polyhedron, which itself maps to a possible neighbour of the current vertex[3]. So, we take this set to be the set of possible destination vertices.

Given a destination vertex, we also need to choose a neighbourhood state at the destination. For this, we choose a state such that the current pivot state can be retained as much as possible, so as to reduce the computation needed to do the transition. To do this, we start with the current neighbourhood state and remove any variables that are non-zero in the destination. We then add/delete variables, chosen at random with a uniform probability, until we reach the correct number of members for the new value of m_1.

So, if r_x of the members of the source neighbourhood state are non-zero in the destination then and the source and destination states have σ_x and σ_y zero-valued basic variables respectively, and the size of the neighbourhood state in the source and destination is m_{1x} and m_{1y} respectively, then we need to choose $m_{1y} - m_{1x} + r_x$ new variables out of a total of $n - m + \sigma_y + m_{1x} - r_x$. So, the acceptance probability is given by

$$\alpha = \min\left(1, \frac{P(Y)P_{Y\to X}\binom{n-m+\sigma_x}{m_{1x}}\binom{n-m+\sigma_y+m_{1x}-r_x}{m_{1y}-m_{1x}+r_x}}{P(X)P_{X\to Y}\binom{n-m+\sigma_y}{m_{1y}}\binom{n-m+\sigma_x+m_{1y}-r_y}{m_{1x}-m_{1y}+r_y}}\right)$$

So, in order to make this as close to 1 as possible, we choose a destination, from the set of possible pivots, with probability proportional to

$$P_{X\to Y} \propto \frac{P(Y)\binom{n-m+\sigma_y+m_{1x}-r_x}{m_{1y}-m_{1x}+r_x}}{\binom{n-m+\sigma_y}{m_{1y}}}$$

4 Notes

To prove that a vertex is a valid Fermionic trajecotry...

We first show that if all variables $\tau^{t-1}_{\phi a}$, at time $t-1$, are non-negative integer on a vertex, then all variables $\tau^t_{\phi a}$, at time t, are also non-negative integer.

Since by conjecture all $\tau^{t-1}_{\phi a}$ are non-negative, integer and all $f^{\psi a}_{\phi}$ are, by definition, also non-negative integer, it follows that

$$\sum_{\psi,a} f^{\psi a}_{\phi}\tau^{t-1}_{\psi a} \in \mathbb{Z}_{\geq 0}$$

Substituting 7 into 2 gives

$$\sum_{\psi,a} f^{\psi a}_{\phi}\tau^{t-1}_{\psi a} \leq 1$$

so

$$\sum_{\psi,a} f^{\psi a}_{\phi}\tau^{t-1}_{\psi a} \in \{1,0\}$$

substituting back into 2 gives

$$\sum_{a} \tau^{t}_{\phi a} \in \{0,1\}$$

[starting with start state, this can take any binary value, now show that the first timestep is integer and induce for other timesteps. Perhaps add actions in groups that share the same start-state and show that vertices remain integer in the higher dimensional state. This can be shown more easily by introducing an auxiliary variable that is the number of agents in a given (phi,t) state]

[start with a fractional solution, recursively remove valid agent trajectories until there are non left, proving that it was a sum of valid trajectories][since all coefficients start at value 1, all]

[show that a valid trajectory, once pivoted in, leaves all coefficients 1, so all neighbours of all valid trajectories are valid trajectories, so all vertices are valid. Since, before any pivots, each column has exactly one +ve value (i.e. the source of the action edge) and in a continuous trajectory there is exactly one action for each (state,time) pair and *the connectivity graph is a tree*]

[choose the smallest fractional value and subtract that weight of the integer solution that has maximum overlap]

[With just agent continuity constraints, we have a cone whose vertex is at zero and whose beams are the trajectories of single agents. Show that addition of integer upper bounds can only add corners at integer solutions, i.e. show that the intersection of the hypercube with the beams occurs at the vertices of the hypercube]

[]

[3]We can prove that this row isn't equal to zero by disallowing the $\sum_i y_i = 1$ row from being pivoted-in during the mapping from μ to B

4.1 Agent policy constraints

Let a policy be defined as a computer program whose input is ψ, the current state of the agent, and Ψ a sparse vector consisting of the number of agents in other states, and whose return value, Φ, is a vector consisting of the number of agents in each state resulting from the actions of this agent at the end of this timestep (including its own state, if applicable). The agent can die (set to dead state or don't include in return state), set its state or create another agent in a given state. There is also a Filp function which takes a probability and returns a Boolean whose probability of being true is the supplied probability.

The interface to Ψ consists of $occupation(\phi)$ which returns the occupation number of state ϕ. Occupation numbers can be added, subtracted, negated, multiplied by an integer and added to an integer to give another occupation number. There is a greater-than-or-equal-to operator that that compares with an integer to give a linear Boolean expression. Logical operations between linear Boolean expressions and local variables or expressions on ψ will reduce to logical operations on true or false values.

The "support semantics" of the program gives the actions that are possible given a known ψ but an unknown Ψ, where each path through the program is considered a separate action. The support semantics, for a given ψ, is a set of (a, B, v) pairs where a is a string of '0's and '1's denoting the direction of program flow at each branch, B is a Boolean expression on linear inequalities on Ψ for which a is possible if B is satisfied, and v is an assignment of local variables to values (in the case of occupation number type variables, these will be linear expressions) which give the values of any variables that remain in scope after execution (possibly as a function of local variables in-scope at the start of execution).

Let $v(a)$ be the assignment to local variables at a given point in the program, given an execution along a. [Let $P(a, B)$ be the probability of following a given that B is satisfied.]

Let $S \otimes T$ be the set $\{(a + a', B \wedge B', v' \otimes v) : (a, B, v) \in S \wedge (a', B', v') \in T \wedge (B \wedge B' \neq false)\}$ where $a + a'$ is interpreted as concatenation and $(v' \otimes v)(x) = v'(v(x))$.

The semantics of two consecutive statements with semantics A and B is $A \otimes B$.

A raw Boolean expression (i.e. as found in an if statement/loop guard) can be converted into a Boolean solely on Ψ, given values of any other local variables, by substituting the variables for their values. We write this, for convenience, as $C(v)$

b, is a set of (a, B) pairs representing the disjunction of its members. A linear Boolean expressions on Ψ is just $\{(\emptyset, B)\}$ where $\emptyset$ is the empty string. `b && Flip(p)` means $\{(1 + a, B) : (a, B) \in b\}$, `B || Flip(p)` means $\{(\emptyset, B), (1, true)\}$.

The negation of a raw Boolean, b, is $\{(\arg\min_{\{\overline{a}:(a,B)\in b\}} |a|, \bigwedge_{(a,B)\in b} \overline{B})\}$ if $\bigwedge_{(a,B)\in b} \overline{B} \neq false$ and all $\{a : (a, B) \in b\}$ agree (i.e. they are all prefixes of the longest member), or the empty set otherwise. The negation of a Flip string $\overline{a}$ is the string with all

The semantics of an if statement on a raw Boolean expression, B, is $\{(1, B)\} \otimes D \cup \{(0, \overline{B})\} \otimes E$ where B is the Boolean expression, D is the semantics of the if body and E is the semantics of the else body.

The semantics of a loops whose guard contains an expression on Ψ is the fix-point of

$$S = S \otimes \left(\{(0, \overline{G})\} \cup \{(1, G)\} \otimes D\right)$$

[The guard here may depend on program variables, so to get a fix-point they need to be part of the semantics. Better to just go around the loop collecting paths until the path becomes impossible]

[local variables may get entangled with Ψ so need to be included in the semantics. e.g. if(Psi.occupation(1)) then x = 1 else x = 2. In which case, the semantics needs to be a function from paths to joint assignments to variables and Boolean expressions on Psi (and perhaps probabilities)]

References

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

Blanchet, B., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., … Rival, X. (2003). A static analyzer for large safety-critical software. In *Proceedings of the acm sigplan 2003 conference on programming language design and implementation* (pp. 196–207).

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).

Dantzig, G. B., Orden, A., Wolfe, P., et al. (1955). The generalized simplex method for minimizing a linear form under linear inequality restraints. *Pacific Journal of Mathematics*, *5*(2), 183–195.

Gal, T., & Geue, F. (1992). A new pivoting rule for solving various degeneracy problems. *Operations Research Letters*, *11*(1), 23–32.

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

Thie, P. R., & Keough, G. E. (2011). *An introduction to linear programming and game theory*. John Wiley & Sons.

Yamada, T., Yoruzuya, J., & Kataoka, S. (1994). Enumerating extreme points of a highly degenerate polytope. *Computers & operations research*, *21*(4), 397–410.

Zörnig, P. (1993). A theory of degeneracy graphs. *Ann Oper Res*, *46*, 541–556. doi: 10.1007/BF02023113