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

February 2, 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 $\mathcal{S} = \{\sigma_0 ... \sigma_m\}$
- A computer program, $\pi(\psi, \Psi)$, where $\psi \in \mathcal{S}$ is the agent's state and Ψ is a (sparse) vector whose i^{th} element is the number of agents in state σ_i at the start of the timestep. The program can make calls to a random number generator, so its returned value may be non-deterministic and we write $P(\pi(\psi, \Psi) = a)$ to be the probability that the program will return value a, given inputs ψ and Ψ. The program returns an action $a \in \mathcal{A}$ which is the action performed by the agent in this timestep.
- 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).

As a very simple demonstration, we'll take a spatial predator-prey model where agents are either predator or prey moving about a 2-dimensional grid. In this case, the domain of actions are $\mathbb{A} = \{$move-left, move-right, move-up, move-down, reproduce up, reproduce down, reporduce left, reproduce right, be eaten, die$\}$. So the domain of agent states, $\mathcal{S}$, can be thought of as the class

```
class Agent {
        Boolean isaPredator
        Integer xPosition
        Integer yPosition
}
```

We can assign an ordering of agent states by placing agent `a` in the (`a.isaPredator*G`^{2} `+ a.yPositition*G + a.xPosition`)th position, where `G` is the size of the grid.

The timestep function for a predator/prey agent is shown in algorithm 1 (where we assume the grid has periodic boundary conditions so we don't need to worry about its edges). The transition tensor encodes the results of the various actions. So, for example, state σ_0 corresponds to a prey at position (0,0). If this agent moves right (i.e. performs action a_1) then the result is a prey at position (1,0) (i.e. in state σ_1) so $F_1^{01} = 1$ etc.

*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)

Algorithm 1 Timestep of a predator/prey agent

```
function TIMESTEP(agent, otherAgents)
    if agent.isaPredator then
        if RANDOM( ) < β0 and number of prey on neighbouring gridsquares of otherAgents > 0 then
            return reproduce left, right, up or down based on a call to RANDOM( )
        end if
        if RANDOM( ) < γ0 then
            return die
        end if
    else
        if RANDOM( ) < β1 then
            return reproduce left, right, up or down based on a call to RANDOM( )
        end if
        if RANDOM( ) < γ1 then
            return die
        end if
        if RANDOM( ) < δ and number of predators on this or neighbouring gridsquares of otherAgents > 0 then
            return be eaten
        end if
    end if
    return move left, right, up or down based on a call to RANDOM( )
end function
```

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, we'll call this the *non-negative integer constraint*

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

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*:

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

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

Suppose we have a prior belief, $P_0(\Psi)$, over the state of the ABM at time $t = 0$ (in our predator/prey example, this may consist, for example, of some knowledge about the initial population distribution). 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, with a likelihood function $P(\Omega|T^t_{\psi a})$. In our running example, let's suppose we note down when and where we see footprints of predator and prey. Our aim is to generate a number of samples from the posterior distribution

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

We assume that observations are independent given the trajectory and that each individual observation in Ω can be simulated by a computer program $\omega(T^t_{\psi a})$ which may make calls to a random number generator. The probability, $P(\omega(T^t_{\psi a}) = v)$, that the program returns a value v defines the likelihood of observing value v upon making observation ω on trajectory $T^t_{\psi a}$. i.e. $\omega(T^t_{\psi a})$ is just a simulation of the observation, which is usually quite easy to write and to compute.

In the case of the predator prey model, let's suppose at the end of every day we go out to a fixed set of inspection sites and check for footprints. Suppose that if predators were present in the grid-square containing an inspection site on that day, there's a 50% chance we'll see a footprint, and if prey were present, there's a 25% chance. The observation function for a site is shown in algorithm 2

Algorithm 2 Observation function for predator/prey footprints

function OBSERVATION(T) ▷ T is the ABM trajectory
  $t \leftarrow$ time of observation
  $x \leftarrow$ x-position of observation
  $y \leftarrow$ y-position of observation
  $G \leftarrow$ size of grid
  $\psi \leftarrow G^2 + Gy + x$ ▷ state of predator in this gridsquare
  $\phi \leftarrow Gy + x$ ▷ state of prey in this gridsquare
  $F_{pred} \leftarrow$ FALSE ▷ did we observe predator footprints?
  $F_{prey} \leftarrow$ FALSE ▷ did we observe prey footprints?
  if $\sum_a T^t_{\psi a} > 0$ and RANDOM( ) < 0.5 **then**
    $F_{pred} \leftarrow$ TRUE
  end if
  if $\sum_a T^t_{\phi a} > 0$ and RANDOM( ) < 0.25 **then**
    $F_{prey} \leftarrow$ TRUE
  end if
  return F_{pred}, F_{prey}
end function

Given this

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

Also, for notational convenience, we express our prior beliefs 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 priors that aren't at $t = 0$). The posterior can now be written as

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

where Ω now includes any priors.

In many practical applications, sampling from this posterior is difficult because it has zero probability for the vast majority of trajectories (i.e. most trajectories are invalid, contain an impossible action or are refuted by the observations). 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 simple techniques such as rejection sampling are not practical.

Our strategy in this paper will be to use Markov Chain Monte-Carlo sampling. However, this method requires a proposal function which randomly generates the next sample given the current one. This suffers from the same problem that most trajectories have zero probability. For example if we generate a new sample by perturbing one or more elements of the last sample at random, it's very unlikely that we'll end up with a trajectory that is valid, contains only possible actions and satisfies the observations. So, the proposed next sample would almost certainly be rejected and we'd probably end up stuck on the first sample until we grew old.

2 Approximating the support of the posterior

In order to solve this problem, we'll make 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

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

Then from equation 3

$$\text{supp}(P(\,.\,|\Omega)) = \bigcap_{(\omega,v)\in\Omega} \text{supp}\left(P\left(\omega(.) = v\right)\right) \cap \bigcap_{t,\psi,a} \text{supp}\left(P\left(\pi'(t,\psi,.) = a\right)\right) \cap \left\{T^t_{\psi a} \mid T^t_{\psi a} \text{is valid}\right\} \tag{4}$$

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.

The first two terms in equation 4 consists of the supports of various computer programs whose inputs are ABM trajectories and whose outputs are given. So, we need a way to find, or approximate, the set of trajectories that, when

passed to a computer program, would produce a given output; then we need 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 $P(\rho(x) = y) > 0$. 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 in that it is a superset of $\mathrm{supp}(P(\rho(.) = y))$. However, for our purposes, this approximation won't result in any approximation in the Markov Chain. If we happen to choose a proposal from $\mathcal{P}$ that isn't in $\mathrm{supp}(P(\rho(.) = y))$ 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 $\mathrm{supp}(P(\rho(.) = v))$ then we have, from equation 4, the *support inequalities*

$$\mathrm{supp}(P(P(.|\Omega))) \subset \bigcap_{(\omega,v)\in\Omega} \mathcal{P}(\omega(.) = v) \cap \bigcap_{t,\psi,a} \mathcal{P}(\pi'(t, \psi, .) = a) \cap \left\{T^t_{\psi a} \mid T^t_{\psi a} \text{is valid}\right\} \tag{5}$$

In order to deal with the final term, which requires the trajectories in the support to be valid, we first intersect with the continuity constraints in 2, which are also linear. Finally we leave the non-negative integer constraints of equation 1 as separate constraints.

Without loss of generality, we express inequalities as equalities by introducing slack variables so that

$$\sum_i c_i x_i \leq y$$

is equivalent to

$$\sum_i c_i x_i + \lambda = y$$
$$\text{subject to } \lambda \geq 0$$

and

$$\sum_i c_i x_i < y$$

is equivalent to

$$\sum_i c_i x_i + \lambda = y$$
$$\text{subject to } \lambda > 0$$

All the constraints can now be gathered into a single matrix equation

$$\begin{aligned} Ax &= y \\ \text{subject to } x_i &\geq 0,\ i \in X_{\geq} \\ x_j &> 0,\ j \in X_{>} \\ x_k &\in \mathbb{Z},\ k \in X_{\geq} \cup X_{>} \end{aligned}$$

where the vector x contains the elements of the trajectory tensor $T^t_{\psi a}$ and the slack variables λ.

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 to form a Markov process whose states emit solutions to the inequalities and whose stationary distribution is the posterior $P(T \mid \Omega)$. I order to perform MCMC we'd like to define the Markov process in terms of a set of states, $\mathbb{S}$, and a stochastic proposal function $f : \mathbb{S} \to \mathbb{S}$ such that $P(f(s_n) = s_{n+1})$ defines the conditional transition probability $P(s_{n+1} \mid s_n)$. The proposal function should have the following properties:

- There should be a set of transitions with non-zero probability between any two states. This ensures proper mixing as time tends to infinity.

- The probability of emitting a trajectory T should be our target posterior probability $P(T \mid \Omega)$, and all emitted trajectories should be solutions of our set of linear equations.
- For any transition from state $s_a \rightarrow s_b$ the probability of transitioning from $s_b \rightarrow s_a$ should be non-zero. This allows us to use the Metropolis Hastings algorithm to attain detailed balance.
- The proposal function should be computationally efficient.

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*

$$\forall t, \psi : \sum_a T^t_{\psi a} \leq 1 \tag{6}$$

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 and 5.*

Proof. This can be seen immediately since equations **??** and 6 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, the obvious way forward would be to define a Markov process whose states, $\mathbb{S}$, are the vertices of the polyhedron and whose transitions are its edges. So, the proposal function would choose from the neighbouring vertices of the current state. There may exist vertices that don't fall on the grid of integer solutions, so are not associated with valid trajectories. However, extra states can be added to a Markov process without affecting the emitted samples as long as the extra states do not emit samples (i.e. the process can pass through these states, but no sample is generated when doing so). However, we don't want to spend too much computational effort moving between non-emitting states, so we should ensure that the probability of being in a non-emitting state isn't much greater than that of being in an emitting state. In order to encourage the Markov process away from these fractional states their probability is multiplied by a "fractional penalty" e^{-kn} where n is the number of non-integer values in the solution and k is a constant parameter.

This approach is 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), let a *pivot state* be a partition of the variables into m *basic variables* and $n - m$ *non-basic variables* such that if we constrain all *non-basic variables* to zero there remains a unique solution. Such solutions can be shown to be at a vertex of the polyhedron described by the constraints. A *pivot* is a perturbation of a pivot state such that one of the basic variables is swapped with 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, such that the perturbed state is also a valid pivot state. It can be shown that every edge of a convex polyhedron corresponds to a unique pivot, so moving along an edge can be achieved computationally by performing a pivot.

This 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 pivot state. This can occur when any of the basic variables are zero, since a pivot has the effect of making a basic variable non-basic, so its value will be constrained to zero; but if the basic-variable's value is zero to start with, constraining it to zero has no effect. If there are many zero-valued basic variables there may be a large number of pivots, and even sequences of pivots, that do not change the solution, and consequently many pivot states which have the same solution. This has the consequence that a degenerate vertex may have a very large number of neighbours, so choosing one at random becomes a non-trivial task since it would be computationally costly to simply enumerate them all and choose one(Gal & Geue, 1992)(Yamada, Yoruzuya, & Kataoka, 1994).

An alternative strategy is to replace the Markov states representing degenerate vertices with multiple states. As long as they all emit the same trajectory and the sum of their probabilities is the probability of the vertex then the Markov process will still emit samples from the posterior.

Algorithm 3 Algorithm to calculate probability of a degeneracy state

$A \leftarrow$ an ordered tableau
$P_T \leftarrow$ the probability of the vertex that is the solution to this tableau
$P_d \leftarrow 1.0$
$\sigma \leftarrow$ the number of degenerate rows in A
for $i = 0 \dots \sigma - 1$ **do**
  $Cols \leftarrow$ the set of columns in A that have at least one non-zero value in rows $i \leq r < \sigma$
  $P_d \leftarrow P_d / |Cols|$
end for
return $P_d \times P_T$

Let an *ordered tableau* be a simplex tableau where all degenerate rows (i.e. rows with value zero) are above all non-degenerate rows, and non-degenerate rows are ordered by the index of the variable that is pivoted-in on that row. We now associate a Markov state with each ordered tableau, assigning it a probability according to algorithm 3:

The transitions between states are the valid pivots and the swapping of position of any two degenerate rows. The proposal function is shown in algorithm 4

Algorithm 4 Proposal function

function PROPOSAL(A)
  $A' \leftarrow A$ (the ordered tableau of the current state)
  $\alpha, \beta \leftarrow$ constant parameters
  $p_0 \leftarrow$ the set of degenerate pivots of A (i.e. the ones that do not change the solution)
  $p_1 \leftarrow$ the set of non-degenerate pivots of A (i.e. the ones that change the solution)
  if BERNOULLI(α) **then**
    $r \leftarrow$ choose member of p_1 with uniform probability
    perform pivot r on A'
  else if BERNOULLI(β) **then**
    $r \leftarrow$ choose member of p_0 with uniform probability
    perform pivot r on A'
  else
    choose two degenerate rows a and b with uniform probability
    swap rows a and b of A'
  end if
  return A'
end function

We need to show that this assignment of probability to ordered tableaus has the property that the sum of the probabilities associated with a vertex is the probability of that vertex.

Theorem 3. *If $\mathbb{E}(T)$ is the set of all ordered tableaus with solution T, and $P(A)$ is the probability assigned to ordered tableau A by algorithm 3 and P_T is the probability assigned to vertex T then*

$$\sum_{A \in \mathbb{E}(T)} P(A) = P_T$$

Proof. Consider the stochastic algorithm 5. It is clear that every possible output is an ordered tableau with solution T and that every ordered tableau with solution T is a possible output of the algorithm. The probability that algorithm 5 outputs tableau A is

$$\frac{1}{\prod_{i=0}^{\sigma-1} C_i^\sigma(A)}$$

where $C_i^\sigma(A)$ is the number of columns that have at least one non-zero value in rows $i \leq r < \sigma$ in A since $C_i^\sigma(A)$ is invariant under pivots on rows $i \leq r < \sigma$. But this is exactly the probability P_d in algorithm 3.

It is also clear that, with probability 1, the algorithm returns an ordered tableau, so the sum of the probabilities of all possible outputs of algorithm 5 must also be 1. So

$$\sum_{A \in \mathbb{E}(T)} P_d(A) = 1$$

but $P(A) = P_d(A) P_T$ so

$$\sum_{A \in \mathbb{E}(T)} P(A) = \sum_{A \in \mathbb{E}(T)} P_d(A) P_T = P_T$$

□

Algorithm 5 Random choice of ordered tableau with solution T

$A \leftarrow$ any ordered tableau with solution T
for $i = 0 \ldots \sigma - 1$ **do**
  $Cols \leftarrow$ the set of columns in A that have at least one non-zero value in rows $i \leq r < \sigma$
  $r \leftarrow$ random member of $Cols$ chosen with uniform probability
  pivot on column r and lowest index row, $i \leq r < \sigma$, of A
  swap the newly pivoted row and the i^{th} row of A
end for
return A

Given this, it's clear that all the necessary properties of the proposal function are fulfilled.

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